An Interactive Community Microgrid Model to aid Design Exploration

A thesis

submitted partial fulfilment

of the requirements for the degree

of

Master of Science (Research) in Computer Science

at

The University of Waikato

by

Liam T. Clow

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2023

Abstract

This dissertation presents a novel approach to microgrid design and exploration through the development of an innovative software application. The primary objective of this research is to empower users to configure and simulate community microgrids, offering multiple time-frame insights into performance metrics, statistical analyses, and pertinent factors aligned with their specific configurations. In collaboration with Ahuora under their third Aim, a key focus is on optimizing factory process heat management while experimenting with renewable energy sources to effectively mitigate overall power consumption. The application developed for this thesis focuses on representing these assets accurately, facilitating the setup/internal optimization of these assets, and allowing the users to tailor the configuration to explore consequences and effects of their changes.

Employing a multidisciplinary methodology, the study seeks to integrate Digital Twin technology tenets, Object-Oriented Programming, Polymorphism, Data Modelling and Visualization, and Complex Power Calculations to formulate a robust and user-friendly software interface. Furthermore, meticulous GUI design and advanced graphing techniques ensure data visualization is both informative and intuitive.

This research culminates in a comprehensive demonstration of the software's capabilities, successfully showcasing its efficacy in meeting outlined objectives. By affording users the ability to dynamically shape and assess microgrid configurations, this study advances the field of sustainable energy management and underscores the potential for optimized factory operations within a renewable energy framework.

Acknowledgements

I extend my sincere appreciation to Michaela Kerr for her contributions to this dissertation. Her provision of comprehensive factory data spanning a period of approximately 10 months was pivotal in facilitating the rigorous testing of my application and the execution of a comprehensive case study.

Additionally, I would like to express my gratitude to Mark Apperley and the Ahuora Research Group at the University of Waikato for contributing various housing data. This data, which represented 27 houses and a year's worth of data per house, was donated. This data has significantly increased the depth of the case studies in this dissertation, as it facilitated fantastic real-world illustrations of how the application might be utilised to accomplish its intended purpose. Mark also dedicated hours to providing guidance and feedback into the featured application in this thesis, as well as this thesis itself, which has been extremely helpful in this endeavour.

I extend my acknowledgement to the creator(s) of the Oxyplot graphing library for C#, which benefit the expediency of final stages of this project greatly. The repository which hosts the source for this graphing library can be found in the appendices.

I extend my gratitude to my manager Paul Kennett, and my wider employers at the Waikato Regional Council, for employing me as a postgraduate student, and for their flexibility and facilitation of an environment which allowed me to balance study and work.

I extend sincere gratitude, and love for my mother Janine Clow, in her proof reading of this document, and the precision she employed when looking over grammar, continuity, and references. The flow and professionalism of this document is a result of her generous time and proof reading.

I thank the Creator, Yahweh, Father, Son, and Holy Spirit for blessing me with the opportunity and strength to undertake such a study. May any success that comes of it be credited first and foremost to the Glory of God, and may the magnificence of his creation continue to be uncovered forevermore.

Table of contents

List of figures and tables

1 INTRODUCTION

The provision of sustainable and renewable energy sources is a crucial issue facing industrial factories today. New Zealand in particular has a large dairy industry, consisting of over 66 factories that contribute more than 10 billion dollars to the country's economy. The Ahuora project*, based at the University of Waikato, is a pioneering New Zealand-based initiative that leverages adaptive digital twin technology and cutting-edge energy systems science to promote sustainability in New Zealand industries. Collaborating with leading researchers from Waikato, Auckland, and Massey Universities, as well as international partners, the project focuses on the dairy processing and timber drying sectors, aiming to achieve substantial energy and cost savings. The project's centrepiece is the development of an accessible Adaptive Digital Twin energy-technology platform that empowers users to optimize their energy systems, reduce costs, and meet emissions targets. Through smart energy systems and integration, the Ahuora mission is to assist in the development of more sustainable New Zealand industries that are in harmony with the environment and the people. Ahuora Aim 3 looks more specifically into this challenge by exploring innovative solutions to provide renewable energy for factory process-heat installations. The project proposes to achieve this objective through the integration of smart "factory edge" energy grids that connect the industrial site with the local community, renewable energy resources available in the area, and the wider electricity grid. This approach presents an opportunity to achieve significant energy savings while also contributing to the transition to a more sustainable future.

One significant problem with addressing this challenge is the absence of user-friendly modelling software designed for microgrids capable of accommodating the suggested components. Currently, there is a scarcity of tools for investigating and analysing microgrid designs dynamically. Such tools will be essential for aiding researchers and microgrid designers in efficiently exploring diverse microgrid scenarios and configurations, and therefore would be a powerful asset to the Ahuora project in their endeavour towards New Zealand’s industrial sustainability.

Therefore, the purpose of this thesis is to investigate and unravel the finer details of this problem through creating a software that models Ahuora Aim 3's proposed smart energy grid for industrial factories. The aim is to develop a comprehensive software application that integrates the industrial site with the local community, renewable energy sources, and to include considerations when utilizing the electricity grid at large. The application should provide valuable insights into the potential energy savings, economic benefits, and environmental impact of implementing such a solution. Essentially, in its fully completed form, the application will represent a sandbox-style microgrid simulator, which can facilitate factories and all other microgrid asset types, for comprehensive microgrid exploration and design analysis. By developing this software-based solution, it will be possible to assess the viability of Ahuora Aim 3’s proposed strategy quickly and cost-effectively while also learning new and intriguing details about the proposed microgrid through design exploration, all while avoiding the need for immediate real-world implementation.

With Chapter 1 introducing the problem, and proposed solution, the following thesis addresses and documents the creation of this application. Chapter 2 covers an in-depth look into related literature, and other studies, to gauge an appropriate direction for the application. Chapter 3 outlines a clear approach that will be followed when implementing the solution, and also describes specific design facets and elements of the proposed application. Chapter 4 closely explains the implementation of the proposed software, with explanations of critical architecture, and algorithms which contribute to

* https://ahuora.waikato.ac.nz/energy

microgrid modelling. Chapter 5 evaluates the effectiveness and robustness of the proposed application through the undertaking of case studies, with chapter 6 closing the thesis off with conclusions and suggestions to further work.

2 BACKGROUND

2.1 OTHER ATTEMPTS AND RELATED PROJECTS

The goal of this project is to create a software application that can simulate microgrids and allow for a thorough investigation of different microgrid design paradigms. This study, which is particularly unique in its scope, looks toward new territory, as there are not any other projects like it, according to a survey of the literature conducted on a lesser scale. Real-time microgrid controller solutions, such the PXise[*] controller, are commercially available and focus on the present real configuration rather than performance forecasts or theory building, which is the direction this study is headed.

Another interesting service available in the realm of microgrid software is the Microgrid-as-a-service (MaaS) platform provided by Planet Ark Power[**]. This project focuses on a solar microgrid service, which allows customers to have a microgrid installed without having to pay the upfront cost. The company that installs the microgrid owns it and is responsible for its maintenance. The customer can benefit from the microgrid by reducing their energy costs and having a backup power. While the implementation and workings of this project are proprietary to the company, the concept sheds light on the possibilities within this field of research. This service alludes to the potential for accurate organization and representation of grid assets, within a software context, which is a clear direction for this study that will be developed in further sections.

2.2 INITIAL FINDINGS FROM LITERATURE

Microgrid design and optimization has been the focus of several studies in recent years. A simple, yet effective way to begin the research into appropriate directions is to collate a referenced list of useful findings, against their respective sources. After reading and digesting the content of many academic sources, key themes and findings have highlighted themselves across the studies, each of which pertaining in some ways to the forthcoming thesis:

Table 1 - Research findings summarized

Finding	Source(s)
The integration of renewable energy sources, such as solar and wind power, in microgrids is essential.	The majority of referenced papers on microgrids and optimization solidifies this idea. Namely Amoura et al (2021), Han et al (2018), Badruhisham et al (2022)
Particle swarm optimization and multi-objective evolutionary algorithms are frontrunning algorithms for microgrid operation.	Phommixay (2019), Ferreira (2020), Aquino & Unsihuay-Vila (2021)
The optimization of energy management in microgrids is crucial for efficient and cost-effective operation.	Aquino & Unsihuay-Vila (2021)
Decentralized power generation and control strategies are the main trends in microgrid design.	Han et al (2018)

[*] https://pxise.com – Microgrid controlling and software for managing complexity.
[**] https://planetarkpower.com/microgrid-as-a-service/ - Product website for Microgrid as a service.

The use of battery energy storage systems in microgrids can improve energy efficiency and system reliability.	Bhoi et al (2023)
The optimal sizing of microgrid components, such as solar panels and batteries, is critical for achieving sustainable energy goals.	Bhoi et al (2023), Han et al (2018).
The importance of robust and reliable control and operation strategies for safe and efficient microgrid operation.	Altin & Eyimaya (2021)

In the realm of microgrid design and optimization, existing scholarly works have underlined the significance of amalgamating renewable energy sources, fine-tuning energy management approaches, and forging robust control and operational methodologies as pivotal strides toward achieving sustainable energy generation and consumption within microgrid contexts.

An all-inclusive tool should enable the simulation of a variety of scenarios, highlighting and explaining the optimal combination of energy sources, energy reservoir systems, and regulatory strategies that result in the maximisation of the microgrid's operational efficiency, dependability, and adaptability. Furthermore, the tool should allow for continuous monitoring of energy trajectory and storage, as well as the ability to forecast and respond to fluctuations in energy demand and supply. Essentially the tool should be intricately customizable, across all of its facets. Such an instrument can fast-track microgrid researchers toward a sustainable energy landscape by endowing microgrid administrators and vested stakeholders with the ability to closely analyse judgments that are based on the tool's guidance and judgement.

2.3 WORK WITH MODERATE RELEVANCE

2.3.1 Microgrid Optimization through various algorithms

Some interesting research, which pertains moderate relevance to this project is the work in Microgrid Optimization algorithms, to improve overall efficiency and performance of microgrids. As alluded to in Table 1, the algorithms that will be looked into are Particle Swarm Optimization (PSO), and Multi-Objective evolutionary algorithms (MOEA).

2.3.1.1 Microgrid Optimization

Both particle swarm optimization, introduced by Phommixay (2019) and multi-objective evolutionary algorithm introduced by Ferreira (2020), have been used for optimizing microgrid operation. The further advancement of this field holds the potential to yield heightened command and oversight over the microgrid, obtaining operational efficiency and dependability. The most critical factor for microgrid optimization lies deep within the specific details of diverse renewable energy sources, and their integration into the microgrid matrix, as well as the same level of thorough thought and optimization of energy storage technologies (Phommixay 2019). These types of complexities at an individual scale are crucial for accurate and considerate modelling (Mo et al 2018). This collaborative effort maximises the use of renewable energy sources while guiding us away from the overuse of fossil fuels. Additionally, the future for research entails developing novel frameworks to enable the best design and operation of microgrids even in constraints like changing weather patterns and load requirements. The result of such investigations will enhance the resilience and adaptability of said

systems, which is critical to the microgrid system itself. In parallel, a critical area requires a complex consideration of the societal and economic impact of microgrid installations in local communities.

2.3.1.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is described by Phommixay (2019) as a heuristic optimization technique that operates on a swarm (population) of N particles. These particles are initially distributed randomly within a search space. Each particle in the swarm is characterized by its position and its velocity within the search space D. Here, the Particle number index ranges from 1 to N, and space index ranges from 1 to D. PSO involves the iterative movement of particles, where each particle updates its position based on both its best-known position and the best-known position of its neighbouring particles. A result of the final state of the simulation is many particles swarming to a single position, revealing an optimal configuration. In relation to microgrid technology, this can be utilized, but also have its limitations, which are listed below.

Advantages of PSO:

1. Faster convergence to a solution compared to traditional optimization methods.
2. Requires less computational resources.
3. Handles non-linear optimization problems well.
4. Has been successfully applied to microgrid system optimization.

Disadvantages of PSO:

1. May converge to a local optimum rather than the global one.
2. May not work well for large-scale optimization problems.
3. Can get stuck in a search space and fail to find an optimal solution.
4. Parameter tuning is required for best results.

These conclusions are derived from the work of Phommixay (2019).

2.3.1.3 Multi-Objective evolutionary algorithms

In MOEA, a population of candidate solutions, often referred to as "individuals" or "chromosomes," evolves over multiple generations through processes inspired by natural evolution, such as selection, crossover, and mutation. These algorithms aim to explore the trade-off surface between multiple conflicting objectives and find a set of Pareto-optimal solutions. A solution is considered Pareto-optimal if no other solution in the search space can improve one objective without degrading at least one other objective (Ferreira 2020).

MOEAs have applications in various fields, including engineering design, finance, scheduling, and other domains where decision-making involves optimizing multiple conflicting criteria, but as discussed by Ferreira et al (2020), its heuristics can be applied in the microgrid setting for optimization. Various MOEA algorithms help decision-makers explore and understand the trade-offs between different objectives and make informed decisions based on the Pareto-optimal solutions they provide. MOEA algorithms have also been used to solve the demand-side management problem in microgrids, lowering operating costs and reducing emissions (Aquino & Unsihuay-Vila 2021). In the microgrid context, due to the extreme number of moving factors, there will almost only be Pareto-optimal solutions, as opposed to a single “best state”.

Advantages of MOEA:

1. Can handle multiple objectives simultaneously.
2. Can find a set of Pareto optimal solutions.
3. Robustness to uncertainties.
4. Generally, more effective for complex optimization problems.

Disadvantages of MOEA:

1. Requires more computational resources than PSO.
2. Can be slower to converge to a solution than PSO.
3. May suffer from premature convergence.
4. Can be difficult to tune parameters.
5. Does not provide singular final solutions.

These conclusions are derived from the work of Ferreira (2020).

2.3.2 Optimization's place in this research

Both PSO and MOEA have their advantages and disadvantages when it comes to optimizing microgrid operation. Which algorithm to invest research and resources in largely depends on the specific problem being solved and the resources available for computation. A pivotal consideration in this research project is the balance between optimization and exploration. While the core essence of this endeavour revolves around exploration rather than strict optimization, the ability to delve into comprehensive optimization would yield profound insights into microgrid configurations. This approach would enable a direct examination of cause-and-effect dynamics, shedding light on intricate relationships.

While optimization undoubtedly reveals ideal configurations and scenarios within the defined parameters, the explicit visualization of optimization's underlying principles, achievable through exploration and creative freedom, enriches the value of the research. Consequently, the incorporation of intricate optimization equations and algorithms into the final solution is not the primary objective. Instead, it serves as a valuable resource for users and stakeholders, irrespective of their expertise levels, ultimately enhancing their engagement and comprehension of the microgrid dynamics.

2.4 WORK WITH CLOSE RELEVANCE

2.4.1 Digital Twin Technology

2.4.1.1 *Overview*

The research in the realm of the Digital Twin technologies within the software context is prominent theme in many areas of research. The work of namely Minerva et al (2020), Bazmohammadi et al (2022) delve into an in-depth exploration of the underlying technologies that drive this idea. These researchers venture into the practical realm, investigating how the Digital Twin concept is actively harnessed across real-world scenarios.

Minerva et al (2020) uncover the inherent characteristics that truly define this new age idea. Furthermore, they navigate the intricate terrain of how the Digital Twin concept seamlessly integrates into software architectures, particularly within the realm of IoT architectures. A crucial aspect covered is examination of the feasibility of an ecosystem that can effectively leverage the Digital Twin concept. Another researcher describes Digital Twins as application abstractions of complicated physical systems, which are linked to the real system via a communication link to continuously exchange data

with the real environment and create a dynamic digital mirror with a modelling engine that is always running (Bazmohammadi et al 2022).

2.4.1.2 Digital Twins broken down

The Digital Twin (DT) concept is gaining attention in various industries due to its ability to clone a physical object (PO) into a software counterpart. The software twin object, termed logical object, reflects all the important properties and characteristics of the original object within a specific application context. The power that comes from digital twins, and their usefulness is emerging in other fields, such as augmented and virtual reality (e.g., avatars), multiagent systems, and virtualization (Minerva et al 2020). To properly refine a shared DT definition, a set of fundamental properties is identified and proposed as a common ground outlining the essential characteristics (must-haves) of a DT. Once the DT definition has been refined, its technical and business value is discussed in terms of applicability and opportunities. Digital twins are similar to plain simulation, but the key difference between DT and other simulation or representation methods is that DT is designed to be dynamic and intelligent (Bazmohammadi et al 2022).

This idea can be taken into a deeper digital twin concept in relation to the wider organization. A Digital Twin Organization (DTO) includes processes, services, people, roles, and all other relevant elements for the operation of organisations in addition to the usual representation of devices, machines, and physical assets provided by Digital Twins. As a result, DTO can play an important role in realising and analysing aspects of organisations, assisting managers in their understanding of the organization's status, and anticipating the effects of potential changes in the organisation (Edrisi et al 2021).

2.4.1.3 Applications to this research

Correctly utilizing and incorporating digital twin technology would unlock an advanced sense of realism when it comes to the modelling of a microgrid. For example, Guzman Razo et al (2020) investigates the underlying complexities of photovoltaic (PV) systems in great detail. Their work makes a substantial contribution to the enhanced optimisation of PV power systems by building on this thorough understanding, with their strategy encompassing the fields of self-learning and digital twinning, adopting cutting-edge ways to increase the capabilities of PV systems.

In a similar vein, Wang et al (2023) discuss the significance of modelling the status of physical objects, particularly wind turbines, for virtual-real interaction. It emphasises the use of customised software for dynamic simulation, as well as the role of various data sources in developing a comprehensive digital twin that ensures real-time synchronisation and improved operation. These kinds of insights gained from digital twinning PV and wind turbine systems, applied to an explorative tool would be monumental for design analysis of a microgrid.

The overarching objective of this research is to design and create an application that delves further than simply simulation, and numbers. True achievement in this research area relies on closely mirroring the behaviour and performance of physical hardware components. It is important to clearly state that the application operates independently from integration with real-world products. Instead, a distinctive approach is adopted, harnessing the capabilities of digital-twin-like technology that models a real-world component without the real-world connection to it. This methodology empowers the application to closely emulate, interact, and enact the anticipated behaviour of its real-world hardware counterparts, with the requirement for having said components on hand being void.

The work proposed by Edrisi et al (2021), which considers the wider incorporation of assets and functionality into a digital twin, is more in line with the goals of this project. If the solution is to be realistic, the simulation of an entire microgrid must consider processes, key jobs, data sources,

effectively as many moving parts as possible. The intended result of this thesis therefore goes beyond the limitations of conventional simulation. It attempts to capture the subtle nuances and reactions unique to physical equipment in order to create a strong sense of realism and connection to the physical objects. While genuine hardware integration remains detached, the application requires the ability to replicate the anticipated functionalities of the physical objects, which enhances its utility as an experiential and analytical platform. Even though it will be challenging, one of the obvious main objectives of the subsequent research is to establish a symbiotic interaction between technology and reality.

2.4.2 Software Scenario modelling

A critical aspect of this project is meticulous and comprehensive design, which can be accomplished by combining object-oriented programming with scenario-based software modelling. It should be noted that using these methodologies is not a novel concept, but rather a fundamental practice in good software development (Bai et al 2002). To fully leverage this methodology, it is critical to the objectives of this project to conduct a thorough and well-structured mapping and planning of each component of the application to be developed. This diligent planning process will provide a deeper understanding of the application's nature, allowing for simple integration of additional features and enabling future research efforts.

2.4.3 Environmental Considerations

The significance of simulating environmental factors cannot be understated in the context of microgrid simulation and design exploration. This is primarily attributed to the profound influence that environmental variables, such as sunlight and windspeed, exert on the performance of crucial microgrid components, including solar panels and wind turbines. Extensive research in this domain has already been conducted and can be instrumental in informing the development of this application.

Numerous academic papers have prominently employed the Weibull distribution as a predictive model for windspeed. This choice finds its substantiation through a multitude of empirical studies. For instance, Jha and Kumar (2023) have employed the Weibull distribution in practical case studies, while Michalík et al (2022) and Verma et al (2022) have undertaken comprehensive investigations into general and locational modelling, respectively. In their collective endeavours, these researchers have meticulously defined the Weibull distribution, delved into its underlying statistical principles, and elucidated its versatile applications. Furthermore, they have rigorously validated the effectiveness of the Weibull distribution in accurately estimating windspeed through empirical experiments and comparisons against real-world data. The utilization of these principles within this project will be extremely important in order to provide users with realistic scenarios in which their microgrid may be in.

In the development of this application, it is essential to consider environmental data, including sunlight and other relevant factors. Incorporating real data into the simulation of photovoltaic (PV) systems is of paramount importance for multiple reasons. Firstly, real data offers precise and dependable insights into the performance of PV systems. This data can be used for system planning, comparison, and evaluation (Jahn & Nasse 2004). Using real-world data allows researchers and engineers to assess the operational efficiency of grid-connected PV systems, allowing them to make more informed decisions about system design and enhancement (Jahn & Nasse 2004). Furthermore, the use of real data allows for the validation and verification of simulation models, or the proposed pseudo digital twins which are mentioned in 2.4.1.3. The accuracy and reliability of the developed PV simulation models can be determined by comparing simulated results to real-world data (Cieslak & Dragan 2018). By incorporating these principles, the simulation and integration of environmental data

will contribute significantly to the creation of a genuine and robust environment for microgrid design exploration and simulation in this project.

3 APPROACH AND DESIGN

3.1 APPROACH BASED ON BACKGROUND

The microgrid in entirety, as a concept, contains an extensive number of avenues, explorations, and applications to consider as attributes of this study. Given this, it is important to hold the understanding that this concept, in its completeness, contains dimensions that go well beyond the scope of a single scholarly endeavour. Prudent and smart choices are required for the task, to identify particular aspects, which will benefit the study through emphasis. Given the time-restrained nature of this study, it was imperative to precisely define, and prioritize areas that hold the viability for comprehensive assessment within this timeframe. The deliberate process of selecting areas demanded a careful assessment of the most impactful and feasible topics, aligning with the resources, expertise, and time constraints at hand.

The following chapter is rationale for a clear and attainable trajectory for meaningful research within the domain of microgrids. Due to the microgrid's encompassment of a multitude of components, constructing a virtual interplay of this type of system poses as a formidable task. Constructing precise and impactful modelling tools will be one critical hurdle to overcome. On top of this, there is a clear absence of universally accepted standards in terms of metrics and models used to assess microgrid performance. Dissimilar metrics and models can yield markedly dissimilar outcomes, therefore toughening researchers' ability to gauge the comparative effectiveness of different tools. Furthermore, the data pertaining to specific facets of microgrids, such as the behavioural patterns of renewable energy sources is difficult to attain comprehensively, which further introduces complexity in the endeavour to accurately model microgrid behaviour.

Despite these challenges, the ramifications of proficient microgrid modelling tools will be profound. These tools can furnish energy planners and policymakers with well-informed insights, facilitating judicious decisions pertaining to the establishment and operation of microgrid systems. Moreover, they offer microgrid operators the means to optimize energy generation and consumption, ultimately steering toward a trajectory of enhanced energy efficiency and sustainability.

3.1.1 Decisions on Fundamental architecture

3.1.1.1 User freedom

The solution to this exploratory endeavour is an application that allows users to explore a microgrid, make modifications, and see the effects of those changes unfold before them. User flexibility is undoubtedly the most crucial component of this approach in order for the solution to facilitate exploration and design analysis. If user freedom is implemented effectively, the weight of understanding will fall largely on the Users' grasp of the subject matter at hand, therefore keeping the door wide open for exploration. This path entails careful GUI design plus proper definition of related objects to encompass, and careful crafting of the modelling of said objects which the user may be interested in capturing.

3.1.1.2 Reasoning against internally focused optimization

An in-depth analysis of a number of important factors led to the conscious choice to forgo pursuing microgrid optimizations in this thesis. Point 2.3.2 touches on this but does not allude to a direct decision. First, from the research acknowledged prior, it is clear that the research in the domain of microgrid optimizations is extensive, down to specific algorithms and their implications, resulting in a wealth of readily applicable, existing methodologies and solutions that can applied beyond the

functionality of the application. Within the constraints of this study, diverting resources into the redevelopment of optimisation methods could unintentionally draw attention away from the larger exploratory goals. The purpose is exploration, rather than optimization, and allocating time and effort to optimization algorithms may detract from fully immersing in the overarching goal of insightful exploration.

3.1.1.3 Power Generation

Based on the aim of the project, as well as past reading, the inclusions and exploration into power generation avenues is a fundamental. Incorporating this well will allow for:

I. Comprehensive Insight: The inclusion of aspects related to power generation gives a comprehensive picture of the microgrid's self-sufficiency and capacity to generate energy from renewable sources.
II. Environmental Impact: The development of these objects can provide insights into the environmental effects of various energy sources by assessing and simulating power generation, enabling well-informed decisions towards sustainability.
III. Microgrid Viability: Investigating different power generation techniques makes it easier to choose the best solutions for the energy mix of the microgrid, improving dependability and efficiency.

3.1.1.4 Power Consumption

In a similar light to generation, it is absolutely essential to include explorative consumption avenues in this project. Proper implementation will facilitate:

I. Load Dynamics: Power consumption analysis reveals peak usage times and changing energy demands, which are essential for developing reliable microgrid systems that can effectively handle a range of loads.
II. Efficiency Optimization: The ability to identify energy-saving options and load control techniques is facilitated by an understanding of consumption patterns, which consequentially improves overall energy efficiency.
III. Resource Planning: Consumption knowledge enables power generating and storage capacity to be tailored to the microgrid's unique energy needs, avoiding over or under-construction.

3.1.1.5 Factory and House Modelling

As an addendum to the previous point, the power consumption avenues will be further outlined in this section. Within the Ahuora group at the University of Waikato, dairy factories take fundamental prominence as a key study subject. This project seamlessly fits into Aim 3, a crucial trajectory devoted to understanding the complexities of factory load profiles, community load dynamics, utilising the potential of renewable energy, assessing regional grid capacity, closely examining emission factors, and outlining future development pathways. Given that both factory loads, and community loads constitute integral segments of power consumption, their prudent inclusion within this project's implementation emerges as necessity. By accurately representing these facets, the project not only adheres to the mission of Aim 3 but also fortifies its stance as a comprehensive, real-world-driven initiative in harmony with the Ahuora group's overarching research goals.

In the general sense, including factory modelling will provide some general benefits as well:

I. Real-World Context: The inclusion of factory modelling portrays a realistic microgrid scenario in which energy dynamics are strongly correlated with industrial operations, reflecting the complexity of real-world situations.

II. Impact Assessment: By assessing energy demand spikes caused by industrial operations, factory modelling enables the foresight of potential grid strain and the development of optimisation techniques.
III. Holistic Analysis: The addition of factory modelling broadens the project's scope beyond traditional energy measurements and exemplifies a thorough strategy that takes multiple factors into account.

3.1.1.6 Power Storage

Lastly, power storage exploration emerges as the pivotal variable completing this equation. Its absence would preclude any investigation into energy balance and addressing issues associated with surplus and deficient renewable energy scenarios. Therefore, the application should include the ability to explore various energy storage types. Some other benefits of including power storage are as follows:

I. Energy Resilience (as mentioned): Power storage solutions enhance the microgrid's resilience by enabling energy surplus storage for future use during low generation periods, bolstering its self-sufficiency.
II. Peak Shaving: Power storage systems facilitate peak shaving, mitigating grid demand spikes during high load periods, which translates to cost savings and grid stability.
III. Technological Advancements: Studying power storage introduces innovative technologies like batteries and energy storage systems, positioning the project toward the forefront of energy technology exploration.

3.1.2 Methodologies

3.1.2.1 Object Oriented Programming

The adoption of Object-Oriented Programming (OOP) in the development of this thesis stems from its intrinsic ability to encapsulate complexity while facilitating modularity and reusability. OOP offers a structured paradigm that mirrors the inherent complexities of real-world systems, enabling the translation of intricate relationships, components, and behaviours into organized and intuitive code structures. By encapsulating data and functions into classes and objects, OOP facilitates the creation of a dynamic and extensible software framework. This approach streamlines the development process, allowing the construction of individual components with well-defined interfaces, enhancing code maintainability, and promoting efficient collaboration within the project. Furthermore, OOP's flexibility fits in perfectly with the complex structure of microgrids, where a variety of components, including generators, storage facilities, and consumers, call for flexible and interactive modelling. OOP can be used to build flexible, scalable microgrid modelling software that not only accurately models the intricate nature of these systems but also provides the path for future improvements and adjustments.

3.1.2.2 Inheritance and Polymorphism

This microgrid modelling application's integration of inheritance and polymorphism highlights a deliberate approach to boosting both efficiency and versatility. Inheritance, a fundamental concept of Object-Oriented Programming (OOP), empowers the application by enabling the creation of specialized classes that inherit attributes and behaviours from more generalized base classes. This hierarchical structure facilitates the organization of various microgrid components, such as different types of generators or energy storage units, under broader categories. Consequently, it streamlines the codebase and facilitates consistency while accommodating variations in functionalities.

Contrarily, polymorphism makes use of abstraction's power to enable various objects to communicate over a single interface. Regardless of their particular complexity, this idea enables the modelling of various microgrid system components using uniform methodologies and attributes. The programme can handle interactions between several objects without the need for complex conditional checks by implementing polymorphism. This gives the software design an elegant and harmonious touch, enabling adaptability as the microgrid system develops.

Inheritance and polymorphism work together to provide a modular, expandable, and user-friendly microgrid modelling application. It is obvious that as the project progresses, a variety of variables, object types, and class definitions will emerge. Making effective use of inheritance and polymorphism will enable the management of these many objects in a clean, wise, and logical manner. The combination allows for the efficient representation of intricate relationships between microgrid elements while preserving code coherence and simplifying future updates or additions.

3.1.2.3 Caching & Data storing

The strategic incorporation of caching in this project will be elaborated upon in subsequent sections, outlining its specifics. In general, using caching will produce a variety of advantages. The processing time will be significantly reduced, reducing the need for redundant computations. Caching efficiently reduces code duplication, boosting overall efficiency and code maintenance by centralising and storing data globally across many forms inside the programme.

Given that the available and computed data is likely to be required across various junctures within the application, caching inherently caters to this multi-faceted demand for accessibility. This approach streamlines the accessibility of pertinent information and contributes to smoother cross-form data interaction. A tangible advantage emerges in terms of expediting data retrieval from the cache, furnishing an additional layer of expediency to the user experience.

3.1.2.4 Decision against SQL Database/Persistent DB

After carefully analysing the pros and cons of including a database, such as MSSQL, within the programme, it was determined that this approach is not worth the extra work associated. The choice to forego using a database was primarily motivated by time constraints, and previous experience. While SQL databases are useful, and are prominent amongst industry standards, the skill needed to efficiently design, manage, and maintain tables, views, and the intricate data structures built into databases was a crucial factor. It became clear that handling such a database, particularly on a smaller scale within the programme, would bring needless difficulties given that this skill frequently calls for specialist staff. Once more, it would focus attention on a subject that is not necessarily the purpose of this endeavour. If the project is continued after research is stopped, it could be implemented as a software enhancement.

3.1.2.5 Information Visualization

The utilisation of timeseries graphing, statistic totals, and energy balance charts stands as an important decision within this project as information visualisation is essential to design exploration. This method fosters a deeper knowledge of microgrid behaviour by acting as a dynamic conduit for turning complex datasets into understandable insights. Timeseries graphing helps users identify patterns and abnormalities and provides a visual narrative of temporal trends, enabling them to see dynamic alterations in the microgrid system.

This is enhanced by the integration of statistic totals, which streamlines understanding of critical measurements and provides a clear overview of key parameters. Users are equipped to quickly evaluate the microgrid's overall performance thanks to this streamlined display without having to

delve into the raw data. Additionally, energy balance graphs show where the production, consumption, and storage of energy are balanced. This visual representation not only proves the stability of the system but also identifies potential areas for improvement.

The combination of timeseries graphing, statistic summaries, and energy balance plots is essentially more than a visualisation tool; it is a strategic methodology that spans the range between complex data and intuitive understanding. This method enables successful administration and optimisation of microgrid operations by providing users with the tools necessary to make informed decisions by visually representing complex information.

3.2 Application Design

The design of this solution is thoroughly described in the next part, which is based on the intended methodology and the approach that is pre-defined.

3.2.1 High-Level Overview

The application will take form as a user experience focused desktop application. This application is built to empower users by presenting them with a blank canvas: a microgrid to design and customize based on individual preferences. The design of this microgrid lies completely in the hands of the user, emphasizing user-centric control over its composition. Users will be able to start the construction of unique objects called “Assets” using a variety of interactive forms and dialogues. These Assets will effortlessly integrate with the microgrid architecture and take part in data simulations and models as the generators, consumers, and storage entities. An elemental feature is the customizable nature of these Assets. Users retain full autonomy in configuring every aspect of generation, consumption, and storage components, affording fine-grained control over their behaviours and attributes.

Upon setting up the microgrid to their desired specifications, users will transition to the simulation phase. The simulation periods available in this programme range from brief day-long scenarios to detailed year-long estimates. Users can benefit from in-depth study within these simulated environments. Performance analyses and the wide-ranging effects of the microgrid's configuration are precisely illuminated through these tools. The simulation realm serves as an observational learning platform where users witness firsthand the intricacies and consequences of their microgrid configurations, facilitating insightful decision-making and iterative refinement. This combination of user-driven design, adaptable simulations, and thorough analysis reinforces the application's position as a crucial tool for exploring, personalising, and developing microgrids.

Users will still have the ability to alter simulation-contributing events at their leisure, such as the weather and the season, and see the effects of those changes. Directly comparing asset loading will be possible using a live tool that will dynamically update when configurations are altered in the background. To prevent large, sophisticated computations from taking place at inconvenient times, data modelling, simulating, and forecasting will be done on-demand.

The following process diagram in Figure 1 is used to depict what a basic expected user experience when inside the proposed application.

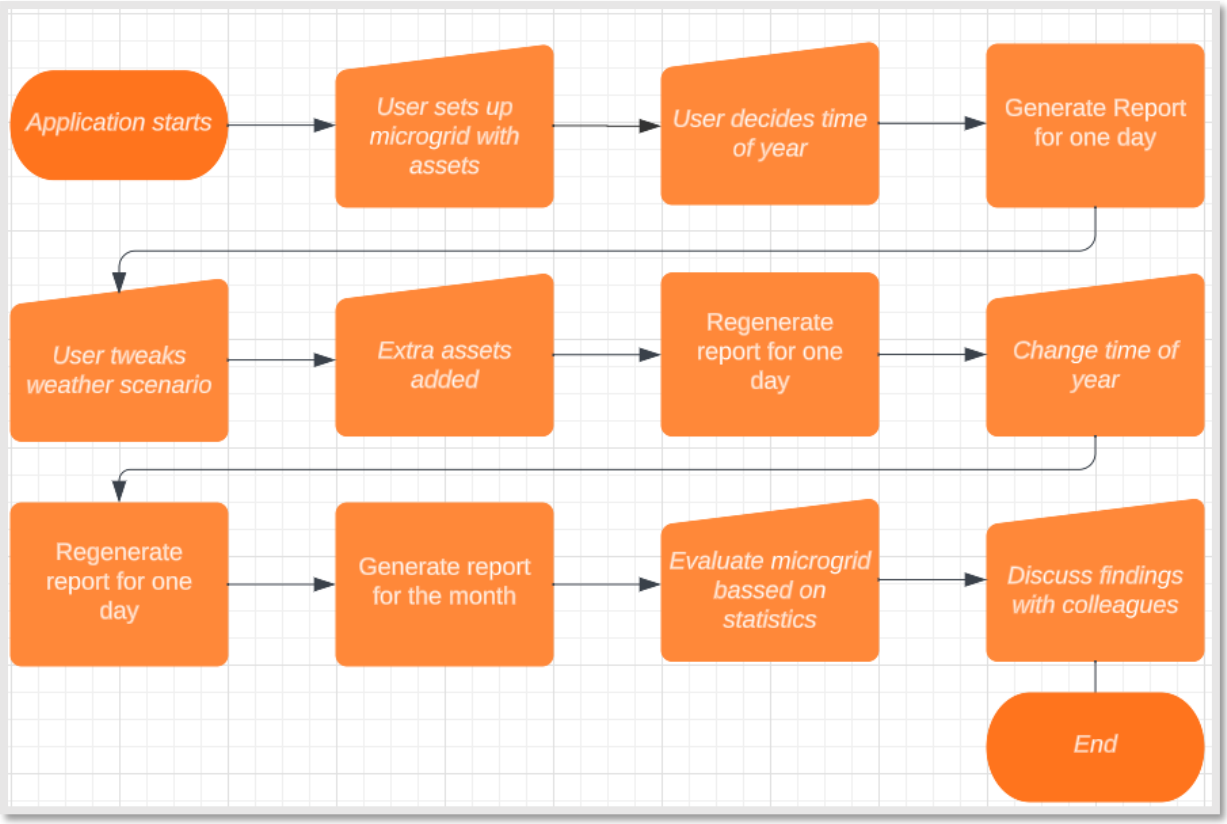


Figure 1 - High level overview of system usage

3.2.2 UML Design

Figure 2 visually represents the recommended class structure, encapsulating essential components and potentially beneficial methods integral to the design of the proposed application. At its core, this structure defines fundamental objects that constitute the bedrock of microgrid functionality. This classification system distinguishes between objects that feed energy into the microgrid, objects that extract energy from the grid, and objects that store excess energy. Positioned between these foundational objects is a static entity termed the "Cache." Functioning as an intermediary, this *Cache* assumes the role of the microgrid, serving as the realm upon which these fundamental objects interact. Conceptually, this *Cache* comprises organized lists, acting as repositories for specialized classes that correspond to each of the suggested fundamental objects.

The UML representation, as laid out in Figure 2, illustrates the structural blueprint that underpins the application's core architecture. Building upon each of these foundational fundamental objects, the class structure includes detailed implementations of physical objects, which can be viewed as child classes. These child classes inherit attributes and behaviours from their respective abstract fundamental objects, adjusting their capabilities to the world of actual, tangible objects inside the microgrid. An outline of this concept is aptly captured in section 2.4.1, wherein digital twins and their integration into the project's framework are discussed. These classes, which are comparable to digital twins, represent the merger of computer modelling with real-world phenomena.

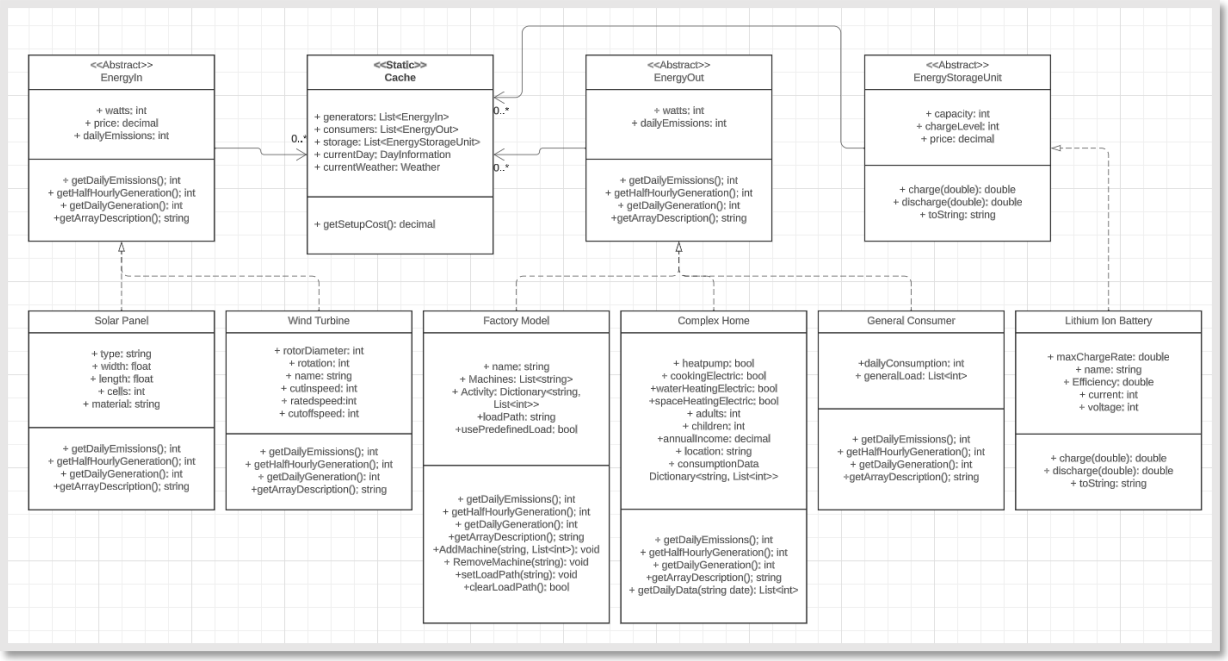


Figure 2 - Initial UML Design

Within Figure 2, a collection of rudimentary methods and variables are thoughtfully proposed to underpin the operational scope of these classes. These recommendations work as a rough compass, pointing the way for features ranging from complex interactions with the microgrid's cache to energy generation, consumption, and storage. As these child classes arise from their abstract counterparts, they actualise the essence of digital twins and embody the merging of conceptual representation and practical reality.

Consumers are further defined into two entities, as outlined in point 3.1.1.5. The classes are simply *Factories* and *Houses*. Data of each kind is readily available thanks to collaborative efforts, as well as the acknowledged parties, and this availability pushes the direction of the suggested *Energy Out* objects toward these two specific distinctions. It must be highlighted that a proper implementation will also include the construction and customization of personalised copies of these objects, to maintain proper user freedom, even while the data that will be read into these objects will serve as good template objects and assets for the users.

3.2.3 Functionality and GUI Design

3.2.3.1 Home screen

Figure 3 shows a hypothesised home screen for the application that is intended to give users an overview of the microgrid's individual objects. This home screen serves as a central hub for interacting with the microgrid's components. User access to various application functionalities is facilitated

through an intuitive menu strip. This interface choice ensures user familiarity and ease of navigation, creating a comfortable and secure platform that resonates with users of all expertise levels.

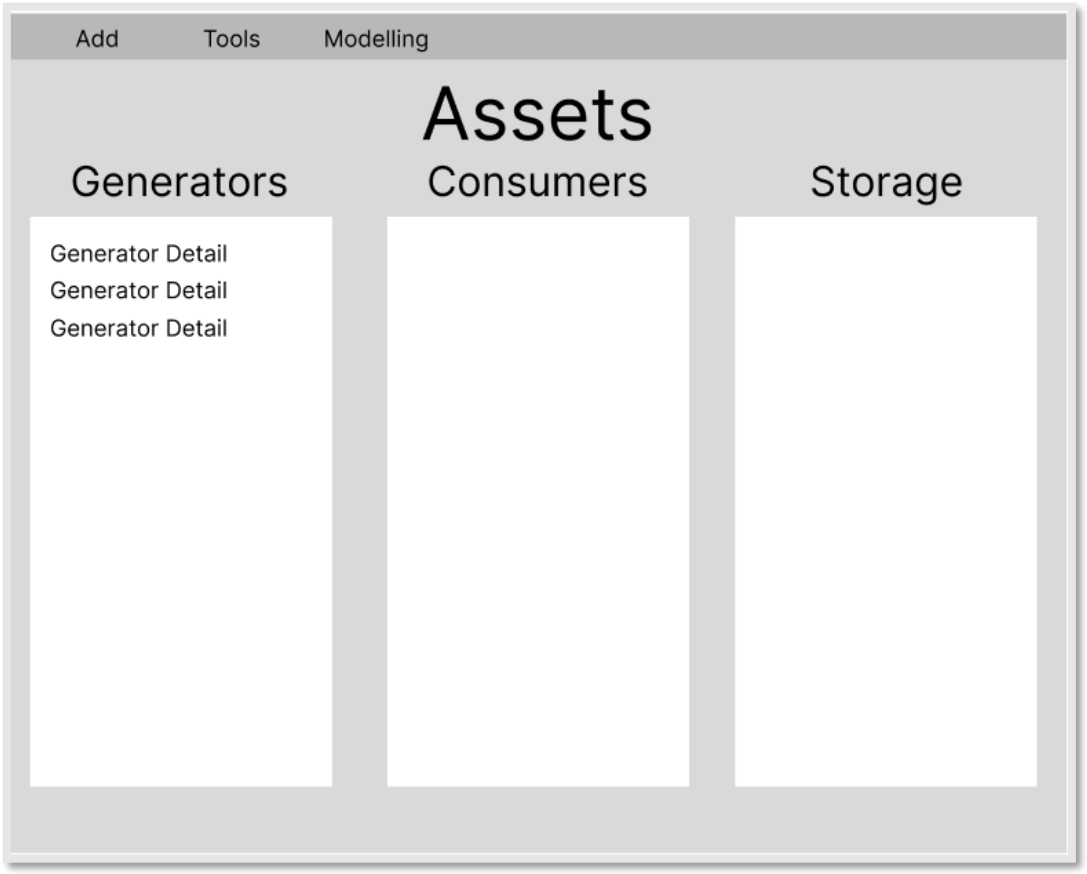


Figure 3 - Main Form Wireframe

3.2.3.2 Adding Assets and Energy Storage

Incorporating generation, consumption and power storage assets needs to be a straightforward process for users. The proposed approach employs a form-based UI design, fostering a simple method of introducing customized items to the grid. Within this interface, the generators' segment contains *Solar* and *Wind* entities, whereas batteries constitute a distinct category. The showcased fields are suggested and not conclusively finalized. The eventual implementation and class definitions hinge on discoveries made during the development process, as well as the availability of pertinent data. This lays the groundwork for object definitions in situations where pre-defined objects in which users could select with ease could be sourced from useful data. The final design (Figure 4), which incorporates improvements learned from the application's development, and data obtained, will be revealed in a subsequent section.

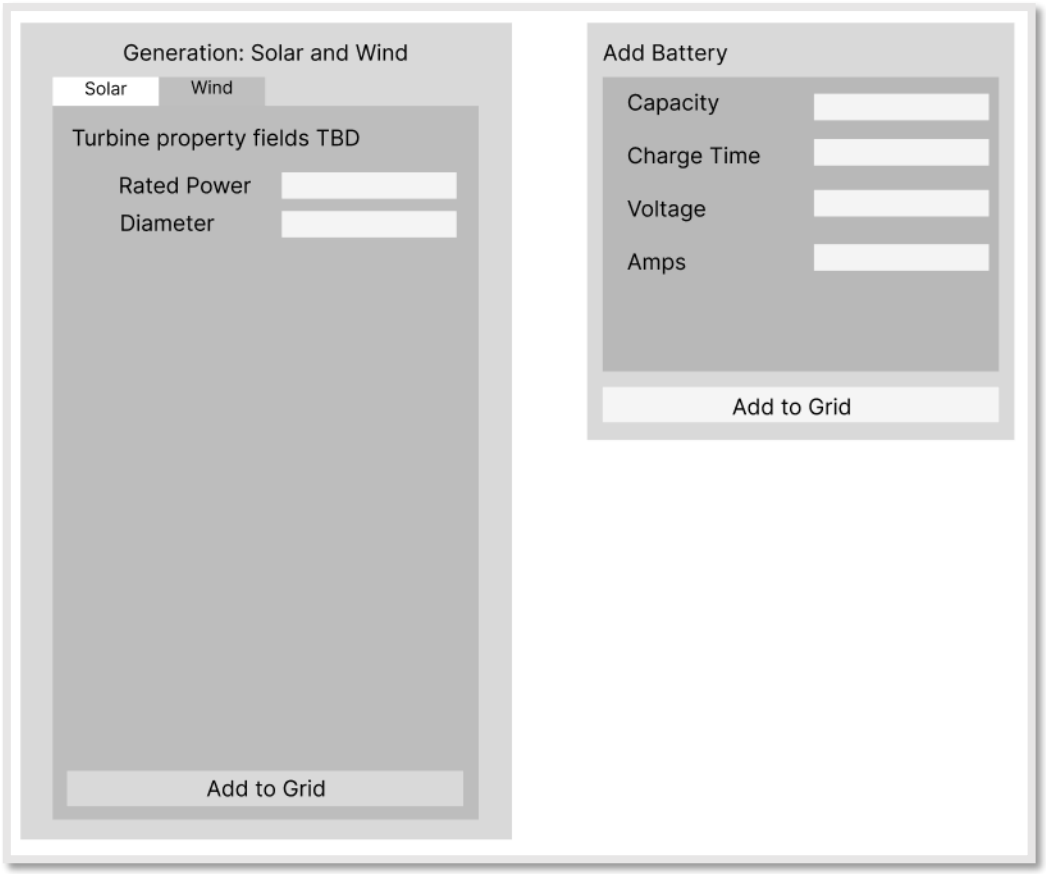


Figure 4 - Generation and Battery adding forms wireframe

3.2.3.3 Factories and House Models

Given the need to clearly distinguish between factory loads and house loads, the UI design acquires critical importance in supporting their seamless integration. A factory setting can be established, providing consumers with the ability to add power-consuming machinery or goods. This covers the provision to carefully define operational timeframes for the designated machinery. This design naturally leads to the creation of a pseudo-load, an artificial construct that simulates the factory's predicted power consumption. Within this interface, the potential for load shifting appears as a result of the design choice, constituting microgrid optimization and exploration strategies. Figure 5 is a depiction of this interface. It should be noted that users should be able to import their own data if they possess it, which is why the interface is divided into two sections for manual load configuration and importing load data.

House models occupy a similar light within the application, albeit with nuanced differences in approach. The nuances come from the application's capacity to provide users with template houses of diverse types, each equipped with reasonably comprehensive load profiles, spanning the duration of a year. Because of this ability to choose from pre-defined house models, users have this choice, or the choice to upload their own data. Consequently, configuring these houses on the user's end deviates slightly from the process applied to factories. Users are given the flexibility to define typical house characteristics, as documented in the pre-loaded data, which contribute to the houses load. These attributes will encompass electrical heating, electrical cooking, and electrical water heating, among others. In line with their preferences, users can opt for the application to select a load profile that closely mirrors their defined household characteristics. The application will be designed to intelligently identify the most suitable load profile, drawing from the proximity of the user-defined

house to pre-loaded templates within the application. Alternatively, users retain the flexibility to import their own data, aligning the application's functionality with their specific needs and enabling a tailored approach to microgrid exploration and analysis. Figure 6 displays the interface designed to support this customization.

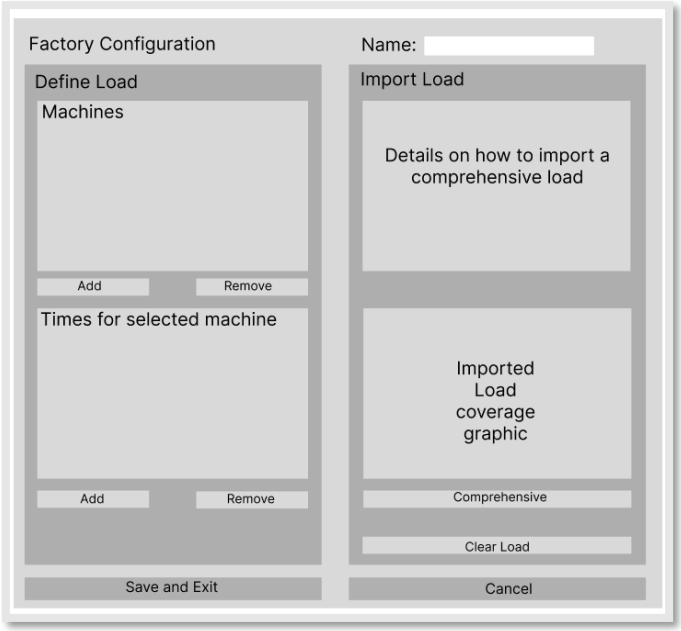


Figure 5 - Factory Configuration wireframe

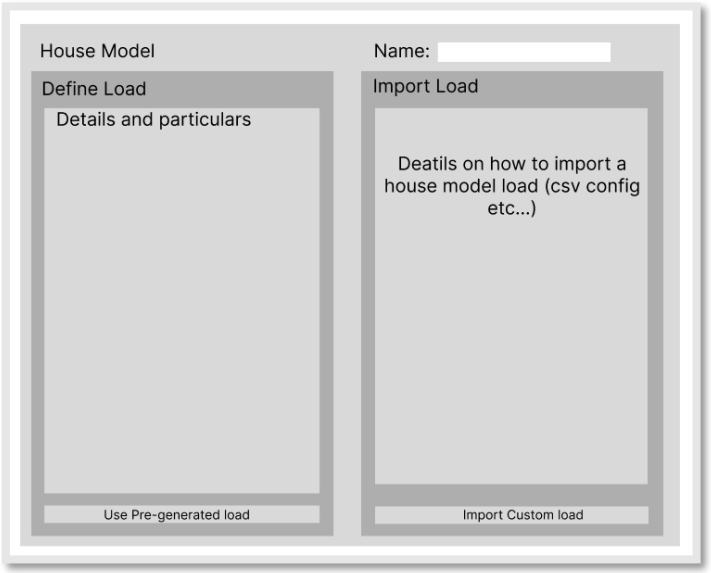


Figure 6 - House Model form wireframe

3.2.3.4 Tool: Load Comparison

Figure 7 portrays a utility designed to facilitate swift load comparisons between two objects. By default, the tool operates within a predetermined time period, typically set at 1 day. Users have the flexibility to specify the generator and consumer they wish to compare or analyse. Once these selections are made, the tool instantaneously generates a snapshot of estimated performance and pertinent statistics, positioned next to the chosen objects. It is important to note that this tool is not

designed for exhaustive modelling endeavours; rather, its objective is to speed up simple load comparisons. This will be particularly useful to gauge energy balance on short notice.

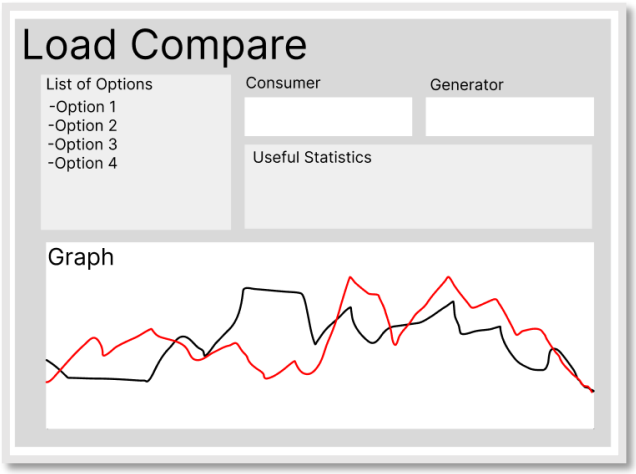


Figure 7 - Load Comparison form Wireframe

3.2.3.5 Reporting: Daily, Monthly, Yearly

The creation of thorough reports is a crucial function, addressing the demands of exploration, results analysis, and the capacity to assess the effects of alterations made to the grid. These reports undertake the task of meticulously modelling the performance of every asset within the grid, capturing their intricate interactions over defined time intervals, including daily, monthly, and annual breakdowns. In essence, the report serves as a comprehensive repository of all pertinent grid-related information. It has many key facets, including cost analysis, power balancing, and when applicable, estimates of emissions. It is noted that the computational complexity of producing such reports, depending on their size and the number of grid items, may need a long processing time. A simple loading animation that reassures customers that the application is actively processing data rather than becoming unresponsive will be built. Figure 8 shows the general layout that each of these reports will follow. Each will vary in the implementation phase, to account for the differences in time spans, and palatability of data over different time periods.

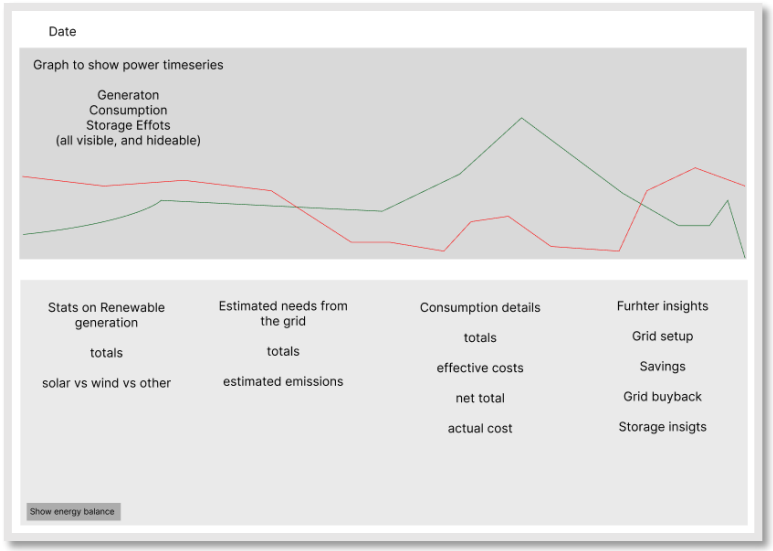


Figure 8 - General Reporting wireframe

4 IMPLEMENTATION: MICROSIM DEV

The section that follows will give a complete summary of the adopted solution, which was methodically built based on extensive background study, to meet the suggested problem. This solution is christened “MicroSim Dev”, denoting a microgrid development and simulation suite dedicated to the exploration of diverse microgrid configurations and design facets. The development of MicroSim Dev represents the culmination of several months of rigorous iterative development, rigorous testing, and the invaluable integration of feedback.

This section will go into the solution’s complexities, providing in-depth insights with references to code snippets and graphical diagrams that clarify the processes and approaches used to create the application. The explicit purpose of MicroSim Dev’s engineering was to provide a thorough response to the problems stated at the opening of this dissertation. It functions in an open-ended way, exhibiting the wide-ranging potential inherent to applications for microgrid design study. Case studies will be used to support the accomplishments and demonstrate how well they addressed the project’s defined objectives.

While MicroSim Dev strives to encompass a substantial portion of the design and pertinent background research, it is essential to acknowledge the presence of numerous avenues for potential enhancements. This is largely attributable to the sheer scope of the subject matter and the virtually boundless array of specific research subtopics that may emerge within this multifaceted project. These avenues can easily be explored in future, and seamlessly integrated into MicroSim Dev, thanks to its backend.

4.1 FUNDAMENTAL ARCHITECTURE

Several fundamental decisions underlie the workings of MicroSim Dev's backend, and it is crucial to understand these before delving into a detailed analysis. Firstly, the representation of timeseries data occurs in half-hour intervals, a choice aligned with available data and the belief that shorter intervals may not significantly enhance the analysis. Within the internal backend, most units for measuring power over time are expressed in kWh, and when units are displayed, they are accompanied by the appropriate scale denotation. It's important to acknowledge that many objects created for the application's internal mechanisms contain non-mentionable code, encompassing functions like getter and setter methods, object viewing, and permission control. In the context of this microgrid solution, it is assumed that it operates within a complete shared community. Here, all resources and grid-associated assets have the ability to pull and rely on any other asset within the grid for power usage or generation, reflecting a shared and interconnected power system. Notably, the entirety of MicroSim Dev spans approximately 11,000 lines of source code, highlighting the depth of its development.

4.2 Abstract Classes

4.2.1 EnergyOut

```
10 references
internal abstract class EnergyOut
{
    //in something..
    protected int dailyEmissions;
    //watts
    //protected int dailyGeneration;

    //watts capacity
    protected int watts;

    //Return the generic daily emissions
    4 references
    public abstract int getDailyEmissions();

    //Return the generic daily consumption
    6 references
    public abstract int getDailyConsumption();

    //Return the description of potential array
    5 references
    public abstract string getArrayDescription();

    //Return the desciption of the "product"
    6 references
    public abstract string getProductDecription();
}
```

Figure 9 - Energy Out class definiton

The EnergyOut class, which is the abstract class for objects that consume power (I.e., energy taken out of the grid), is defined in Figure 9. A common function between EnergyOut and EnergyIn (Figure 10) is the inclusion of generic returns, at a base level. In most cases, reporting was expected to be done on a daily level at lowest, but eventually timeseries became more of a prominent feature of reporting, and as evident from the design, was built into the sub classes of EnergyOut, to cater for the specific parsing of each type of available data.

The *EnergyIn* object as described in Figure 10 provides a blueprint for renewable energy generation objects. These objects are more loosely defined, in comparison to its counterparts *EnergyOut* objects. As many types of commercial solar panels and wind turbines are available, more abstract fields are implemented to support child classes, and make computation easier when it comes to grouping solar panels and wind turbines (and any other renewable generation objects to be developed in future) together.

4.2.2 EnergyIn

```
12 references
abstract class EnergyIn
{
    //Daily emmisiosn (if necessary)
    protected int dailyEmissions;
    //watts capacity
    protected int watts;
    //Price of asset
    6 references
    public decimal Price { get; set; }
    //wattage of asset
    1 reference
    public int Watts { get => watts; }
    //asset may be an array of objects
    protected bool isArray;
    //get daily emissions, which is normally 0 for renewable energy sources.
    7 references
    public abstract int getDailyEmissions();
    //Get specific timeseries point (at a time, and point)
    10 references
    public abstract int getHalfHourlyGeneration(string currTime, int iterationNo);
    //Get generic daily generation
    7 references
    public abstract int getDailyGeneration();
    //get array description
    8 references
    public abstract string getArrayDescription();
    //get asset definition
    8 references
    public abstract string getProductDecription();
}
```

Figure 10 - EnergyIn class definition

4.2.3 EnergyStorageUnit

Energy Storage Units are abstract objects to model any power storage off of. The class (Figure 11) provides a common structure for modelling energy storage units, including properties for capacity, charge level, and price, as well as abstract methods for charging, discharging, and obtaining a description. Derived classes are expected to implement these abstract methods to define the behaviour and characteristics of individual energy storage units within MicroSim Dev.

```
6 references
public abstract class EnergyStorageUnit
{
    // Capacity of storage in W
    12 references
    public double Capacity { get; protected set; }
    //Current charge level (from 0W to capacity typically)
    10 references
    public double ChargeLevel { get; protected set; }
    //price of asset
    3 references
    public decimal Price { get; protected set; }
    // Charge returns energy not charged
    2 references
    public abstract double Charge(double amount);
    //Discharge returns discharged energy
    2 references
    public abstract double Discharge(double amount);
    //get description of unit
    2 references
    public abstract string getDescription();
}
```

Figure 11 - Energy Storage Unit class definition

4.2.4 Initial Cache with Abstract classes

After implementing the base abstract classes, global lists can now be initialized in a static class known as the "Cache" to store these objects. At its infancy, this cache is simply a few lists, however it will grow as more pertinent features are developed. It is simply an internal static class, being uniquely instantiated, which contains all important microgrid information. It is also utilized as a central storage of configuration for objects that are defined later, and pre-loaded data. The following screenshot shows simple initialization of these lists.

(Note that smaller snippets of code will not be labelled as a figure henceforth).

```
99+ references
internal static class Cache
{
    //storing all data for the microgrid

    //energy generation objects
    public static List<EnergyIn> genListin = new List<EnergyIn>();

    //Energy consumer objects
    public static List<EnergyOut> genListOut = new List<EnergyOut>();

    //Energy Storage objects
    public static List<EnergyStorageUnit> energyStorageUnits = new List<EnergyStorageUnit>();
```

4.3 Solar Power

The ensuing definitions encapsulate critical aspects related to solar panel objects, sunlight profiling, and their respective modelling within the simulation, contingent upon the time of day and the occurrences of sunrise and sunset.

4.3.1 SolarPanel class definition

In the context of MicroSim Dev, solar panel objects represent physical photovoltaic (PV) panels employed within the microgrid. These objects embody the real-world counterparts and serve as energy generation units harnessing solar radiation to produce electricity. SolarPanel objects are child classes of EnergyIn object, as they provide energy to the grid. It's constructor, which takes all necessary information for defining this object, is defined by the code snippet in Figure 12. Only the significant aspects that contribute to electricity output are given, even though solar panels generally have numerous noticeable qualities. To set up an object, the object function SolarPanel is defined to require each contributing property.

```
2 references
public SolarPanel(string _type, float _width, float _length, int _watts, int _cells, string _material, decimal _price, int _hours)
{
    type = _type;
    width = _width;
    length = _length;
    watts = _watts;
    cells = _cells;
    material = _material;
    dailyEmissions = 0;
    isArray = false;
    Price = _price;
    hours_till_efficient = _hours;
}
```

Figure 12 - Solar Panel Constructor

4.3.1.1 Panel Arrays

SolarPanelArray objects simply represent a panel of a particular definition, however a quantity can be provided to simulate more than one of the desired panels, without unnecessarily creating many objects and references to objects. This saves greatly on computation power when dealing with large

amounts panel objects on the grid. Figure 13 shows the constructor for this object, and the method which computes timeseries data, which is discussed further in 4.3.4.

```
9 references
internal class SolarPanelArray : SolarPanel
{
    int amount;

    1 reference
    public int Amount { get => amount; }

    2 references
    public SolarPanelArray(string _type, float _width, float _length, int _watts, int _cells, string _material, int amount, decimal _price, int _hours)
        : base(_type, _width, _length, _watts, _cells, _material, _price, _hours)
    {
        this.amount = amount;
        isArray = true;
        Price = amount * _price;
    }

    7 references
    public override int getHalfHourlyGeneration(string currTime, int iterationNo)
    {
        int total = 0;
        for (int i = 0; i < amount; i++) total += base.getHalfHourlyGeneration(currTime, iterationNo);
        return total;
    }
```

Figure 13 - Solar Panel Array Constructor

4.3.2 User Interface for adding solar

A fundamental concept of this application and the overarching research project is to protect user freedom. Therefore, the GUI design proposal presented in section 3.2.3.2 which articulates the interface designed to empower users, is purposely engineered to allow substantial customizability. This interface facilitates the addition of solar generation assets grants users the flexibility to select from predefined panel configurations. It is crucial to emphasize that every property associated with the SolarPanel object is fully editable. In essence, users have the right to customise each characteristic to meet their own needs and expectations. Consequently, upon entering the requisite data, a SolarPanel object is instantaneously generated, modelling the user-defined specifications. This approach empowers users to exercise a high degree of control and customization, aligning the application with their specific needs and facilitating an enriched exploration of microgrid configurations. Figure 14 shows a side-by-side example of the preset solar panels within the interface.

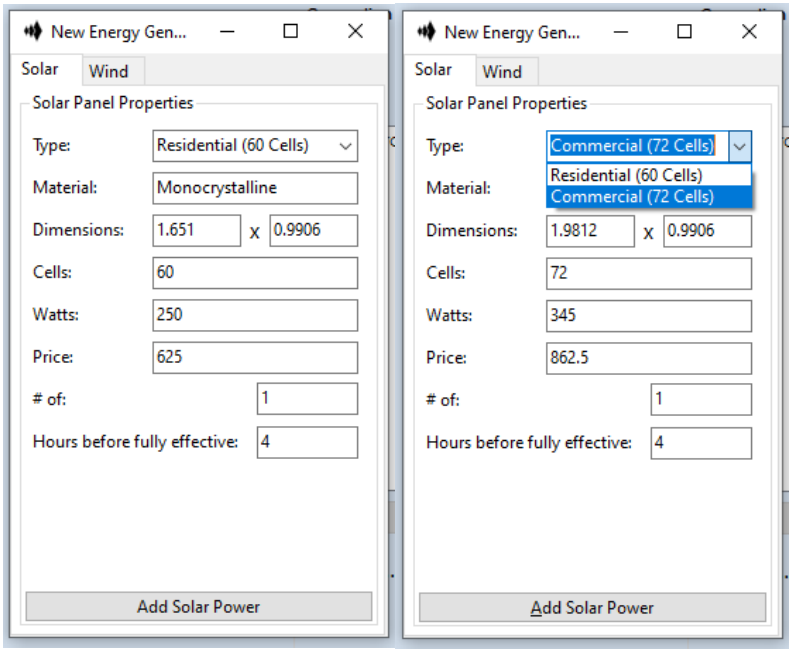


Figure 14 - Residential and Commercial solar panel presets

4.3.3 Sunlight Profiling

In order to accurately store, and utilize sunlight profiling, another class was introduced, to serve as a unique “DateTime” object. Termed as a “DayRiseTimes”, it is defined in Figure 15.

```
10 references
internal class DayRiseTimes
{
    //Time of Year
    private int month;
    private int day;
    //Sunrise and sunset specific times
    private DateTime sunrise;
    private DateTime sunset;
    //Timespan defining how much daylight we typically have
    private TimeSpan daylight;
    2 references
    public DayRiseTimes(DateTime sunrise, DateTime sunset, TimeSpan daylight)
    {
        this.month = sunrise.Month;
        this.day = sunrise.Day;
        this.sunrise = sunrise;
        this.sunset = sunset;
        this.daylight = daylight;
    }

    16 references
    public int Month { get => month;}
    16 references
    public int Day { get => day;}
    9 references
    public int Year { get => sunrise.Year; }
```

Figure 15 - DayRiseTime Class definition

This object exclusively specifies the time of year, devoid of a specific year, as seasons cyclically repeat on an annual basis. For each day to be defined by the object, users have the flexibility to define and customize the precise times of sunrise and sunset as needed. Within this simulation, the application undertakes the calculation of solar performance, factoring in the available daylight hours for the current day. Notably, the data representing sunrise and sunset times is inherently location-dependent, exhibiting significant variations based on geographical coordinates. MicroSim Dev, in its current configuration, accommodates sunlight patterns specific to the Waikato region, accounting for daylight savings considerations.

This data structure, crafted for daylight modelling, allows for the storage of a list of these objects within the *Cache*. By populating this list with the requisite data, MicroSim Dev effectively assembles a sunlight profile spanning the entire year. This holistic approach enables the application to operate seamlessly, delivering precise and location-specific solar performance modelling throughout the annual cycle. Figure 16 represents daylight profiling in the *Cache*.

```
//Default DRT is Curr Time with no sunrise or sunset, but is overwritten promptly by form1 (Main form)
public static DayRiseTimes currDay = new DayRiseTimes(DateTime.Now, DateTime.Now, TimeSpan.Zero);

//Daylight Profile (Default is Waikato Region)
public static List<DayRiseTimes> yearSunTimes = new List<DayRiseTimes>();
```

Figure 16 - additional aspects to Cache

4.3.4 Generating timeseries solar data

All of the defined solar features in MicroSim Dev come together to produce data points that will be used to make calculations and to facilitate power modelling. The application has extensive object information that users have customized to meet their own requirements, coupled with data on available sunlight profiles, allowing for the modelling of crucial time series. The forthcoming section will delve into the abstract method implementation for *getHalfHourlyGeneration*, further describing the high-level algorithm displayed in Figure 17.

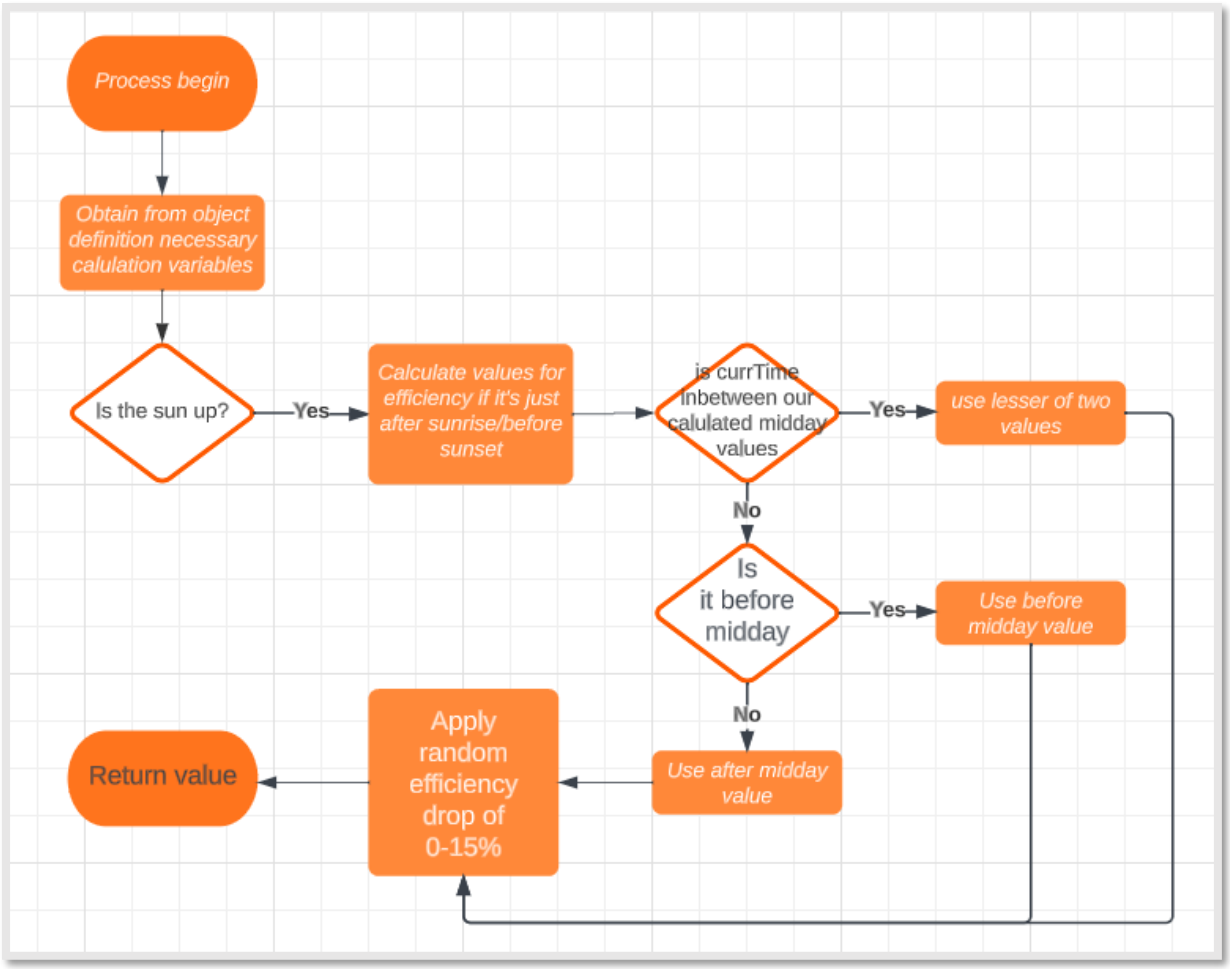


Figure 17 - Solar timeseries logic flowchart

This shows an overview of the decision making that a solar panel object will go through when generating time series. It begins with collating references to all necessary data for calculations, the following code snippet represents this.

```
//base gen is wattage / 2 as this is half an hour of power generation.
float baseGen = watts / 2;
//A 1/100 of baseGen is acquired to emulate random dips in performance
float hundredthOfBase = watts / 100;
//Create dateTime to represent currTime
DateTime currentDateTime = DateTime.Parse(currTime);
int secconds = hours_till_efficient * 3600;
//time to start counting down to sunset instead of up after sunrise
DateTime midday1 = new DateTime(Cache.currDay.Year, Cache.currDay.Month, Cache.currDay.Day, Cache.currDay.getSunrise().Hour + hours_till_efficient, 0, 0);
DateTime midday2 = new DateTime(Cache.currDay.Year, Cache.currDay.Month, Cache.currDay.Day, Cache.currDay.getSunset().Hour - hours_till_efficient, 0, 0);
```

The final two lines, denoting two midday values are the times of day, after sunrise, in which the solar panel is classified as “fully efficient”, based on solar radiation from the sun. This is a field that is defined when creating the panel object, but it normally sits at around 4-5 hours. These two midday values will be used shortly. The algorithm ensures the sun is up, then calculates the relative times (time after

sunrise and time before sunset). This approach was taken to closely isolate edge cases when shorter days are present, and when the line between closeness to sunrise or sunset becomes blurred.

```
//If the sun is up
if (Cache.currDay.getSunrise() < currentDateTime & currentDateTime < Cache.currDay.getSunset())
{
    //we won't be at peak power until its X hours after sunrise
    //timespans for measuring where we are in the day in relation to sunrise and set
    TimeSpan afterRise = currentDateTime - Cache.currDay.getSunrise();
    TimeSpan beforeSet = Cache.currDay.getSunset() - currentDateTime;
```

In order to model the efficiency ramp-up of the solar panel, a sine wave is used to give the base values between sunrise and full efficiency, and vice versa for sunset. In order to transform the sine wave to suit the given output of the panel, a transformation sine wave equation is utilized.

$$y = a\,.\sin\big(b\,(x - c)\big) + d$$

The implementation of this transformation equation within the algorithm goes a little deeper, as the values being used must be accurately calculated. To model a sine wave, starting at 0, ending at the "peak efficiency", the first things to define is maximum (peak) and minimum (0) values. The amplitude is calculated as $a = (max - min)/2$. As the amplitude increases, vertical shift must appropriately be accounted for, so it is calculated in relation to amplitude: $d = max - a$. Wavelength is derived by time through the formula, where t is the number of seconds it takes before the panel is fully efficient. Horizontal shift is calculated as half of t, or $c = 0.5t$. The algorithm initializes these values as shown:

```
//Sin wave definition (for smooth ramping up and down, shape like a sin wave)
float max = (int)baseGen;
float min = 0;
float a = (max - min) / 2;
float b = (float)(Math.PI) / t;
float c = 0.5f * t;
float d = max - a;
```

The *y* value to return for the timeseries, or estimated performance at that particular time of day, can then be calculated through *sin* functions, using the native C# math library. The overall implementation and usage of the *sin* function represents a different implementation of the smooth step function, which is a commonly used function in graphics and shading engines.

```
//now depending on what the time is we decide which value to go with
//c += 0.5f * p;
int rampedValue1 = (int)(a * Math.Sin(b * (afterRise.TotalSeconds - c)) + d);
int rampedValue2 = (int)(a * Math.Sin(b * (beforeSet.TotalSeconds - c)) + d);
```

Next, the algorithm uses the following simple logic determine which *rampedValue* to return to the user, depending on the time of day. This logic is better described in Figure 17.

```
baseGen = (currentDateTime < midday1 & currentDateTime > midday2) ? Math.Min(rampedValue1, rampedValue2) :
    (currentDateTime < midday1) ? rampedValue1 :
    (currentDateTime > midday2) ? rampedValue2 :
    baseGen;
```

The sine wave computations are small, and consequentially can both be executed simultaneously before this logic takes place. The logic is handled by a large ternary operator, which performs better under stress than general 'if' statements, so in the case of larger batches of data being generated, this algorithm handles well. Finally, the base value that is left is altered at a random amount from 0-15%

efficiency lost before being returned. This is to simulate the uncertainties that occur when using these objects in the real world.

```
//Random performance drop of 0-15%
float drop = r.Next(0, 15) * hundredthOfBase;
return (int)baseGen - (int)drop;
```

It should be noted that the efficiency drop was originally intended to serve as a placeholder for things like cloud cover and weather effects on the panels. However, MicroSim Dev has not fully explored this particular avenue, so the placeholder still acts as a random calculation over a determined deficit based on weather. Due to the random values used each time the return value is calculated, the curves produced by this technique are more like "suggestions" than precise predictions.

4.4 Wind Power

The ensuing definitions encapsulate critical aspects related to wind turbine objects, wind speed modelling, and the turbines' respective modelling within the simulation, contingent upon current windspeed.

4.4.1 Turbine class definitions

Within the framework of MicroSim Dev, the definition of multiple objects for wind turbines has been undertaken. At the core of wind energy modelling are the WindTurbine objects, serving as the foundational class for generic wind turbines. These generic turbines use more complex mathematical calculations to estimate the power output unique to each turbine because they lack linked established power curves.

In contrast, the WindTurbineExisting objects belong to a class of pre-defined turbines equipped with associated power curves. This feature facilitates a closer and more precise modelling of performance contingent upon wind speed, thus reducing reliance on complex power calculations. In the case that the existing turbines are not provided with curves to estimate on power performance, the mathematical computations in the base class are utilized.

Similarly, akin to the SolarPanelArray, the WindTurbineArray class operates as a versatile medium for representing multiple turbine objects, all while maintaining a low processing overhead. This structural hierarchy enables MicroSim Dev to conduct comprehensive wind energy simulations and power modelling with efficiency and flexibility.

4.4.1.1 WindTurbine

Figure 18 shows the definition of a generic turbine. All critical information pertinent to calculating the power output of this turbine is definable by users. The mathematical equation for determining said power is covered in 4.4.5. Again, this is the baseline definition, and is not normally represented on the grid.

```
7 references
internal class WindTurbine : EnergyIn
{
    //m
    protected int rotorDiameter;
    //degrees from north (90 being east, 180 being south, etc)
    protected int rotation;
    //name
    protected string name;
    //kmh
    protected int cutoff;
    protected int rated;
    protected int cutin;
    //general effectiveness
    protected double effectiveness;


    3 references
    public WindTurbine(int watts, int rotorDiameter, int rotation, string name, int cutoff, int rated, int cutin)
    {
        this.watts = watts;
        this.rotorDiameter = rotorDiameter;
        isArray = false;
        this.rotation = rotation;
        this.name = name;
        effectiveness = 0.2;
        this.cutoff = cutoff;
        this.rated = rated;
        this.cutin = cutin;
    }
```

Figure 18 - WindTurbine class definition

4.4.1.2 WindTurbineExisting

WindTurbineExisting objects represent wind turbines of refined and more specific design. This definition was guided by the data available for setting up pre-defined turbines to the users. Figure 19 shows this definition, which has become the default object in the application, to allow users to have more customizability when it comes to defining their turbine objects. These objects are the main representations of wind power on the microgrid.

```
18 references
internal class WindTurbineExisting : EnergyIn
{
    //general details
    private string name;
    private string manufacturer;
    //in kW
    private int ratedPower;
    //in m
    private double rotorDiameter;
    //in m^2
    private double sweptArea;
    private int numBlades;

    //in m/s
    private double cutInSpeed;
    private double ratedSpeed;
    private double cutOutSpeed;

    //power curve
    private Dictionary<double, int> curve = null;
```

Figure 19 - WindTurbineExisting class definition

4.4.1.3 WindTurbinePowerCurve

In tandem with the WindTurbineExisting class, the WindTurbinePowerCurve objects assume the role of representing power curves tailored to specific turbines. Figure 20 shows their implementation, and the like terms they share with WindTurbineExisting objects. These power curves can be seamlessly

associated with existing turbines, fostering a symbiotic relationship that collectively aids in the precise calculation of a turbine's power output.

While these WindTurbinePowerCurve objects are conceptually distinct, they share a common object definition for the curve data, akin to the WindTurbineExisting objects. This shared format allows for the straightforward establishment of references/associations between the two, streamlining the process of incorporating and leveraging power curves in the power output calculations for wind turbines.

```
5 references
internal class WindTurbinePowerCurve
{
    private string manufacturer;
    private string name;
    //double being windspeed in m/s and int being kW at that windspeed
    private Dictionary<double, int> curve = new Dictionary<double, int>();

    0 references
    public string Manufacturer { get => manufacturer; set => manufacturer = value; }
    0 references
    public string Name { get => name; set => name = value; }
    1 reference
    public Dictionary<double, int> Curve { get => curve; set => curve = value; }

    /// <summary>
    /// New power curve
    /// </summary>
    /// <param name="manufacturer">name of manufacturer</param>
    /// <param name="name">model number/name of turbine</param>
    1 reference
    public WindTurbinePowerCurve(string manufacturer, string name)
    {
        this.manufacturer = manufacturer;
        this.name = name;
    }
```

Figure 20 - WindTurbinePowerCurve class definition

The application will simply match up the manufacturer and name of turbine to that of the defined curve and associate the two together. Existing turbines, and associated wind turbine power curves are married up on startup, and are stored in the cache, as preset turbines the users can select with ease*. A list for preset WindTurbinePowerCurve objects is also available, in case in future these are needed.

In Cache:

```
//Lists for storing Avaialble Generators
public static List<WindTurbineExisting> availableTurbines = new List<WindTurbineExisting>();

//list of power curves... Just incase
public static List<WindTurbinePowerCurve> curves = new List<WindTurbinePowerCurve>();
```

Using a list of preset objects saves on processing power, as the application can point to pre-existing references of these objects, instead of creating new definitions for these objects.

4.4.2 User interface for adding wind power

The same form, which is like the SolarPanel interface, is employed to enable complete user flexibility and give users the ability to specify their turbine characteristics. This is depicted below in Figure 21.

*http://www.thewindpower.net for the wind turbine and power curve samples – original data sample obtained here.

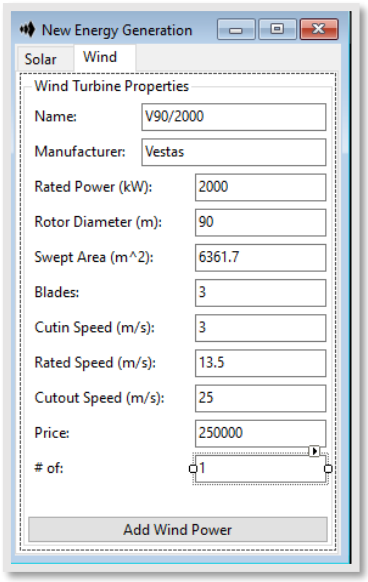


Figure 21 - Wind Turbine addition form and preset

The users can add preset wind turbines simply, without having to complete through forms and dialogues, by using a MenuStrip object that is dynamically updated to incorporate the preset turbines. This MenuStrip is located on the main form, shown in Figure 22.

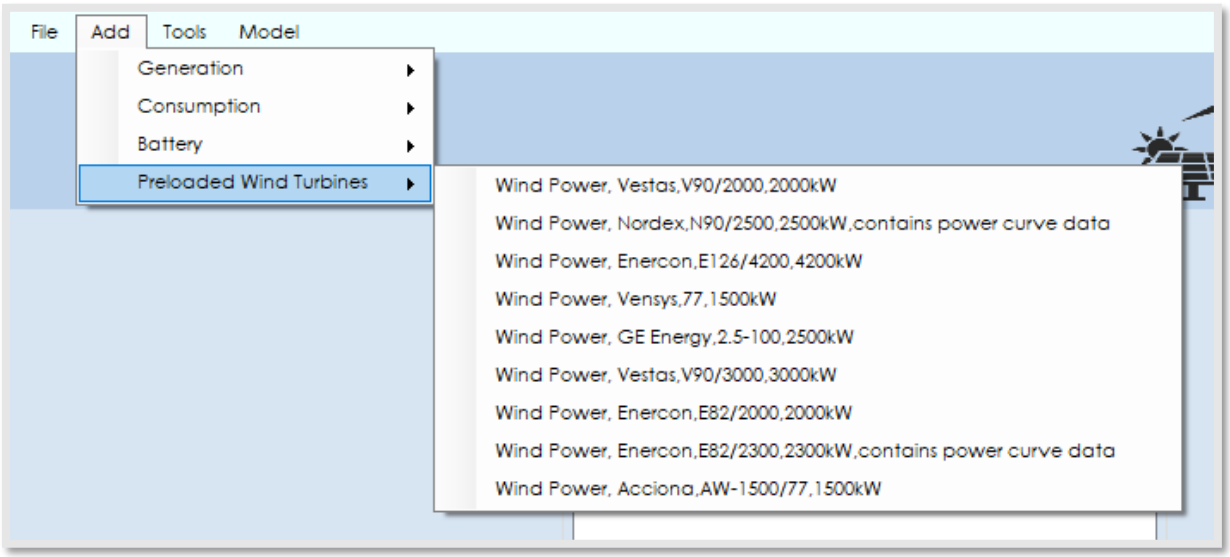


Figure 22 - Adding preset turbines, and seeing which ones have power curves

4.4.3 Wind Speed Modelling

Due to the unavailability of recent real-time wind speed data in the Waikato Region, which corresponds with the default sunlight patterns used in the application, MicroSim Dev adopts an alternative approach by modelling wind speed variations throughout the day. This modelling is based on the utilization of a Weibull distribution, which was dissected in section 2.4.3.

To facilitate this modelling, a dedicated class is employed to encapsulate wind speeds spanning a full year's duration. This class can be stored within the application's *Cache*. When wind speeds are required for modelling purposes, they can be readily retrieved from this object by specifying the relevant date. It is important to note that as this approach involves generating data based on a

statistical distribution, the results will exhibit inherent variability. Consequently, a wind speed cache is established over the course of the application's lifespan. The underlying algorithm for retrieving wind speeds operates in two modes: it either retrieves the data from the wind speed cache if available, or dynamically generates new results, stores them within the cache, and subsequently returns them as needed. This adaptive approach ensures the availability of wind speed data for modelling, enhancing the application's fidelity in simulating wind energy scenarios.

The application utilizes a C# Weibull library for carrying out the calculations, however within the base class, complex calculations take place for returning results from the distribution. The Weibull distribution is initialized with a hard-coded shape $a = 2$. The spread of the distribution is determined by averaging the provided max and min wind speeds, $b = (max + min)/2$.

Samples are based on the distribution formed by the following equation, where y is the probability of the windspeed occurring, and x is a random double, which represents wind speed, generated by the Weibull class:

$$y = \frac{a}{b^a} x^{a-1} e^{-(\frac{x}{b})^a}$$

A sample based on these calculations is returned by the Weibull Class and is subsequently included in the windspeed models.

4.4.4 Altering wind speed

A wind speed tool was created for testing which has ended up serving as a useful tool for users to customise the simulation's parameters further. However, in its current form, this tool just allows users to view the current wind speeds throughout the current day and draw their preferred curve or alter specific values with their mouse. In the future, this tool may be expanded into a full weather defining tool. Figure 23 shows this tool in usage, where the user has seen the wind speeds, and changed them to their liking. This will have an immediate effect on how wind turbines calculate power, and changes can be observed in real time when other tools are used simultaneously with this one.

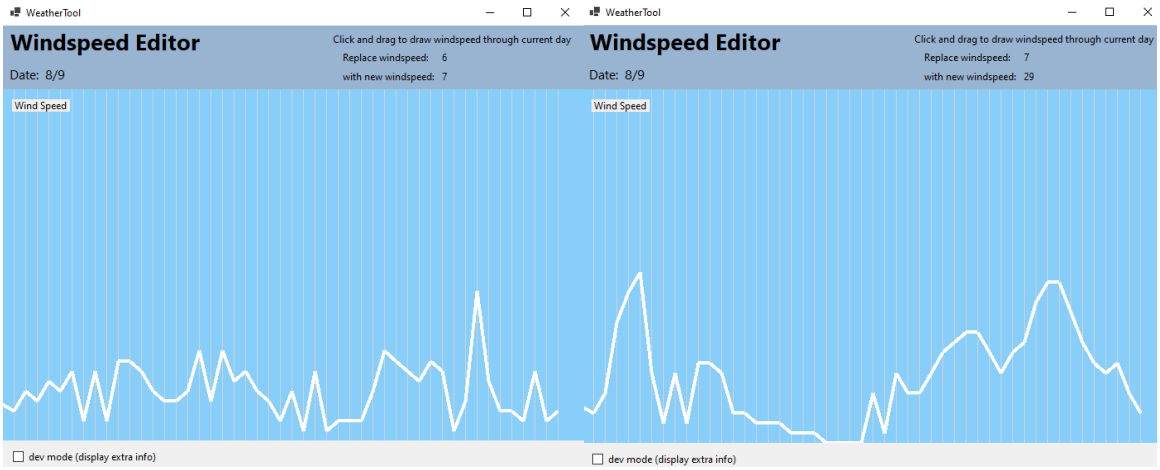


Figure 23 - Windspeed Editor examples

4.4.5 Generating timeseries wind data

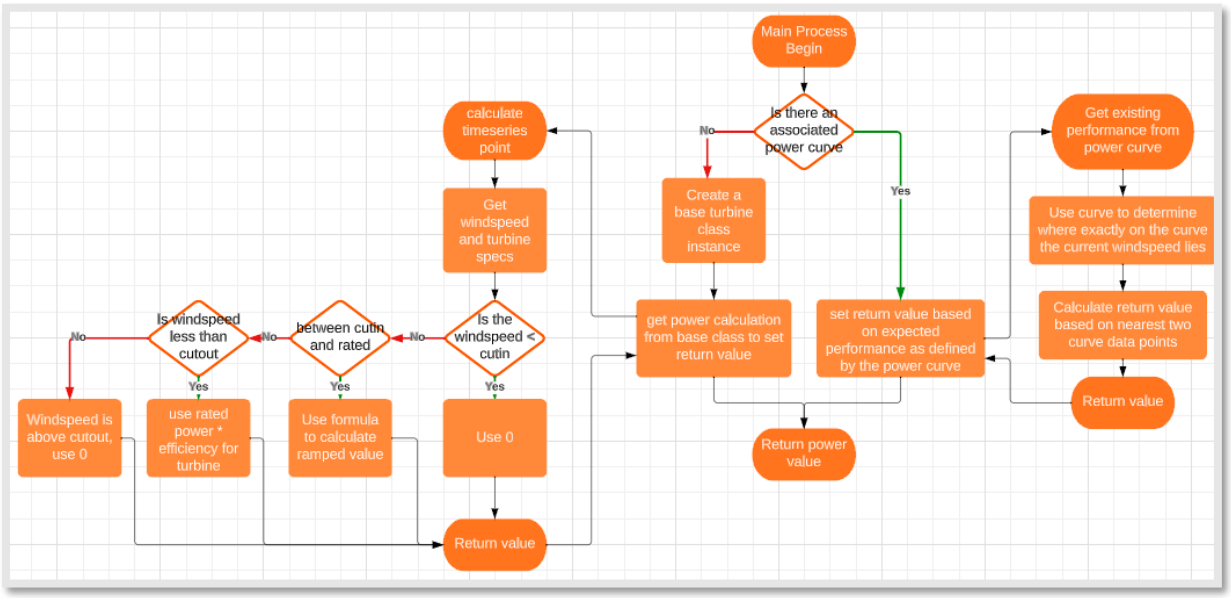


Figure 24 - Wind timeseries logic flowchart

Figure 24 shows a high-level overview of the algorithm used for accurately determining the windspeed timeseries datum to return, based on all available parameters considered thus far. The first check is to confirm whether the turbine has a power curve. If a curve is present, the windspeed is obtained and the following algorithm is executed (Figure 25).

```
/// <summary>
/// Returns the performance of the turbine at the current windspeed
/// </summary>
/// <param name="key">windspeed in m/s</param>
/// <returns></returns>
1 reference
public int getExistingPerformance(double windspeed)
{
    //only if we have a power curve do we return something
    if (windspeed > 35)
    {
        return 0;
    }

    //get nearest .5 below and above
    double lower = rounder(windspeed);
    double upper = rounder(windspeed + 0.5);
    //Ensure we are in the right .5 interval
    if (lower > windspeed)
    {
        upper = lower;
        lower = rounder(windspeed - 0.5);
    }
    //We have no values above 35, so set it as the hard limit
    if (upper > 35)
    {
        upper = 35;
    }
    int upperWattage = curve[upper];
    int lowerWattage = curve[lower];
    //Simple maths. Rise/run
    int rise = Math.Abs(upperWattage-lowerWattage);
    double run = 0.5;
    int val = (int)((rise / run) * windspeed);
    if (isArray) return val * amount;
    return val;
```

Figure 25 - getExistingPerformance algorithm

The algorithm must estimate the return values based on the two closest points if the windspeed is between two of the power curves, which are defined at 0.5m intervals for each datum. The midpoint between the two curve data points is determined from the gradient formed between the two points. If the turbine does not have an associated power curve, it instead opts to run more specific calculations to determine the return value. It must gather all the relevant information before starting the calculation, which is shown here:

A notable mention from the following code snippet, the application makes sure to consider the Betz limit, which is the theoretical maximum efficiency of a turbine, regarding how much kinetic energy can be used to spin turbines. This is applied directly to windspeed before the power calculation, but after cutin/cutout logic.

```
string justDate = currTime.Split(' ')[0];
List<double> windSpeeds = Cache.windModel.getDailySpeeds(justDate);

//get the current windspeed
double windspeed = windSpeeds[iterationNo];
//change m/s into km/h
double kmhWindSpeed = windspeed * 3.6;

//calculate area
double rotorRadius = rotorDiameter / 2;
double area = Math.PI * (rotorRadius * rotorRadius);

//best case cut in halfhourly
double peakPerformance = watts * 0.5;
```

After that, the high-level algorithm's logic is used to choose the appropriate return value. The turbine power calculation is used when the wind speed is below the turbine's rated windspeed, which is the

most notable path. Since area is already known, this formula is compressed.

$$P = \frac{1}{2} * p * A * v^3$$

P is the power generated by in watts, p is the air density in kg/m3, A is the swept area of the turbine and v is the current windspeed. The below code shows this formula in MicroSim Dev.

```
//Power(W) = 1 / 2 x ρ x A x v (not cubed3)
//calculate available power estimate
//using constant for wind density so far
double currPerformance = 0.5 * 1.225 * area * (windspeed*windspeed*windspeed);
//get the half-hourly value
currPerformance = currPerformance / 2;
return (int)Math.Min(peakPerformance,currPerformance);
```

To ensure that irrational power levels are not computed, with respected to the users decided rated windspeed speed for the turbine, the algorithm conducts the calculation, converts it to a half-hour value and returns the lower value of the calculated performance compared to peak performance.

4.5 House Models

The integration of housing components into MicroSim Dev aligns intricately with the design principles and objectives delineated in point 3.2.3.3. It is worth noting that these house model objects' final design reflects an evolution and refinement process, which may deviate from the initial design decisions initially set forth. These deviations stem from iterative development, and feedback-driven adjustments, all aimed at achieving the most effective and user-friendly housing modelling features within the application. Being the first implementation of the abstract EnergyOut class and relying less on complex calculation algorithms to determine load at specific points, it lays the base groundwork for representing any object that could be on the grid as a consumer/customer.

4.5.1 HouseModel class definition

```
24 references
internal class HouseModel : EnergyOut
{
    //Stores and has calculations for a model of a home that consumes power and stuff
    private bool heatpump;
    //ck is cooking
    private bool ckElectric;
    private bool ckMains;
    private bool ckBottledGas;
    //wh is water heating
    private bool whMains;
    private bool whElectric;
    private bool whWood;
    //sh is space heating
    private bool shElectric;
    private bool shMains;
    private bool shWood;
    private bool shFossilFuels;

    private int adults;
    private int children;
    //In thousands
    private string income;
    //city vs rural
    private string location;

    //is it a special house
    private bool totalsHouse;
    //special values for special house
    private double averageElectric;
    private double averageMains;
    private double averageOther;

    //Dictionary to store days, and watt consumption against time (in 48 half hours)
    public Dictionary<string, List<int>> consumptionData;
```

The definition of a house model is shown on the left. The housing information offered includes detailed explanations of how the house functions, mainly in terms of heating and cooking. Each of these criteria were included in the HouseModel description because they are essential contributing components to the load profiles of a typical home, and in the inclusion the objects further facilitate user freedom. The housing consumption data is simply recorded as a dictionary (key-value storage), where the load curve for each day is represented by a list of integers, and the key to identify this list is a date, which is a string.

4.5.2 Preset HouseModels

With this data structure initialized, MicroSim Dev, on startup will load all available data into the application and allow users to select from the preset house models. These “Available Houses”, which is how MicroSim Dev names them, are stored in the *Cache* as a list.

```
//Complex Houses (availalbe)
public static List<HouseModel> houseModels = new List<HouseModel>();
```

Like the wind turbine presets, an option on the MenuStrip is dynamically updated to allow users to select these presets and add them to the grid with ease (Figure 26).

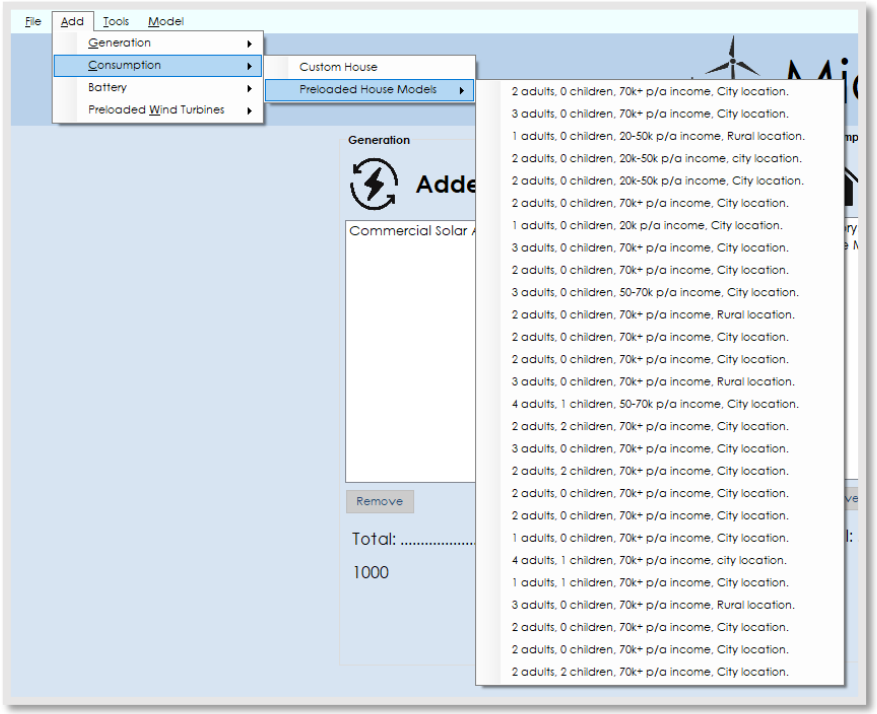


Figure 26 - Preset house model example

4.5.3 User Interface for adding Housing Models

The interface for adding house models is one of the more complicated interfaces for adding assets to the microgrid. This is evident from the initial design but has been amplified somewhat in the implementation. The user interface is shown in Figure 28. Any set of parameters for a home model, as described in 4.5.1, may be defined using it. Then, the desired definition is connected to a load profile. Users have the option to grant MicroSim Dev the judgement to automatically assign a load, which may represent the household definition, based on a determined similarity score between the definition provided by the users and the preset households. Figure 27, which runs when the user clicks the button labelled "Use pre-generated Load" demonstrates this. Using *getHouseModel*(), the application retrieves the user's definition of their home, and then computes similarity scores for each predefined house model. Finally, it allocates the load profile depending on the closest score's index, which is the location in the preset houses list for the similar house definition. If the users want to import their own data, this is possible via the right-hand side of the UI, which guides the user on how to import their own data successfully.

```
HouseModel newHouse = getHouseModel();
if (newHouse == null) return;
//now find some data that best fits this description
List<int> scores = new List<int>();
foreach (HouseModel h in Cache.houseModels)
{
    scores.Add(newHouse.CalculateSimilarityScore(h));
}
//Figure out which house is closest
int maxIndex = 0;
int maxValue = scores[0];

for (int i = 1; i < scores.Count; i++)
{
    if (scores[i] > maxValue)
    {
        maxIndex = i;
        maxValue = scores[i];
    }
}
newHouse.consumptionData = Cache.houseModels[maxIndex].consumptionData;
Cache.genListOut.Add(newHouse);
MessageBox.Show("House model setup with load auto-selected.");
```

Figure 27 - Algorithm for identifying similar loads

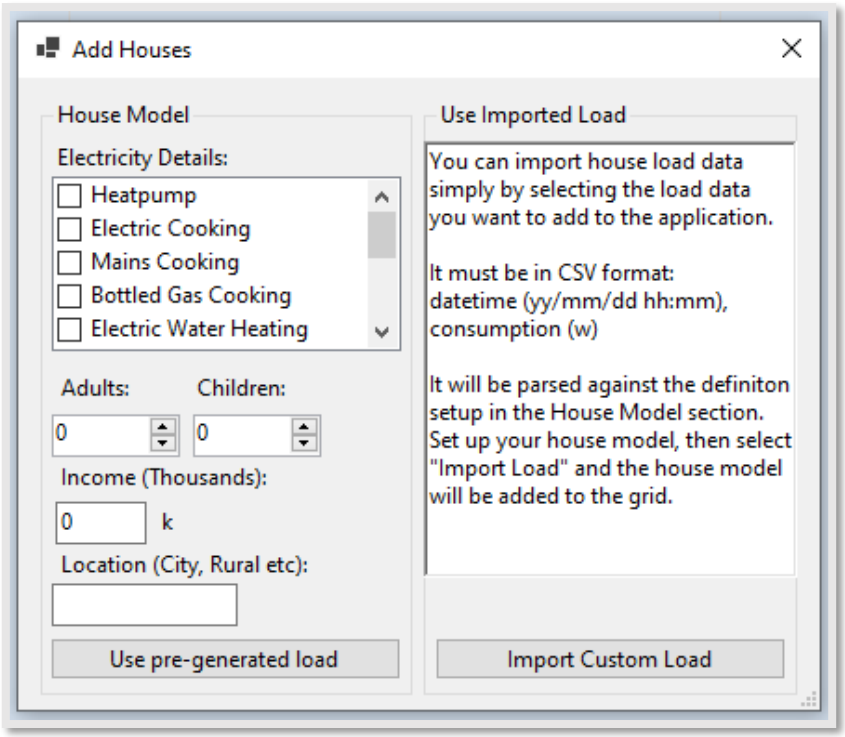


Figure 28 - Custom House model setup form

4.6 Factory Modelling

The factory model definition represents the most intricate class within the application, characterized by its multifaceted functionality. At a high level, this class affords users the flexibility to import their own data, mirroring the capabilities offered by housing models. Alternatively, users can carefully design the load profile by defining machinery or objects, specifying their attributes, and setting operation times. The application then generates a load profile based on these user-defined parameters. While this approach is ideally suited for any consumer connected to the microgrid, it is currently exclusive to factory modelling. Given the paramount importance of factory load profiles in Ahuora's research, this class definition underwent extensive iterative development and refinement to ensure its robustness and precision. It is worth noting that, in the future, the house model class could be similarly expanded and brought up to par with the factory model through the utilization of shared abstract classes and functions, streamlining the development process.

Presently, MicroSim Dev is tailored to scenarios involving a standalone factory within the microgrid. It enables users to customize and explore the ramifications of modifying a single factory, though the capability to add multiple factories simultaneously is not yet supported. This functionality could potentially be achieved by incorporating a list of factory objects in the cache, rather than relying on a single definition. However, this avenue was not pursued in the current phase of development. The factory is represented in the cache as such:

```
//The main factory of the application
public static FactoryModel mainFactory = new FactoryModel("Default Factory");
```

From a backend perspective, the *mainFactory* is integrated into the grid calculations and modelling processes, albeit with no associated data initialized. This means that if users do not intend to simulate

a factory in their analyses, they can simply bypass this feature. Proper initialization from the users is a prerequisite for the factory to contribute to grid loads and outcomes.

4.6.1 FactoryModel class definition

The class definition depicted in Figure 29 will be dissected into digestible segments, ensuring a logical flow from one section to the next. Before delving into further details, here are some overarching points to note:

- Users have the option to assign a name to the factory.
- Functionally, the factory operates in three distinct modes: calculated load profiles, set load profiles, and comprehensive load profiles.
- The internal logic of the factory model will determine the appropriate approach based on the user's defined parameters.

```
8   class FactoryModel : EnergyOut
9   {
10      //variable loads per item.
11      //Flexibility in how a load is defined
12
13      // Machine item Lists. These are kept in sync through methods.
14      private List<Tuple<string, double>> machines;
15      private Dictionary<string, List<Tuple<TimeSpan, TimeSpan>>> machineTimeRanges = new Dictionary<string, List<Tuple<TimeSpan, TimeSpan>>>();
16
17      //generic (or static) load, and associated bool
18      private List<double> genericLoad = new List<double>();
19      private bool usesPreLoad = false;
20
21      //Comprehensive load, and associated bool
22      //Path to data
23      private string compLoadPath = null;
24      //date, data (in half-hour intervals)
25      private Dictionary<string, double[]> comprehensiveLoad = new Dictionary<string,double[]>();
26      private bool usesCompLoad = false;
27
28      // Properties
        8 references
29      public string Name { get; set; }
        2 references
30      public List<Tuple<string, double>> Machines { get => machines; }
        3 references
31      public Dictionary<string, List<Tuple<TimeSpan, TimeSpan>>> MachineTimeRanges { get => machineTimeRanges; }
        5 references
32      public bool UsesCompLoad { get => usesCompLoad; }
        1 reference
33      public string CompLoadPath { get => compLoadPath; }
34
35      // Constructor
        2 references
36      public FactoryModel(string name)
37      {
38          Name = name;
39          machines = new List<Tuple<string, double>>();
40          machineTimeRanges = new Dictionary<string, List<Tuple<TimeSpan, TimeSpan>>>();
41      }
```

Figure 29 - FactoryModel Class definition

4.6.1.1 Calculated load profile

By default, it operates in the first of these modes, which is to base its power consumption off the complex structure defined in lines 14 and 15. Essentially, these two data structures represent power consumption objects, and ranges of times for said object in which it is operational (or consuming power) throughout the day. This is all grounded around a string value, which outlines the name of the object, and acts as the point of parity between the two data structures. (MicroSim Dev refers to them as machines, but they can technically be any appliance or power consuming object).

'*machines'*, line 14:

- *List<Tuple<string, double>>* defines the data type of the field. It's a List<T> where T is a Tuple<string, double>.
- This means that '*machines*' is a collection that can store elements, with each element being a tuple containing two values: a string and a double.

- In other words, '*machines*' is a collection of pairs where each pair consists of a machine name (string), and its associated wattage (double).

'*machineTimeRanges'*, line 15:

- *Dictionary<string, List<Tuple<TimeSpan, TimeSpan>>>* defines the data type of the field. It's a dictionary where keys are strings (string) and values are lists (List<T>) of tuples (<T> = Tuple<TimeSpan, TimeSpan>).
- In other words, *'machineTimeRanges'* is a collection that maps strings to lists of tuples, and each tuple contains two values of type TimeSpan. This data structure is used to store time ranges associated with specific machines (represented as strings, based on 'machines')
- TimeSpan objects are utilized to represent start time, and end time, because arithmetic between the like objects is simple when treated as 24hr time.

Together, lines 14 and 15 initialize a data structure which defines a complex map of machines, and their operating times through the day. Figure 30 has been created to further signify the relationship between the two data structures, showing the lists, tuples, and references to other lists.

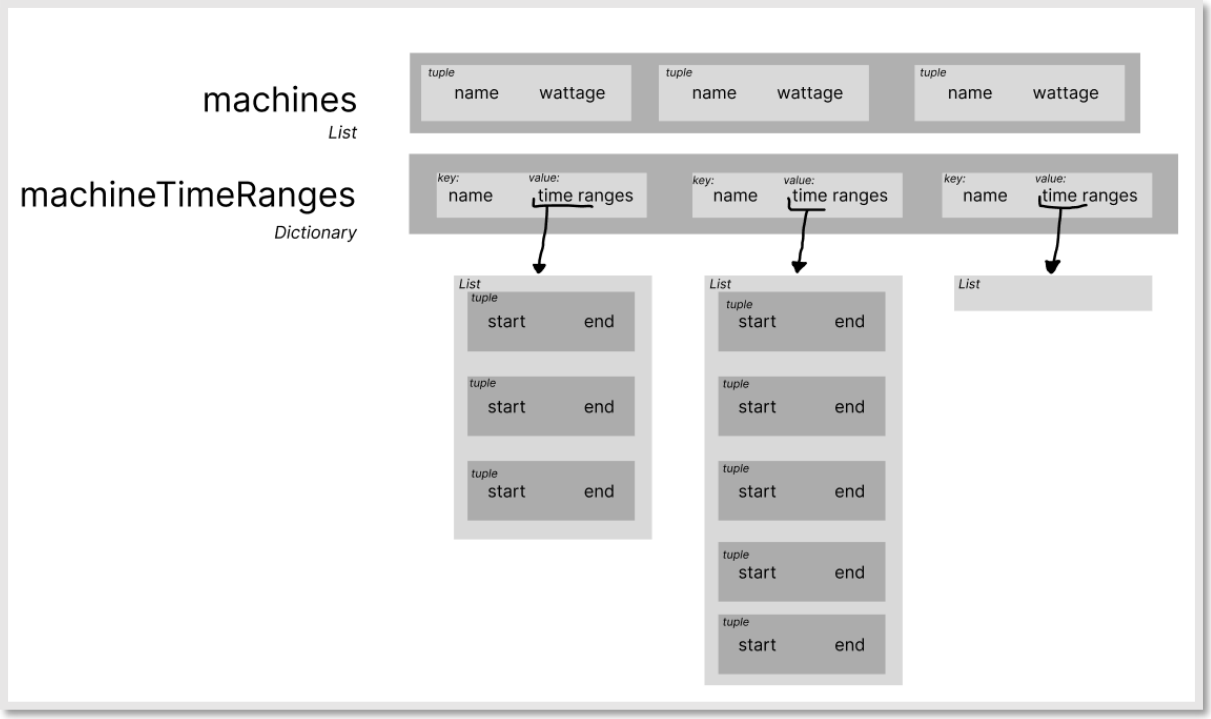


Figure 30 - Datastructure for machines and timeranges

To calculate the load profile for the factory in this mode, the application follows the algorithm depicted in Figure 31. A second set of logic checks determines the condition to see if a machine is in use in the current half-hour, and only returns true or false depending on whether the time falls within the timespans (as contained in the dictionary).

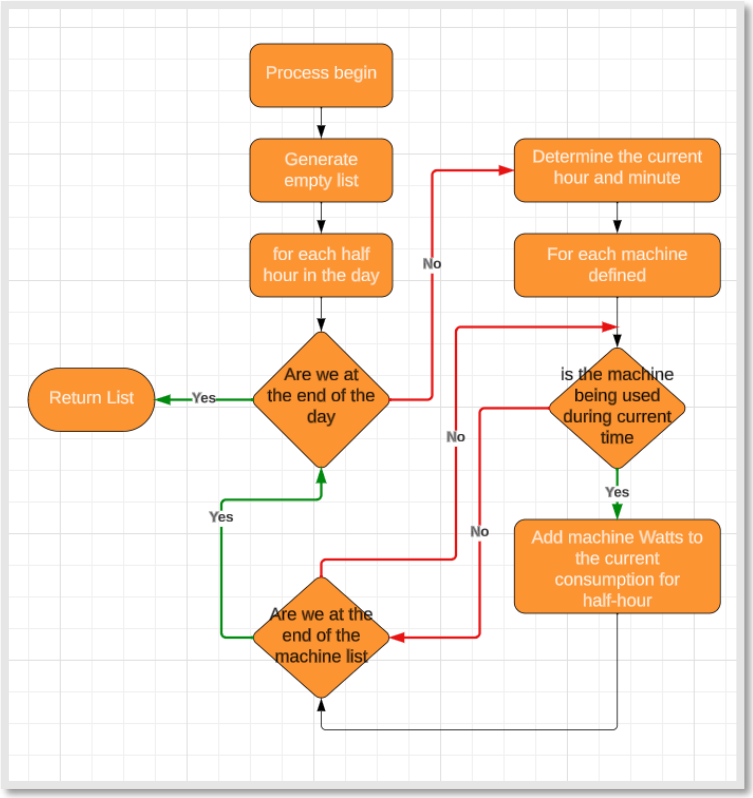


Figure 31 - Factory load based on machines flowchart

4.6.1.2 Static load profile

If the users desire, they may upload a static load profile, which is a single curve of 48 points to represent the factory's load profile through the day. If a static load profile is used, the factory load will be the same at any time of year, based solely on the input data. Lines 18 and 19 of Figure 29 represent this static load. An associated Boolean is also initialized to ensure the class knows when to use this stored load, or a different kind of load.

4.6.1.3 Comprehensive load profile

MicroSim Dev also offers users the capability to upload their own comprehensive load data, as represented by lines 23 to 26 in Figure 29. The load is structured as a dictionary, with the key denoting the date for the load curve, and the corresponding value being a list of data points constituting the profile. Unlike HouseModels, the data is not initially read in; instead, a reference to the location of the data is stored (Figure 29: *compLoadPath*). This approach is designed to conserve resources, as potentially, an entire year or more of data could be read in all at once. When the application requests data from the load, it first checks the dictionary to determine if it has been read in. If the data is found in the dictionary, it is retrieved; if not, the application attempts to read it from the location specified by the user.

4.6.2 Factory Configuration UI

Machine Configuration and load association are streamlined processes within MicroSim Dev, facilitated by the dedicated "Factory Config" form, shown in Figure 32. This form serves as a central hub for users to establish machine parameters, define time ranges during which these machines operate, and consequently determine load profiles. Within this interface, users have the flexibility to configure both static and comprehensive loads, tailoring the usage of the factory within the microgrid to their own requirements.

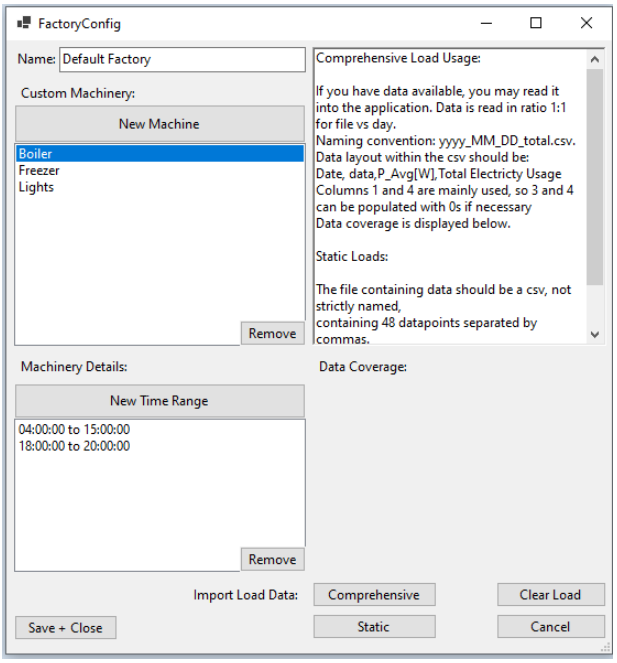


Figure 32 - Factory Configuration Form

As discussed, the incorporation of comprehensive loads is a noteworthy feature of MicroSim Dev. When users import comprehensive load data, the application provides a visual representation of data coverage through informative bar graphs. These graphs break down the data into monthly segments, allowing users to assess the comprehensiveness and distribution of their load data over different times of the year. This graphical representation aids users in gaining a clear insight into how much data they can work with within the simulation.

4.7 Power Storage

Power storage in MicroSim Dev, as it stands, is designed to accommodate a single implementation of the abstract class, specifically, the Lithium-Ion Battery. While there was contemplation about incorporating Lead Acid batteries during the later stages of development, the decision leaned in favour of solely Lithium-Ion Batteries. This choice was influenced by the growing prominence of Lithium-Ion Batteries in the current sector. Their lightweight and improved efficiency made them the preferred option for optimizing the available time in this research project. Looking ahead, MicroSim Dev lays the foundation for potential inclusion of other power storage methods, building upon the groundwork established by the abstract class.

4.7.1 Lithium-Ion Battery class definition

The class definition for Lithium-Ion Batteries (LIB), extending the abstract class EnergyStorageUnit, is intentionally kept concise. This approach is chosen to maintain the application's simplicity and to preserve the intrinsic behaviour of the energy storage units. While the specifications for various battery types often encompass multiple parameters, MicroSim Dev's LIB units rely primarily on the calculated charge rate and capacity for managing charging and discharging. Properties like current and voltage can be stored, and it is important to note that all properties can be customized for potential future enhancements, however in the current state of MicroSim Dev, the maximum charge rate is determined prior to initializing the object, with voltage and current remaining unused within the class definition. As for future development, there could be exploration into dynamically calculating the charge rate of the battery based on its specific definition. Batteries are assumed to be fully charged on creation. Figure 33 provides the code responsible for initializing and constructing the class.

```
11 references
public class LithiumIonBattery : EnergyStorageUnit
{
    // Rated Efficiency
    2 references
    public double Efficiency { get; set; }
    // Store the rated Amps
    0 references
    public int Currnt { get; set; }
    // Store the rated Voltage
    0 references
    public int Voltage { get; set; }
    // String description (typically model number)
    2 references
    public string Name { get; set; }
    // The max charge rate (for per half-hour), which is calculated ouside the class def, to make internal calculations simpler.
    2 references
    public double MaxChargeRate { get; set; }

    // Constructor
    4 references
    public LithiumIonBattery(double capacity, double maxChargeRate, string name, decimal price)
    {
        Capacity = capacity;
        ChargeLevel = capacity;
        MaxChargeRate = maxChargeRate;
        Name = name;
        Price = price;
        Efficiency = 0.95;
    }
```

Figure 33 - LithiumIonBattery class definition

4.7.2 User interface for adding Lithium-Ion Batteries

Users are presented the following parameters for defining the specs of the battery to add to the microgrid. They are laid out in Figure 34 and broken down; each behaves as such:

- **Type**: a string field for distinguishing the make.
- **Capacity**: a numeric-character only field, to specify power storage, in W.
- **Voltage**: a numeric-character only field, to specify voltage.
- **Max C/D Current**: a numeric-character only field, to specify Amps.
- **Life Cycle (Months)**: a field for recording the estimated lifespan of the battery.
- **Price**: a field for specifying battery price (which is utilized later in cost-analysis).
- **# of:** a field for specifying quantity.
- **Charging Time**: a calculated field, displaying the expected charging time of the battery, in hours. This is a calculation based on voltage, amps, and capacity.
- **kWh rating**: a calculated field, displaying the expected kWh rating of the battery specs. This is used for *MaxChargeRate* in the LIB class definition.

The equation $P = VI$ is employed to calculate the estimated wattage intake over one hour, and subsequently, charging time (in hours) is determined by dividing the capacity of the battery by the rated watt-hours. The watt-hour value is divided in half to indicate power consumption over a half-hour during the battery initialisation procedure, and it is then entered into the construction method. Figure 34 is also a 2-frame demonstration on how this form is used, and how the computed fields update dynamically dependent on user customization. Computing this during usage allows users to assess the expected behaviour of their Lithium-Ion Battery while setting it up.

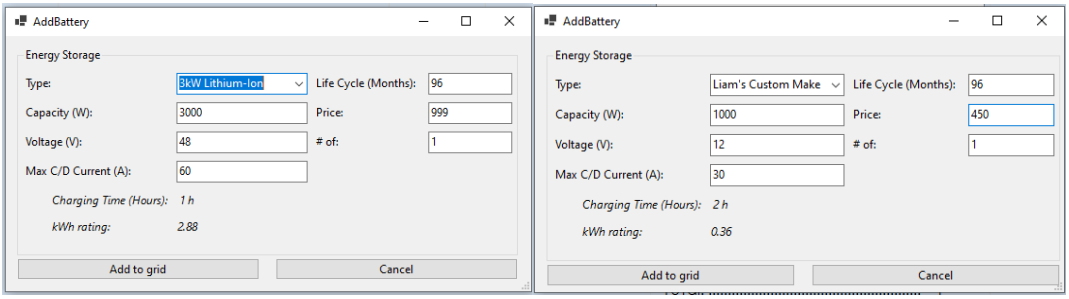


Figure 34 - Adding preset batteries, and customizing battery definitions

4.7.3 Charging a battery

When the user wants to add charge to a battery, it is assumed that this is done over a half-hour period, so the max charge rate metric can be used to determine the capacity change of the battery. Figure 35 handles charging the current battery:

In the 'charge' method, variables are initialized to store the old charge level and the energy not yet charged. The function then checks if the battery is attempting to charge beyond the theoretical maximum achievable within the half-hour window. If it exceeds this limit, the charge amount is reduced to what can be realistically charged in half an hour, and the energy not charged is updated with the remaining amount. Subsequently, the battery's charge level is increased by as much as possible from the available amount. If it's expected to be overcharged, it is once again capped at the battery's capacity, and the calculated amount that was charged is returned. Any remaining uncharged energy is also returned to the function caller.

```
/// <summary>
/// Charge the battery
/// </summary>
/// <param name="amount">Amount in watts</param>
/// <returns>Energy unalbe to be charged</returns>
2 references
public override double Charge(double amount)
{
    double not_charged = 0;
    double old_charge = ChargeLevel;

    if(amount > MaxChargeRate)
    {
        not_charged += amount - MaxChargeRate;
        amount = MaxChargeRate;
    }

    ChargeLevel = Math.Min(ChargeLevel + amount, Capacity);
    double difference = ChargeLevel - old_charge;
    if (difference != amount)
    {
        not_charged = Math.Abs(difference-amount);
    }
    //return not charged energy
    return not_charged;
}
```

Figure 35 - Algorithm for charging a battery

4.7.4 Discharging a battery

The same assumption as earlier applies to discharging, that it is happening over a half hour period. The following algorithm is used to discharge the battery and retrieve power. A simpler technique is employed, as shown in Figure 36, the return value is the energy that is successfully discharged. A check happens at first to ensure the amount is not greater than the *MaxChargeRate* of the battery. Subsequently, the charge level of the battery is decreased by the desired amount (with battery efficiency accounted for), and the amount of energy discharged is calculated and returned to the user.

```
/// <summary>
/// Discharges a battry
/// </summary>
/// <param name="amount">Returns energy that was discharged</param>
2 references
public override double Discharge(double amount)
{
    if(amount > MaxChargeRate) amount = MaxChargeRate;
    double effectiveReturn = amount * Efficiency;
    double old_charge = ChargeLevel;
    ChargeLevel = Math.Max(ChargeLevel - effectiveReturn, 0);
    return old_charge - ChargeLevel;
}
```

Figure 36 - Algorithm for discharging a battery

4.8 Load Comparison

The load comparison tool is one of the tools included in MicroSim Dev for analysing microgrid load profiles at a high level. This application offers a basic analysis of these loads and lets users compare two load profiles. In addition to choosing a generator and a consumer for analysis, users have access to extra features including the ability to total arrays of items and the ability to overlay data on current wind speed and daylight, which can be helpful when adding many generators of the same type to the microgrid.

This feature of MicroSim Dev's tool helps users quickly see how changes to various parameters impact the microgrid. Understanding the causes and effects of various factors is where it is most helpful. Users may quickly see, for instance, how changes in sunrise and sunset hours affect the load profiles. The tool demonstrates how these environmental parameters connect to energy generation and consumption by allowing the switching between overlay options like sunshine and the speed of the wind. This enables users to swiftly analyse and acquire insights into how various microgrid design factors, such as weather conditions, affect energy output and consumption. The tool is depicted in Figure 37 showing how two loads for a particular day can be viewed.

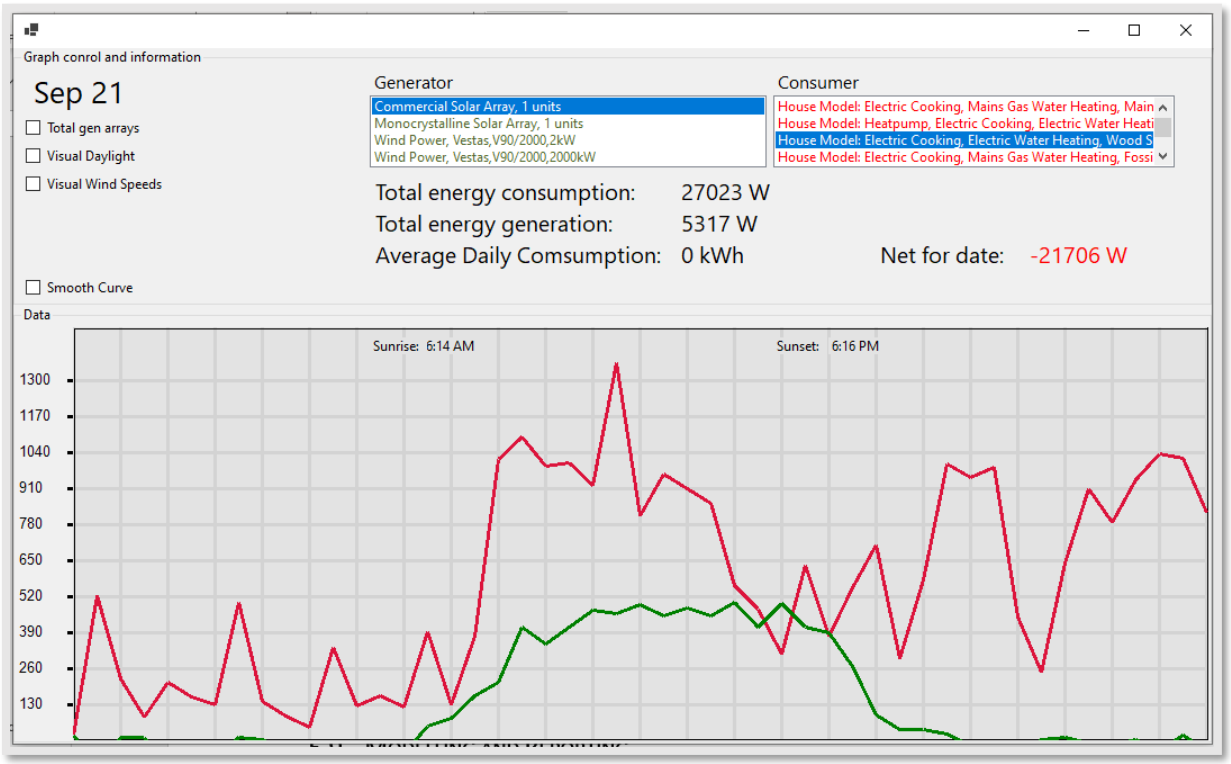


Figure 37 - Load comparison form demonstration

As the tool is used and parameters change, so do the resultant loads. Figure 38 shows how when the day is changed to later in summer, and the amount of sunlight during the day increases, the panel being viewed performs better. Note that a different day results in a different load for that house, but it is instantly seen to have a better performance with this change.

In terms of logic and algorithms, this form is mainly a graphical tool. It does little in the way of calculations, and merely gathers available data and displays it. The main graphing component, custom built for MicroSim Dev, takes a list of 48 values, and plots it along the main axis. The main calculations being undertaken is the totalling of the two loads, and calculation of the NET generation for the day. This way users can quickly see whether there is a surplus, or deficit of generation, regardless of distribution. Variables are used to easily gain the totals, and then displayed.

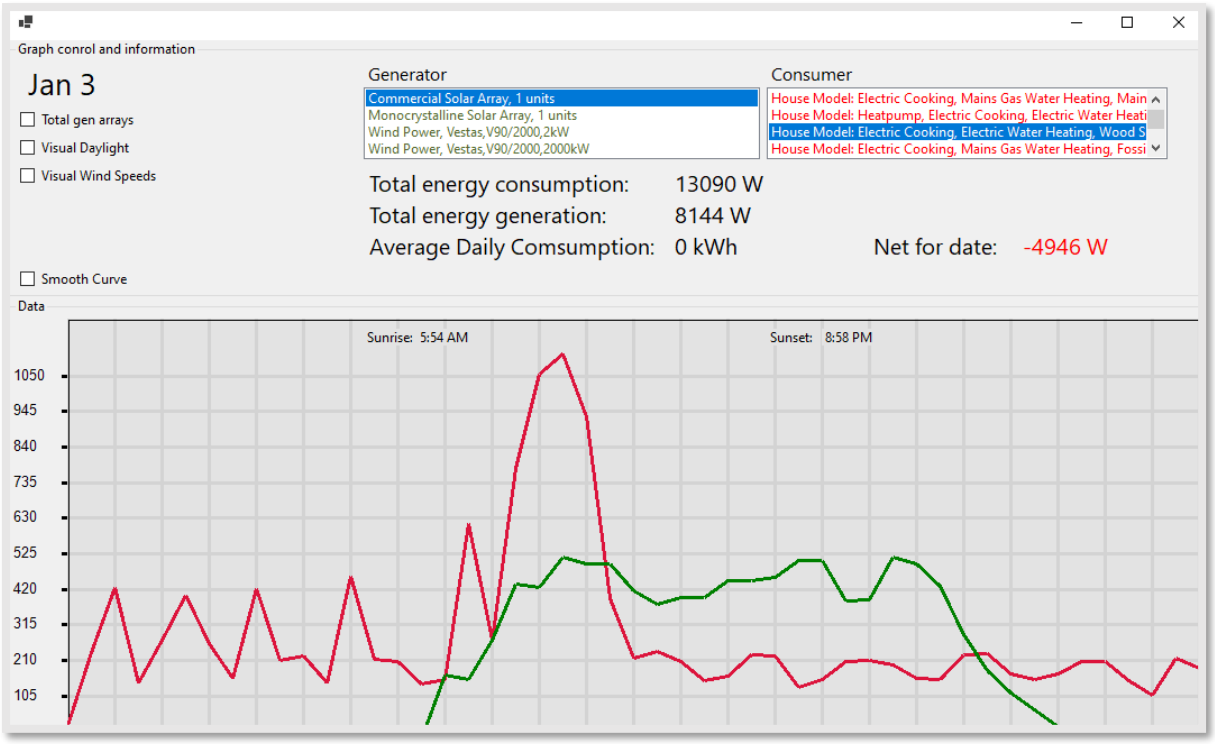


Figure 38 - Load comparison form post date change to summer[1]

4.9 Modelling and Reporting

MicroSim Dev employs a trio of powerful tools to model microgrid behaviour over fixed periods. These tools closely integrate with a dedicated static reporting class, specialized in generating comprehensive performance reports. These reports encapsulate a wide array of crucial microgrid aspects, offering insights into power storage, emissions, grid reliance (if connected), generation, and consumption. The application also boasts an assortment of methods for calculating cost analysis factors, ensuring a holistic evaluation.

For extended timeframes exceeding a single day, a collection of these insightful reports is systematically stored and aggregated. This facilitates in-depth analysis and allows users to grasp the broader trends and patterns within the microgrid's performance. Complementing the detailed reports, MicroSim Dev provides graphical representations, such as energy balance graphs, which offer an additional layer of visual data for enhanced insights and decision-making. This comprehensive suite of tools and reports empowers users to gain a deep understanding of their microgrid's behaviour and make informed adjustments for optimal performance.

4.9.1 Report architecture

At the core of reporting within MicroSim Dev, the relationship between the ReportViewer, the Report Calculator, and the reports themselves is important to understand. The ReportViewer relies on the DailyReporter to generate and provide access to a DayReport. This DayReport acts as a comprehensive container for diverse data and methods. Within the DayReport, a range of 'get' methods is available, allowing users to access specific information about the data and retrieve particular curves as needed. Notably, the DayReport employs a data structure consisting of a List of Tuples, with each Tuple

[1] Changes to the form in *Figure 38* post overview are as follows: Axis have received labels, *AverageDailyConsumption* label removed due to being unused in this iteration, and units changed from 'W' to 'Wh' to correctly represent time metric.

containing a string and a list of integers. This structure essentially functions as a list of named lists, enabling the storage of different power timeseries, each with an appropriate name assigned by the DailyReporter. Figure 39 is a UML representation of these objects, and their relationships to one another.

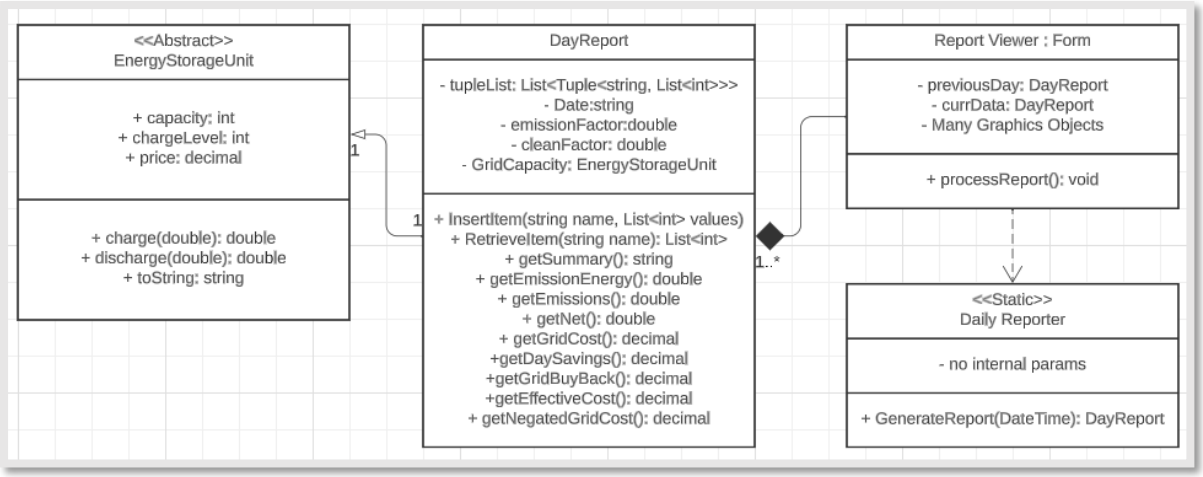


Figure 39 - UML for reporting

4.9.2 Reporting algorithm

The culmination of all developed components thus far hangs upon this single class. Its importance is unprecedented. Without a sensible algorithm for computing interactions and performance of the assets on the microgrid, design analysis and exploration cannot be effectively explored, which is the crux of this project. Define the microgrid, and then explore its design, with respect to asset changes. The algorithm which undertakes the calculations was carefully crafted to ensure the insights provided are as accurate as possible, to allow for the most realistic decision making to happen. The algorithm which calculates the performance is a sequential process which calculates curves and performance from scratch, based on the current day of the simulation. Figure 40 shows a visual representation of this algorithm at a high level, with each step being subsequently broken down in the following section.

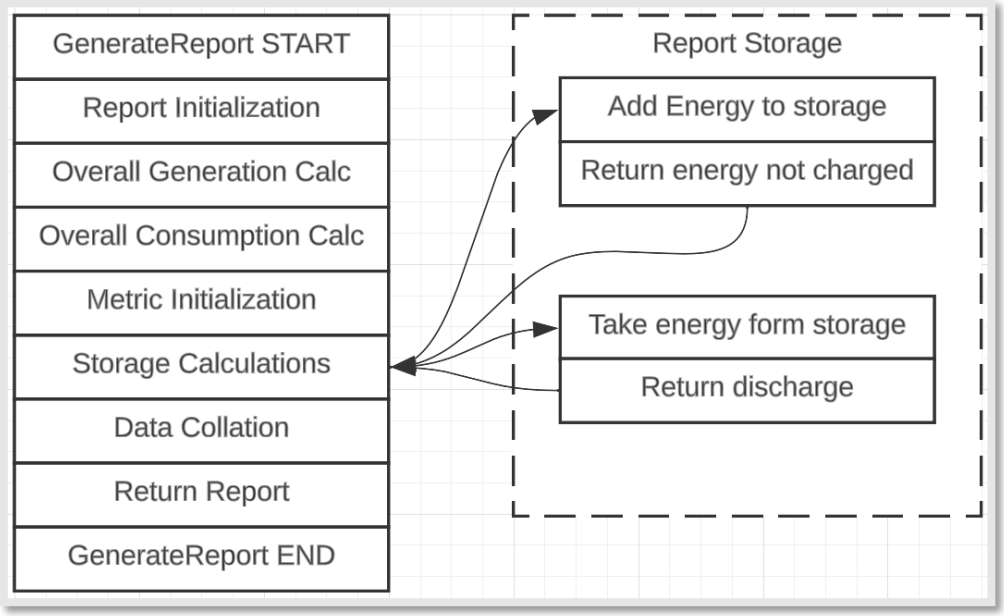


Figure 40 - Process diagram for reporting

The algorithm breakdown begins with the initialization phase (Figure 41). This phase covers initializing variables which are used for tallying, totalling, and returning the data once everything is complete. The final deliverable, which is a DayReport object is initialized with the date here, so calculated metrics can be added to it throughout the reporting process.

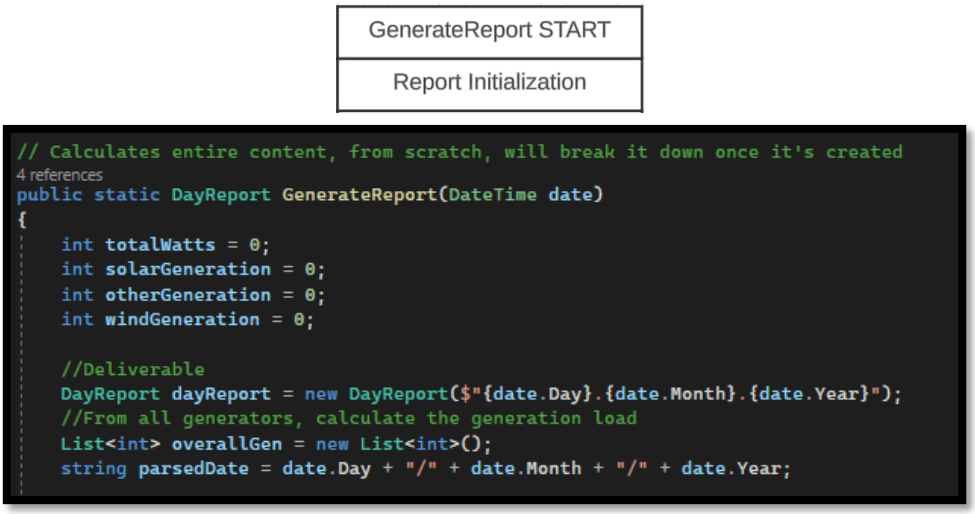


Figure 41 - Report initialization

Following the initialization process and the preparation of data structures, the algorithm proceeds to calculate the total energy generated over the specified time period (Figure 42). To represent this data, a time-series is calculated. The calculation occurs in half-hour increments as the algorithm iterates through the day. For each EnergyIn object, the algorithm retrieves the current generation for that specific point in time and adds it to the running total for the current half-hour interval. Furthermore, separate tallies are maintained, allowing the algorithm to determine contribution percentages for solar and wind sources. This approach provides a detailed breakdown of energy generation, enabling users to gain insights into the balance between solar and wind power during the analysed period. The logic which does the majority of the heavy lifting is shown below. Thanks to the use of abstract classes and polymorphism, this is a concise process. Once the collation of this data is completed, it is added to the DayReport with this statement: *dayReport.InsertItem("Overall Generation", overallGen).*

Figure 42 - Overall generation calculation

The next step in the report generation process involves calculating consumption (Figure 43), which requires a slightly different approach due to the way in which consumption data is stored within MicroSim Dev. Since there is a singular factory within the application, its consumption data is retrieved for the current day initially. This data is converted into integers and stored for later use.

The algorithm then efficiently collects all load profiles for the current day from each house. These profiles are merged into a single list, with each half-hour segment summed to calculate the estimated consumption timeseries for the entire day. Subsequently, this resulting list is merged with the factory data to create a comprehensive time-series representation of all available consumer assets. To ensure the calculations are only performed when the microgrid is configured with HouseModels, appropriate logic is in place. The code which handles all of this is displayed beneath “Overall Consumption Calc”.

Overall Consumption Calc

```
//from all consumers, calculate the consumption load
List<int> overallConsumption = new List<int>();

List<double> factoryLoad = null;
//factory load added separately.
if (Cache.mainFactory.UsesCompLoad)
{
    factoryLoad = Cache.mainFactory.GetHourlyConsumptionList(date);

}
if (factoryLoad == null) factoryLoad = Cache.mainFactory.GetHourlyConsumptionList();

List<int> factoryInts = factoryLoad.ConvertAll(d => (int)d);

List<List<int>> lists = new List<List<int>>();
foreach (EnergyOut consumer in Cache.genListOut)
{
    if (consumer is HouseModel)
    {
        HouseModel house = (HouseModel)consumer;
        List<int> dailyData = house.getDailyData(parsedDate);
        if (dailyData != null) { lists.Add(dailyData); }
    }
}

if (lists.Count > 0)
{
    List<int> resultList = lists.Aggregate((acc, curr) => acc.Zip(curr, (a, b) => a + b).ToList());
    overallConsumption = resultList.Zip(factoryInts, (a, b) => a + b).ToList();
}
else overallConsumption = factoryInts;

dayReport.InsertItem("Overall Consumption", overallConsumption);
//Daily Consumption done.
```

Figure 43 - Overall consumption calculation

Once generation and consumption are calculated, the algorithm can move onto estimating how the storage is utilized throughout this period, given the storage capabilities and setup of the grid. The following breakdown is of the section highlighted in red. To save on processing time, and utilize all available storage effectively within the grid, a single battery object is created (Figure 44) which represents all storage on the microgrid. Note: with this approach, it is assumed in the real world an efficient storage management system is setup, with converters and management systems appropriately researched and built. Unfortunately, given time restraints and scope, this avenue could not be properly implemented in MicroSim Dev’s first iteration of development. The microgrid storage is initialized simply through tallying all the energy storage units’ values and constructing a battery for the report based on these values. The battery coins the name “Report Battery” as it is not technically a member of the official assets but used as a calculative tool. If other types of energy storage were implemented, this approach would be much different, as the conversion and efficiencies between battery types would not compare so simply.

```
double totalCapacity = 0;
double totalChargeRate = 0;
decimal totalCost = 0;
foreach (EnergyStorageUnit esu in Cache.energyStorageUnits)
{
    if (esu is LithiumIonBattery)
    {
        LithiumIonBattery lib = (LithiumIonBattery)esu;
        totalCapacity += lib.Capacity;
        totalChargeRate += lib.MaxChargeRate;
        totalCost += lib.Price;
    }
}
//Big ol battry :D
LithiumIonBattery reportBattry = new LithiumIonBattery(totalCapacity, totalChargeRate, "Report Battery", totalCost);
```

Figure 44 - Report battery creation

The metric initialization phase (Figure 45) ends with the setup of four lists, and many variables, which are all utilized in cost analysis, energy balance analysis, and insights into grid demands.

```
//Generate a list of state of charge throughout the day
List<int> storageCharge = new List<int>();
//Generate a list of times when the grid requires power from the grid, and when it gives to the grid
List<int> gridNeeds = new List<int>();
//For energy balance plot
List<int> balanceCon = new List<int>();
List<int> balanceGen = new List<int>();

int currDemand, currSupply, net = 0;
double giveToGrid = 0;
double needFromGrid = 0;
double recievedCharge, currCharge = 0;
```

Figure 45 - Various energy storage lists, and their initialization

The List *storageCharge* is utilized to keep track of the storage capacity throughout the day, and subsequently represent a timeseries for power storage. The following list, *gridNeeds* is used to keep track of when the overall energy balance is a deficit, or surplus. The power storage timeseries then is modelled throughout the day. It is covered by the code in Figure 46, and is further broken down as such:

The algorithm loops through the full day at each 48-hour point, and the current demand and current supply are retrieved from the pre generated lists. Two values are initialized, one to track energy that is generated and taken from energy storage and another to track energy that is consumed and given to energy storage. These variables enable the tracking of energy balance. The current net energy generation for this half hour period is calculated and then a series of logic checks is undertaken based on this net value.

If the net energy generation is negative:

- Surplus energy can be stored in the report battery.
- The report battery's charge method is activated with the positive value of the net energy generation as its parameter.
- Any excess energy that cannot be stored is returned, indicating it will be wasted or “sold” to the grid.

- The amount by which the battery is charged is calculated based on the net value and the estimated energy sent to the grid.
- As the current half-hour net energy is negative, it is assumed that there is no need to draw energy from the grid. Consequently, the value tracking energy consumption and injection into storage is increased by the amount charged.

If the net energy generation is positive:

- An attempt is made to discharge the report battery.
- The report battery's discharge method is invoked with the net value as its parameter.
- The amount received from the report battery is calculated against the net value to determine the power needed from the grid to maintain power supply.
- If the received amount matches the requested amount, the simulation for this half-hour effectively reaches a net zero state.
- If the power storage is empty, the received amount will be zero.
- Consequently, the value tracking the generation and storage usage is incremented by the received charge amount.

If their net value is 0:

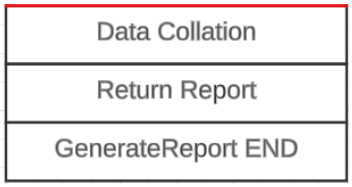
- There is no need to do any retrieval or storage of power.

Consequently, upon the completion of power estimations for the current half-hour, each list is incremented by the appropriate amount(s).

```
//For times of demand when storage is empty, take power from grid instead (therefore adding more carbon emissions, or cost)
for (int i = 0; i < 48; i++)
{
    currDemand = i >= overallConsumption.Count ? 0 : overallConsumption[i];
    currSupply = i >= overallGen.Count ? 0 : overallGen[i];

    //For energy balance plot
    int gen_and_taken = currSupply;
    int consume_and_given = currDemand;

    net = currDemand - currSupply;
    //Console.WriteLine($"Net: {net}");
    //if net is less than zero we have charge!
    if (net < 0)
    {
        //Console.WriteLine($"Charging Battery: {-net}");
        giveToGrid = reportBattry.Charge(-net);
        int chargedAmount = Math.Abs(-net - (int)giveToGrid);
        //Console.WriteLine($"Energy Unable to be charged in half-hour: {giveToGrid}");
        needFromGrid = 0;
        consume_and_given += chargedAmount;
    }
    //if net is more than zero, we deduct charge
    if (net > 0)
    {

        //Console.WriteLine($"Discharging:{net}");
        recievedCharge = reportBattry.Discharge(net);
        needFromGrid = net - recievedCharge;
        //Console.WriteLine($"Energy Recieved:{recievedCharge}, need from grid to makeup: {needFromGrid}");
        giveToGrid = 0;
        gen_and_taken += (int)recievedCharge;
    }
    if (net == 0)
    {
        needFromGrid = 0;
        giveToGrid = 0;
    }
    balanceCon.Add(consume_and_given);
    balanceGen.Add(gen_and_taken);
    currCharge = reportBattry.ChargeLevel;

    //Console.WriteLine($"Current Charge after calculations: {currCharge}");
    gridNeeds.Add((int)(needFromGrid - giveToGrid));
    storageCharge.Add((int)currCharge);
}
```

Figure 46 - Battery charging logic and calculations for daily reporting

Finally, each time-series generated by the report for the current day is added to the report object and returned to the caller, which is shown in Figure 47. With this structure in place, reports of any length can be easily generated by iterating over the days that interest the caller.




```
dayReport.InsertItem("Battery Usage", storageCharge);
dayReport.InsertItem("Overall Grid Needs", gridNeeds);

dayReport.InsertItem("Balanced Generation", balanceGen);
dayReport.InsertItem("Balanced Consumption", balanceCon);

//battery alignment
dayReport.GridCapacity = reportBattry;

return dayReport;
```

Figure 47 - Inserting the lists into the report

4.9.3 Cost Analysis functions

DayReport objects are responsible for conducting all cost analysis calculations for the microgrid on the specific day they represent. In this current version of MicroSim Dev, the average cost of power is fixed to Hamilton's estimated cost per kWh, which is 20.1c. However, users can readily modify this global metric on the main form to maintain their freedom of choice in the simulation. This flexibility ensures that the application can adapt to various locations and pricing models as required.

The report object can return many values which represent various cost-analysis metrics. A list of important cost-analysis metrics is outlined below:

- *getEmmisionEnergy*(): This method calculates the total energy consumption that comes from sources with positive values in the "Overall Grid Needs" item. It iterates through the list and sums up positive values, which represent times from the report where energy was required from the grid to continue operating.
- *getGridCost*(): This method calculates the grid cost based on the emission energy, scaled to kWh, and a power cost from the cache. It returns the result rounded to two decimal places.
- *getGridBuyBack*(): This method calculates the cost of buying back energy from the grid based on negative values in the "Overall Grid Needs" item, a scale factor, and a power cost from the cache. It returns the result rounded to two decimal places.
- *getEffectiveCost*(): This method calculates the effective cost of energy consumption based on the overall consumption, scaled to kilowatt hours, and a power cost from the cache. It returns the result rounded to two decimal places.
- *getNegatedGridCost*(): This method calculates the negated grid cost by subtracting the grid cost (output of *getGridCost*()) from the effective cost (output of *getEffectiveCost*()). It ensures that the result is at least 0 and returns it.
- *getDaySavings*(): This method calculates the daily savings by adding the negated grid cost (output of *getNegatedGridCost*()) and the grid buy-back (output of *getGridBuyBack*()).

4.9.4 Report Data visualization

The Report Viewer Objects in MicroSim Dev serve as essential tools for visually representing the comprehensive data collected and calculated within the DayReport objects. Three distinct tools are available for visualizing these reports, each with unique features. The Monthly Report Viewer is a prime example, offering a comprehensive overview of the application's capabilities.

The Monthly Report Viewer, implemented as a Form object, takes an integer representing the month as input to specify the month of interest. Its constructor manages the initialization and population of the report viewer. It loads data for each day within the selected month, updates the user interface (UI), and handles loading screens during the report generation process. The viewer presents the collected data in a list box, allowing users to selectively display or hide different timeseries aspects. This feature enables users to overlay various data sets, facilitating the exploration of intricate relationships between different time series.

All report viewer classes feature two main UI elements: a timeseries graph and the totals section. The totals are computed during the initialization of the report and are static throughout the user's interaction. In contrast, the timeseries graph is dynamic and updates in real-time as users interact with the form's options and dialogs.

To update the totals displayed on the graph, the form iterates through each DayReport object for the specified month. It collects and sums the totals for each metric, then presents these cumulative figures on the UI. Additionally, the form calculates the estimated payback period for generative objects and

batteries, providing users with valuable insights into the financial aspects of their microgrid configuration. Figure 48 shows the totals section in use, with some example values.

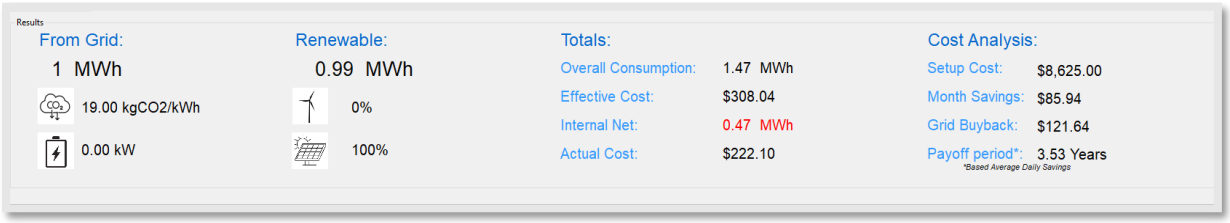


Figure 48 - Month report viewer's result totals

The task of updating the form's graph is delegated to a distinct process designed for this purpose. This process plays a pivotal role in redrawing the graph on the canvas, which encompasses various components such as grid lines, labels, and data points. It relies on the data stored in the *monthlyReport* and the user's selection of time-series items from the available list. This ensures that the graph is accurately scaled and visually represents the data for comprehensive analysis.

To optimize this process and avoid redundant coding, the same custom-built graphing techniques used in the load comparison tool are applied here. By reusing these techniques, the application minimizes the need for additional graphic functions and ensures a consistent user experience. Furthermore, the process assigns a unique colour to each time-series, and these colours are showcased in a legend displayed in the top-right corner of the canvas. This legend aids users in identifying and distinguishing between the various data sets presented on the graph. Figure 49 depicts an example state of this graph, with consumption, generation, and battery time-series all toggled for drawing.

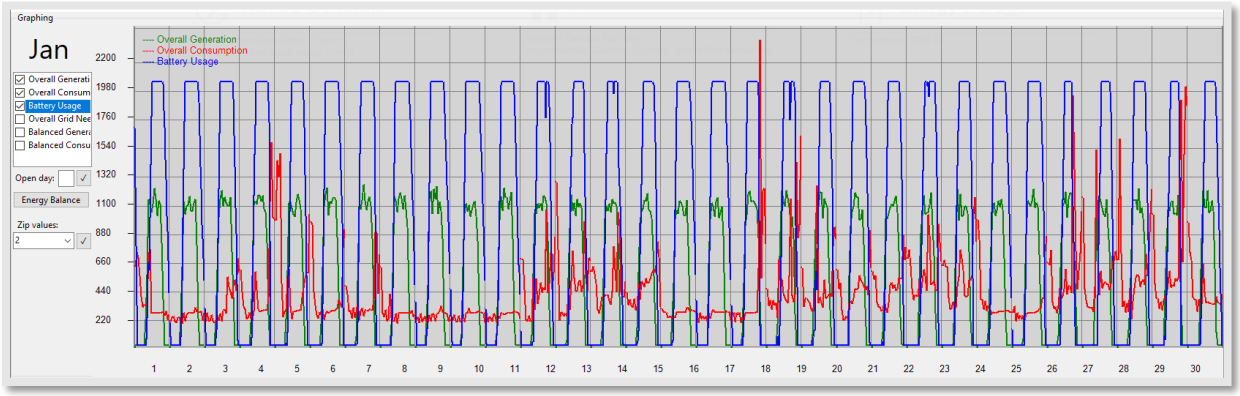


Figure 49 - Month Report viewer's graphing capabilities

4.9.5 Energy Balance Plots

Energy Balance Plots stand as a pivotal feature within MicroSim Dev, facilitating in-depth analysis and exploration of the microgrid setup. These plots serve as valuable visual representations, illustrating how the microgrid manages the distribution of renewable energy generation. By incorporating these plots, MicroSim Dev provides users with an additional layer of insight, allowing them to instantly observe the impact of their adjustments on the system's performance. This feature significantly enhances the depth of analysis and empowers users to make informed decisions regarding their microgrid configuration. These particular balance plots represented are designed as a result of the work by Apperley (2017).

Their incorporation is achieved through the usage of the OxyPlot graphing library, and utilizing information that is calculated during the reporting algorithm. During the reporting algorithm, two lists are initialized which are used to track two metrics:

- Total generated energy, and energy that is retrieved from storage, over the time interval.
- Total consumed energy, including energy that is delivered to storage.

The use of these metrics enables us to monitor when the energy generation and consumption align over the specified time interval, while considering energy storage. This means that any excess energy generated, which is not immediately consumed, can be stored, and used to balance the system at a later time. This balance is crucial for maintaining a stable and efficient microgrid.

The balance plots are essentially form objects that take two lists as input. It is assumed that these lists are of the same length, with corresponding points in each list representing the same time interval. These corresponding points are treated as X and Y values, creating a dot plot that visually illustrates the energy balance. Figure 50 provides two example outputs of this graph: one depicting a system over a month without balance and another showing a system with some degree of balance.

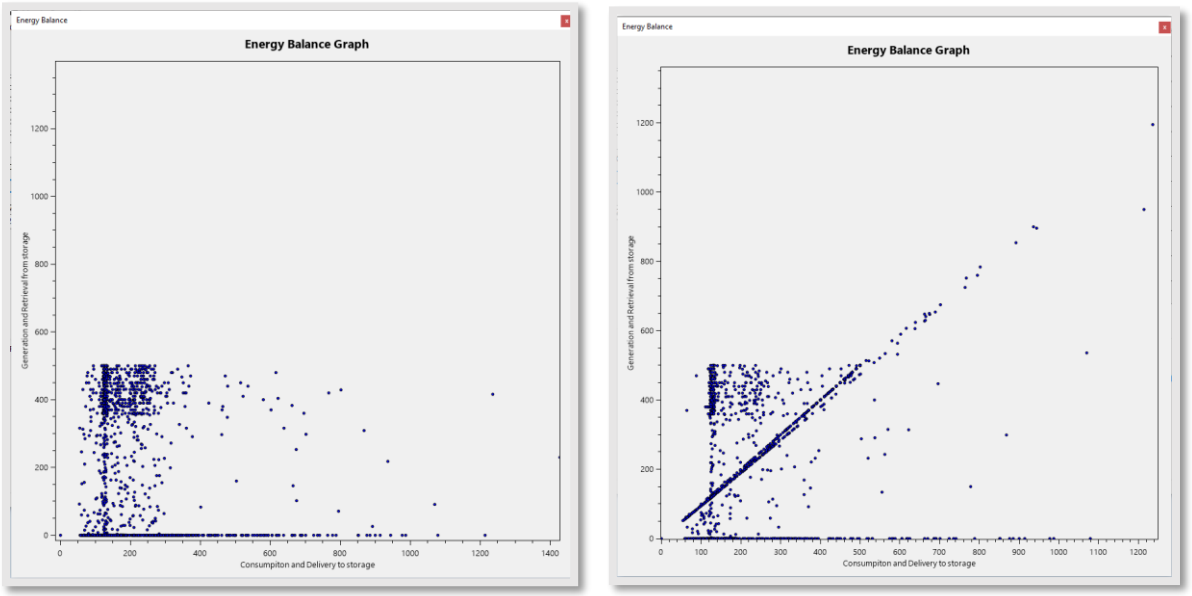


Figure 50 - Energy balance - left: no balance, right: some balance

This ends the strategy used to develop MicroSim Dev in its initial prototype form. The user-friendly GUI and straightforward functionality of the application provide for a free environment for the investigation of microgrid design. There are a lot more details, components, and supporting structures that go into making MicroSim Dev function, but listing them all would take up much more space than is required. The efficiency of MicroSim Dev and its facilitation of exploration are assessed in the part that follows.

5 EVALUATION

The evaluation of MicroSim Dev's effectiveness and the insights it offers will be demonstrated through a series of case studies. These case studies will depict three scenarios that illustrate the diverse applications of MicroSim Dev, spanning from detailed, small-scale exploration to comprehensive assessments of larger microgrids. To maintain realism, the case studies will rely on real-world data, with minimal adjustments except for some load-shifting aspects in the case of custom-defined loads for factories. These case studies will effectively showcase the practicality and versatility of MicroSim Dev. Through these real-world scenarios, it will become evident how this implementation aligns with the original objectives and goals of this project, emphasizing its value in the field of microgrid design and analysis.

These case studies will be as follows:

1. Exploring the setup of a single home
2. Exploring the setup of a small community
3. Exploring load shifting with a dynamic factory load
4. Microgrid design exploration

5.1 CASE 1: SINGLE HOME

In this particular case study, the primary objective is to gain a comprehensive understanding of the energy load associated with a specific house. The focus is on investigating the potential consequences and benefits of integrating solar energy generation and a limited energy storage system into the house's existing setup. Essentially, this house is considered to operate in an "off-grid" mode, meaning it functions independently or as a small-scale microgrid. The desired outcome of this scenario is to minimize or ideally eliminate the house's reliance on the external grid as much as possible. This implies achieving a situation where the house can sustain its energy needs using the locally generated solar power and the available energy storage capacity. This case study aims to determine how feasible and practical such a setup is and what challenges or advantages it may present. To begin, a house was chosen that represents two people with above average income, living in a rural location (Figure 51).

With this house added to the microgrid as the sole object, it's load can be analysed. To gain a quick insight into the cost of power on this house, the yearly reporter is called for a quick overview of the totals to be worked with. Figure 52 depicts the initial yearly load. The totals tell us that the generative capacity will need to be around 5MW a year. The current cost of power per year is estimated $1,100 if the cost of power is set to 21c per kWh. Since the energy used in this setup is estimated to be retrieved from the grid, there are associated carbon emissions automatically added into the results.

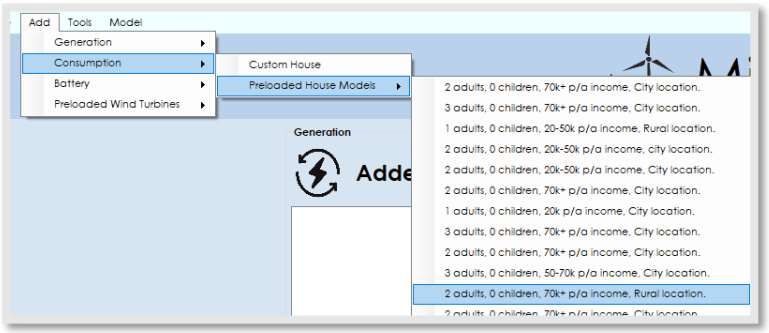


Figure 51 - UI depiction of selecting preset house models

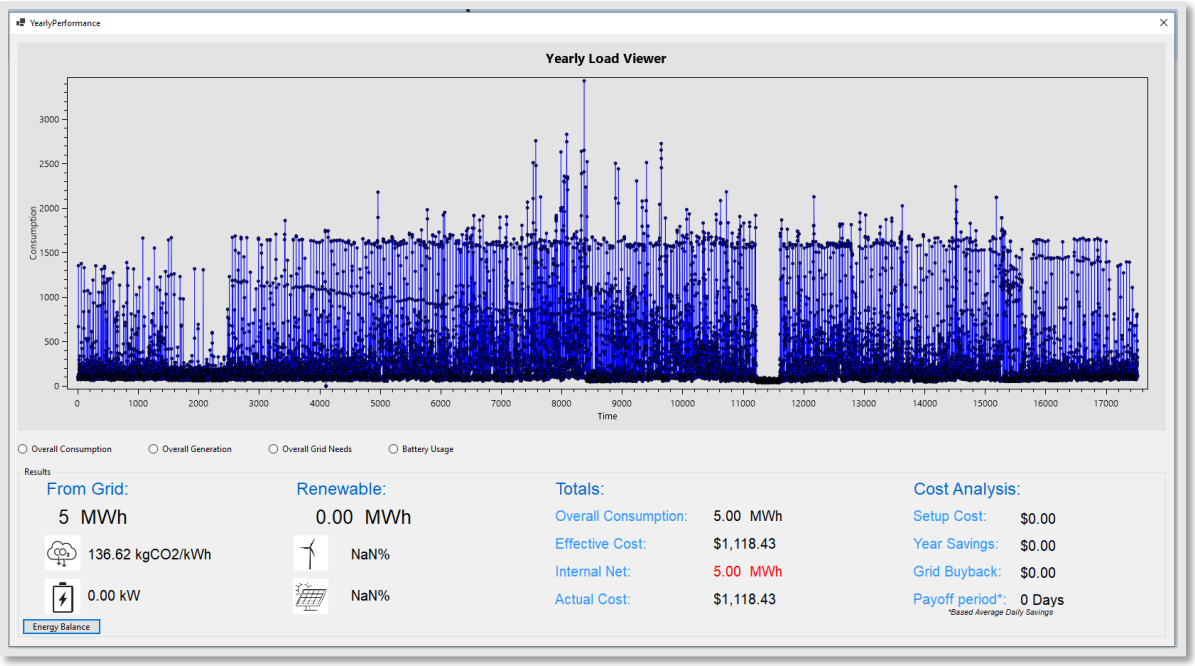


Figure 52 - Initial year report for case study 1

The question can be asked, what differences occur when adding solar panels to the house. To emulate closely how users may interact with MicroSim Dev, the term 'residential solar panel' is searched on the internet, and a solar panel that is listed for sale is chosen at random. Its specifications are entered into the solar panel interface, and five of these panels are added to explore what kind of difference this may make to the setup (Figure 53). The implications of this change can now be explored anywhere within MicroSim Dev, so the yearly report will be viewed again to see these changes reflected in the grid.

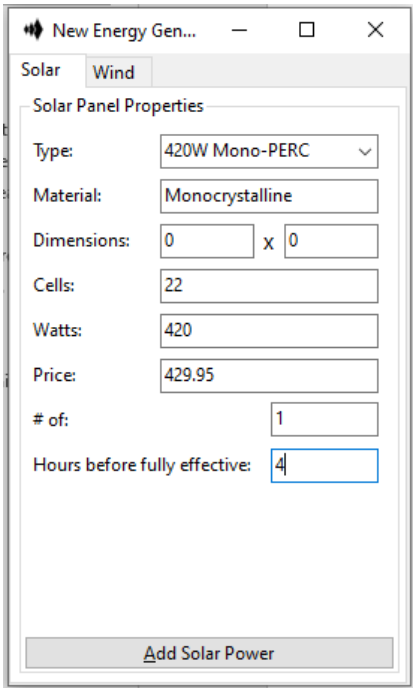


Figure 53 - PV setup for case 1

The changes are instantly reflected in the yearly performance, and it can be deferred that an appropriate number of panels have been installed, to match the yearly consumption. The five installed panels each generate 1MWh of power, totalling up to 5MWh which brings the total net power usage for the year to around the zero mark. The text is represented as red, to signify that there may still be a slight deficit. The system still requires around 4MWh from the grid to remain operational at all times however, which may spark interest into the actual balance of the power in this scenario. As renewable energy is incorporated into the microgrid, more carbon emissions are factored out, which is reflected in the yearly load viewer post adding solar panel potential.

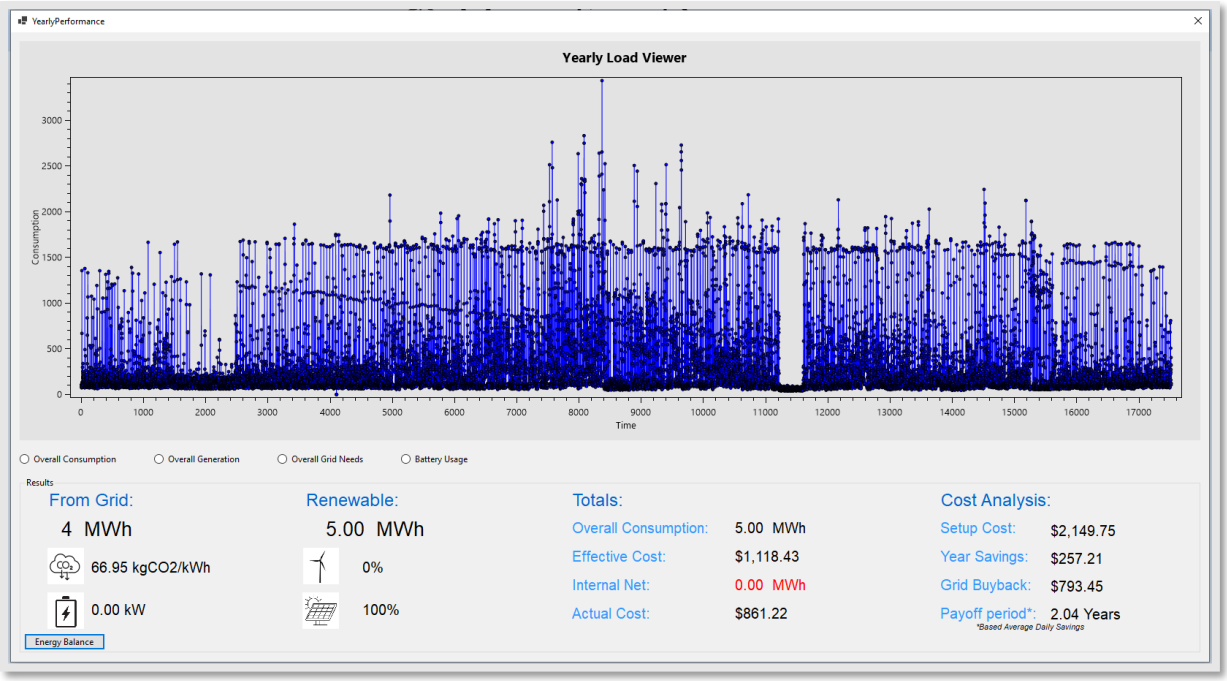


Figure 54 - Year report for case 1 after solar panels added to microgrid, costs being noteworthy

Figure 55 - Month report to show lack of balance between generation and consumption

The current energy balance plot for a year, with the panels included, is depicted in Figure 54. This figure clearly shows the unbalance in the system, despite the renewable generation potential being enough to match the consumption. This is further supported by viewing a month of performance and comparing generation against consumption. Figure 55 shows this comparison, and clearly depicts the out of balance system, represented through high loads (red) at times of low generation, and high generation (green) at times of lower loads. Costs can also be analysed too, with Figure 54 showing us totals of setup, grid buyback, savings and expected payoff period.

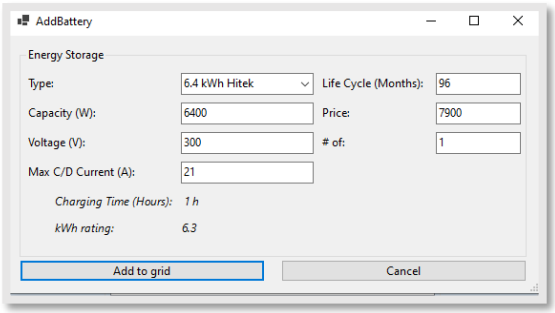


Figure 56 - Battery setup for case 1

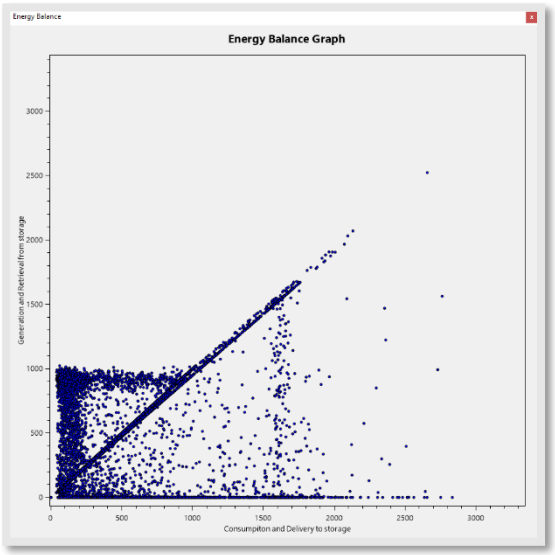


Figure 57 - Energy Balance at case 1's conclusion

Users can utilize this information to gain a vague idea of how long it would take for these investments to become fully effective. Adding storage to the simulation is emulated in the same way as adding solar panels (Figure 56). "Lithium-Ion batteries for home power storage" is searched on the internet, and at random, a product is chosen to be entered into MicroSim Dev. The product chosen is a particularly expensive, but high-performing storage option. Again, changes are instantly shown in the energy balance plot (Figure 57), and it can be seen that, although not perfect, the microgrid is balancing with better performance. The graph still depicts unused generation and therefore the microgrid requires more storage potential.

To conclude this case study, MicroSim Dev's adaptability and ability to include a variety of asset kinds enable users to thoroughly explore smaller setups. The case study shows the potential for achieving off-grid sustainability, even though more development and research into various products is required. This study demonstrates how a small-scale system, such as a single home, may be utilised inside MicroSim Dev to develop sustainable energy solutions and successfully manage renewable generation efforts, particularly during times of decreased generation.

5.2 Case 2: Small Community

In the next scenario, all available house models will be added to the grid, to simulate a small, closed community microgrid of 27 houses. This utilizes all available housing data, to provide a variety of moving parts, which is realistic when dealing with real microgrids. In the Waikato Region, July is one of the coldest and darkest months of the year, resulting in solar power being less effective, and heating being more prominent, which contributes to these household loads. The goal of this case study is to explore renewable energy sources and optimize the month of July in terms of power distribution between households.

First the simulation current date is changed to July 18 (Figure 58), so there is a default centre point for viewing a single day-report or altering weather conditions. A control performance can be obtained simply by requesting MicroSim Dev's monthly report for July. All houses on the grid are included in the report, and the following graphic (Figure 59) is shown:

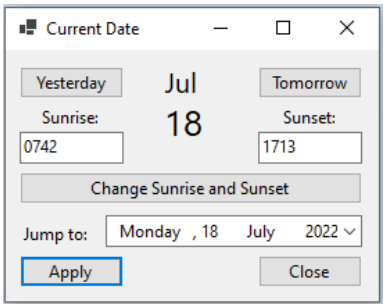


Figure 58 - Date changer tool in usage

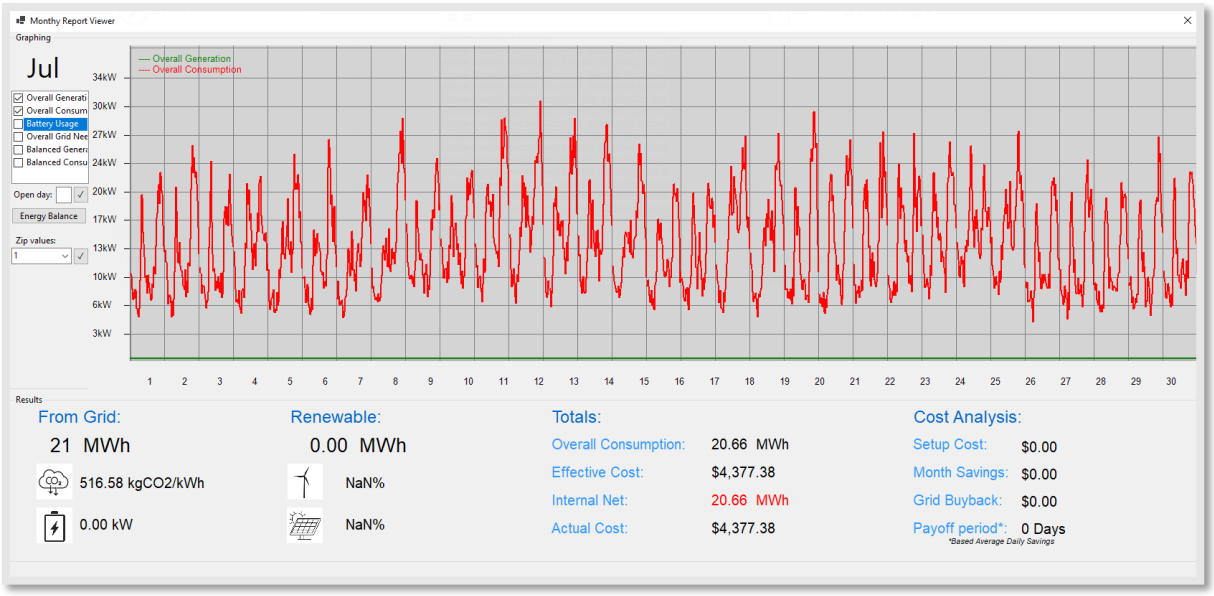


Figure 59 - Initial month of July report for case 2

It is intriguing to note that by examining the visualized data alone, one can discern that most of the energy consumption within this community is concentrated in the evenings before midnight (indicated by vertical grey lines separating each day). This insight suggests that the microgrid configuration could be enhanced through the incorporation of wind energy generation, particularly because the peak consumption periods occur during nighttime hours. Following a similar process as before, a search for "Small-Scale wind generation" is conducted on the internet, and a random product is selected to be implemented in this scenario. Initially, 25 out of the 27 houses will be equipped with one of these wind turbines. Figure 60 shows the immediate changes to the month of July's predicted performance.

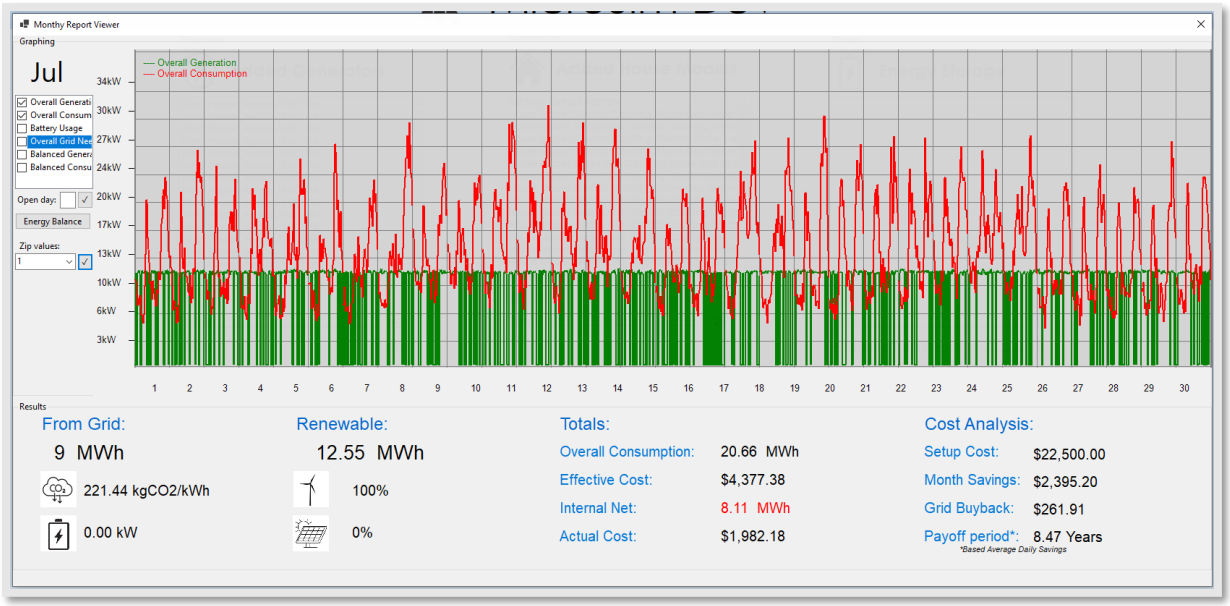


Figure 60 - Month of July report for case 2 with wind power

The report in Figure 60 shows that the turbines added do not cover the load of the community, as the month has an internal deficit of 8.11 MWh, but generally speaking the cost of power has been slashed down thanks to the turbines. The present deficit does not mean this study has failed; however, it

simply means there is more exploration into the configuration to undertake. A reminder that the overarching goal of this project is to facilitate exploration. To supplement the turbines, a simple solar solution will be implemented. Again, 25 preset panels will be added to see how the incorporation of solar generation may change how the outcome of this scenario looks.

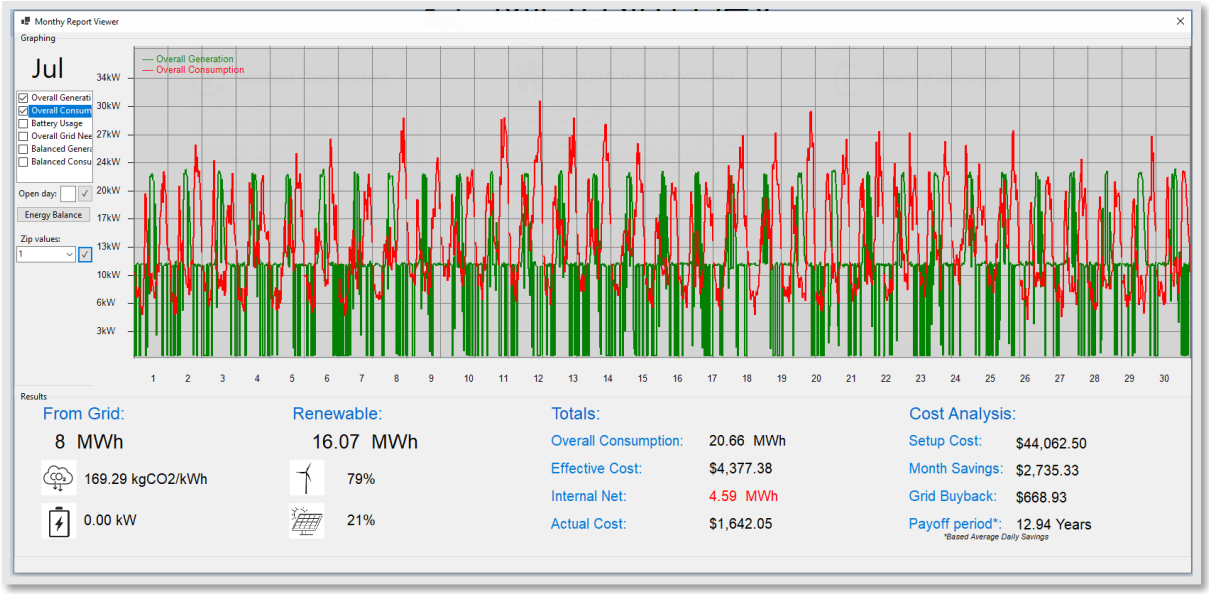


Figure 61 - Month of July report with wind and solar power

With the addition of some generic solar, the amount of generation potential has increased by around 4MWh (shown in Figure 61), however the amount that needed from the main electricity grid has only been decreased by around 1MWh. Again, with the month of July having the least sunlight, this is not as effective as it could be. To fully utilize the generative capabilities of the solar, energy storage will be added to the scenario. Four preset large batteries (20kW Capacity) are added. Figure 62 and Figure 63 respectively show the final isolated grid needs of the microgrid for July, and the energy balance over this time.

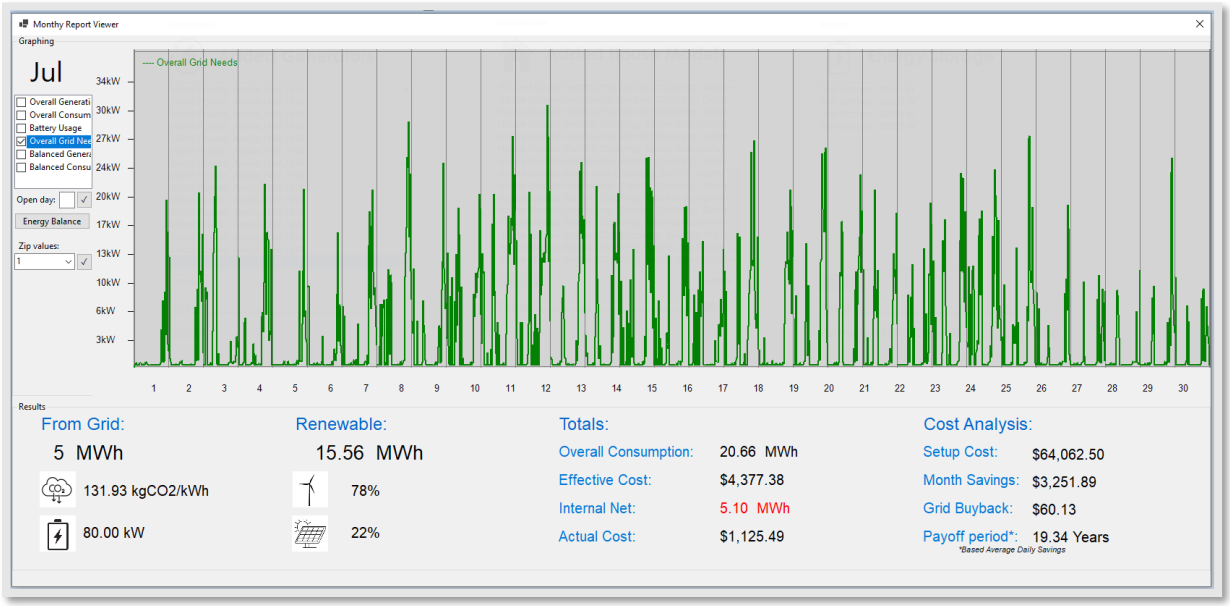


Figure 62 - Month of July report with just grid provisions

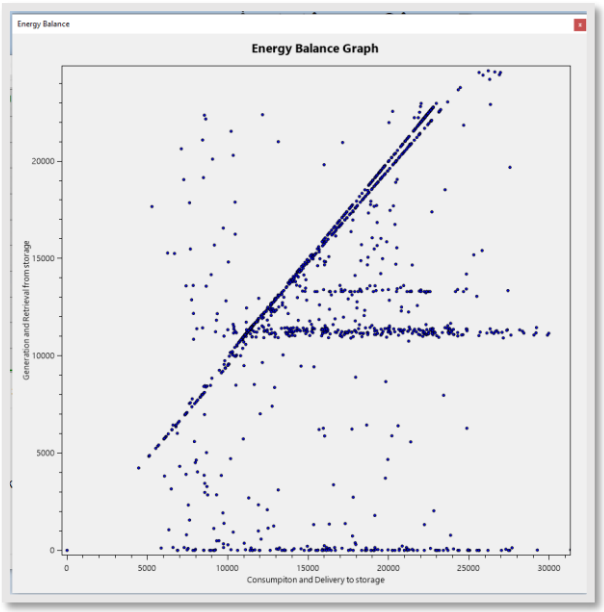


Figure 63 - Energy balance for Figure 62

This case study concludes with the assurance that the majority of generative potential within the microgrid will be efficiently utilized, minimizing wastage, and ensuring stability (although grid buyback stability is not a focus of this study). Through this study's findings, it is also possible to predict that the microgrid stabilised towards the end of July, which appears to have the overall lowest common demand on the grid and has regular pauses in grid demand, possibly as a result of rising sunshine and a declining need for house heating. These are the kinds of insights that mere trends and data do not offer, but visual reporting and the modelling of a full microgrid reveal these complexities.

Decisions made throughout this case study were influenced by various factors presented in the monthly reports. These factors encompassed net energy statistics, grid demand, cost, generation type, and carbon emissions. With each modification to the microgrid, all these factors experienced changes, providing users with a comprehensive breakdown of the system's behaviour. These diverse avenues of exploration further underscore MicroSim Dev's effectiveness in facilitating in-depth analysis of microgrid design.

5.3 Case 3: Dynamic Factory Load

In this case study, the intricacies of a single factory and its daily operations are explored, utilizing MicroSim Dev as a potent tool to scrutinize its specialized load requirements. The study serves as a compelling demonstration of MicroSim Dev's capacity to accommodate and simulate highly specific industrial loads. The following example is entirely fabricated and serves as an imaginative narrative for the uses of this application.

The scenario unfolds within the context of a prestigious dairy product processing and distribution company embarking on the creation of a new dairy factory in the Waikato Region. However, the company is not merely pursuing conventional factory construction; instead, they are deeply committed to integrating renewable energy principles and adopting carbon reduction strategies into the very fabric of their new facility. To gain deeper insights into the viability and implications of these ambitious initiatives, the company chooses to replicate one of their existing factory configurations within MicroSim Dev, thus initiating an in-depth analysis of the potential benefits and challenges associated with their sustainable objectives.

Table 2 depicts the initial configuration for this factory's machines. The load comparison tool, represented in the screenshot below, is then used to quickly see how this load is across one day (Figure 64).

Table 2 - Initial machinery setup for factory

Refrigeration unit, 5000W	24hrs a day
Pasteurization Equipment, 2500W	7am - 11am, 1pm - 5pm
Homogenization Equipment, 1500W	8am – 10am, 2pm – 4pm
Filling and Packaging, 2000W	6am – 10am, 12pm = 4pm
Lighting, 750W	6am – 7pm
Boiler, 7500W	5pm – 10pm

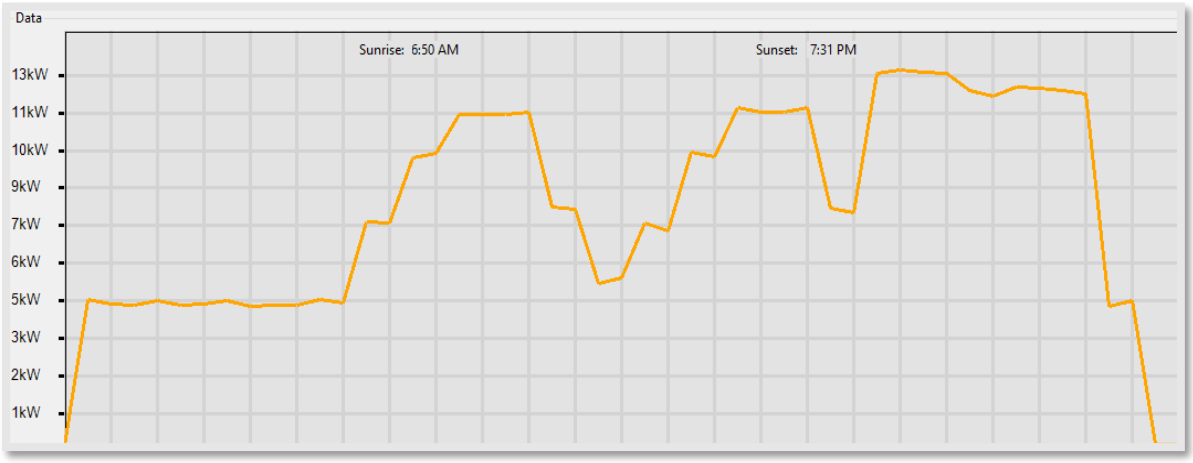


Figure 64 - Load Comparison (Graph only) for viewing factory load profile. Note the 0 values either side of the graph are graphing bugs which have been addressed post study completion

Following a study of the current configuration, the dairy factory owners decide that solar panels will be placed to help offset part of the daytime power consumption. In order to observe the initial change in load brought about by the installation of renewable generation, 30 generic commercial panels offered by the application are initially chosen.

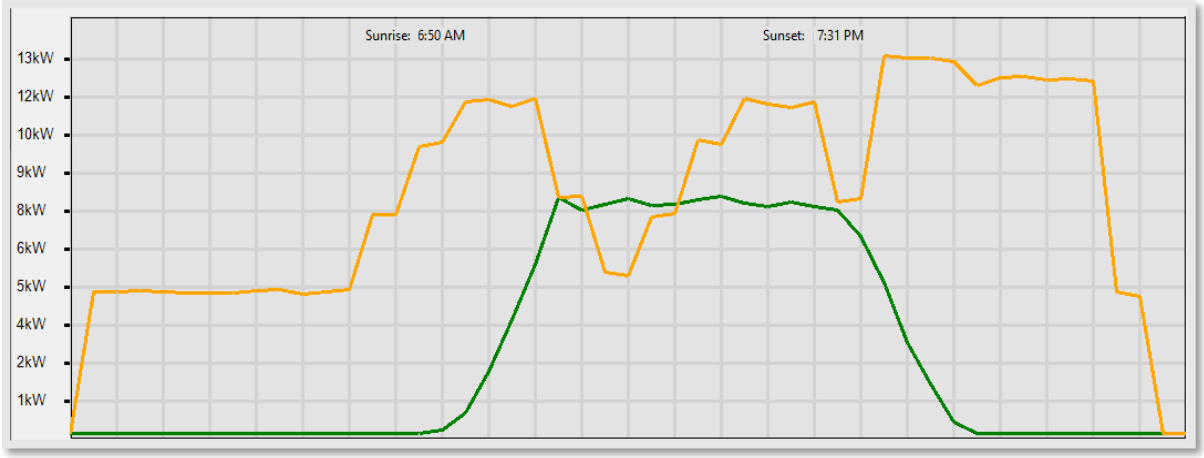


Figure 65 - Load comparison with added solar generation

This offsets the generation during daylight hours by a decent amount (shown in Figure 65), however the factory owners think they can utilize all solar generation through load shifting. Their boiler is currently run during the evenings but have suggested a new workflow that allows the boiler to be run

in-between morning and afternoon shifts. This can be easily changed within MicroSim Dev by deleting the time range for the boiler and re-specifying the new range. The affected load is shown in Figure 66.

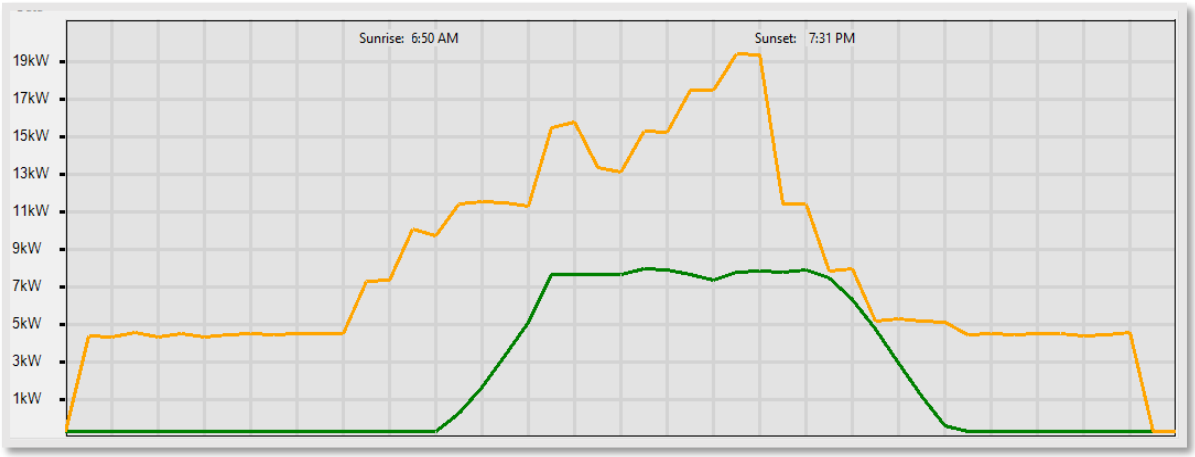


Figure 66 - Load shifting result, seen in the load comparison tool

With effective utilization of load shifting, the solar panels that the factory owners plan to implement can be fully utilized throughout the workday, without wasting as single watt of its generated power. In light of this, the factory owners are also considering the possibility of switching their delivery truck to an electric vehicle in order to save money on gasoline and eliminate a source of carbon emissions from their brand-new facility. This would mean charging the truck after a long day of labour, and the factory arrangement can be modified to include another "machine" to stand in for EV charging.

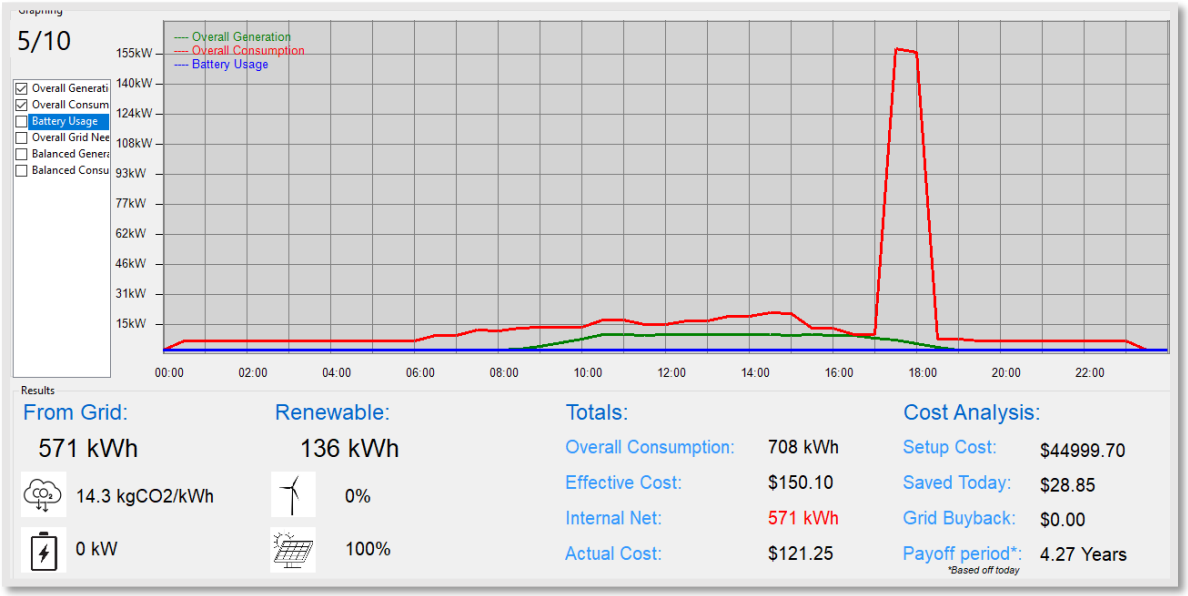


Figure 67 - Daily report with EV charging

Upon implementing the change in MicroSim Dev, and generating a daily report (shown in Figure 67), the factory owners review within their team whether or not utilizing an electric vehicle will be cost effective, and actually cut down carbon emissions, granted the amount of power it takes the charge the truck at the end of the day, and the assumed emissions associated with grid-received electricity. From here, it would be expected that iterations of development and further research take place outside the realm of MicroSim Dev, and if other options need to be explored, the users will promptly return to enter their new findings into the grid setup.

This case study is a testament to MicroSim Dev's capacity to facilitate load shifting and create a versatile sandbox environment that accommodates external assets, such as electric vehicles, seamlessly integrated into the simulation, even when they are not inherently part of the application. Furthermore, it underscores MicroSim Dev's factory objects, allowing users to meticulously define and customize these objects for microgrid modelling. Users are able to make difficult decisions and analyse their situation with as much information as possible thanks to the various tools and reporting options, demonstrating the usefulness of the programme once more.

5.4 Case 4: Microgrid design Exploration

This final case study will delve into the very question asked by this thesis. Can we explore the design of a microgrid, and view the repercussions of our design decisions in a computer application? To answer this question, MicroSim Dev will provide a blank canvas, to be populated into a miniature energy ecosystem where sustainability, efficiency, and exploration intertwine. The microgrid will be composed of all available elements. 54 residential house models, a factory model, wind turbines, solar panels, and the steady heartbeat of energy storage.

Contextually, this case study is assumed to revolve around a coastal community, nestled in a region abundant in wind and sunshine. This community has a factory at its heart, striving to harness renewable energy, reduce emissions, and optimize energy usage. Meanwhile, the residents, each residing in their uniquely designed homes, are enthusiastic participants in the quest for a sustainable future. As the microgrid design is navigated, operated, and optimized, MicroSim Dev will show its ability to explore the delicate balance between generation, consumption and storage, and the intricate interplay of assets on a larger scale. The main goal is to show this application's capabilities and will be done through this study's internal goal of discovering a cost-effective long term sustainable energy solution.

This scenario begins with the importing of 54 house models, and yearly factory data. The factory data will be imported via the factory configuration tool, and its coverage is visually represented (Figure 68). The data being used for modelling a factory has a gap in the months of May, June, and July, as shown by the empty green bars for each of these months. Knowing this before modelling is important, as users can understand better why the application behaves the way it does. Upon adding these objects, the main form of MicroSim Dev shows the them to the user (Figure 69).

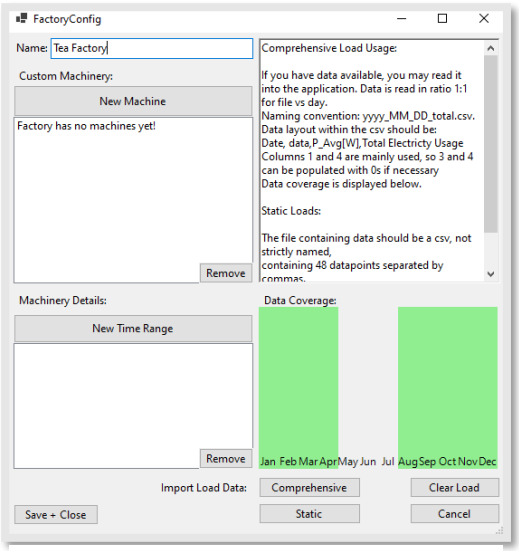


Figure 68 - Factory configuration, with data coverage visuals

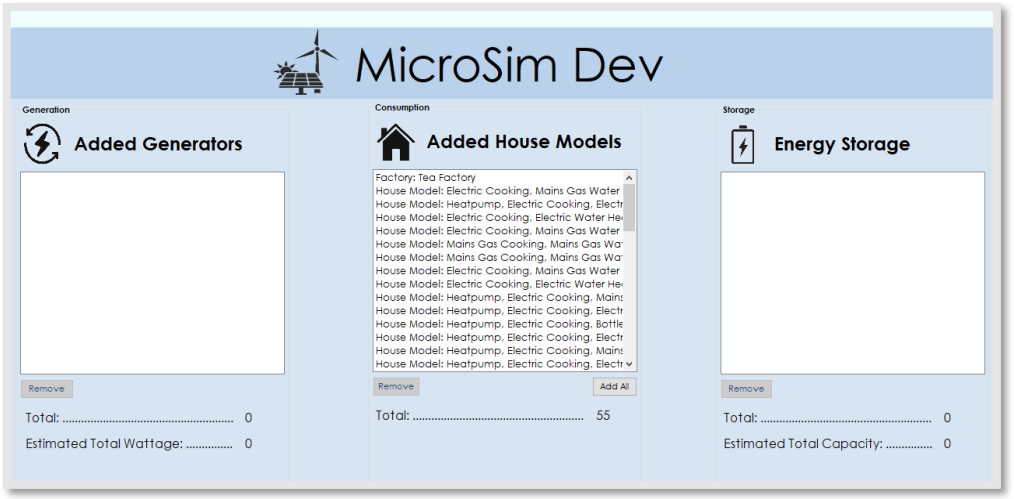


Figure 69 - MicroSim Dev main form with added consumption assets

To obtain approximate statistical insights into the microgrid's performance, an analysis of several random daily reports has been conducted, each corresponding to different times of the year. By utilizing the date tool within the simulation, the current date setting was altered to represent various days of significance: January 11, April 1, July 21, and December 22. Presented below are the consecutive daily reports for each of these selected dates. This approach was employed to capture a diverse range of seasonal conditions and their impact on the microgrid's functioning.

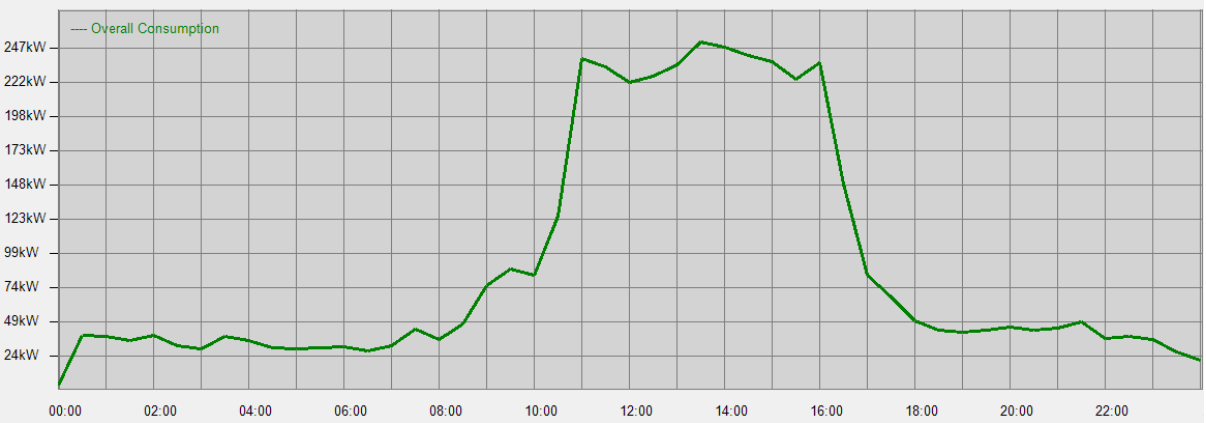


Figure 70 - Daily report for Jan 11, case 4

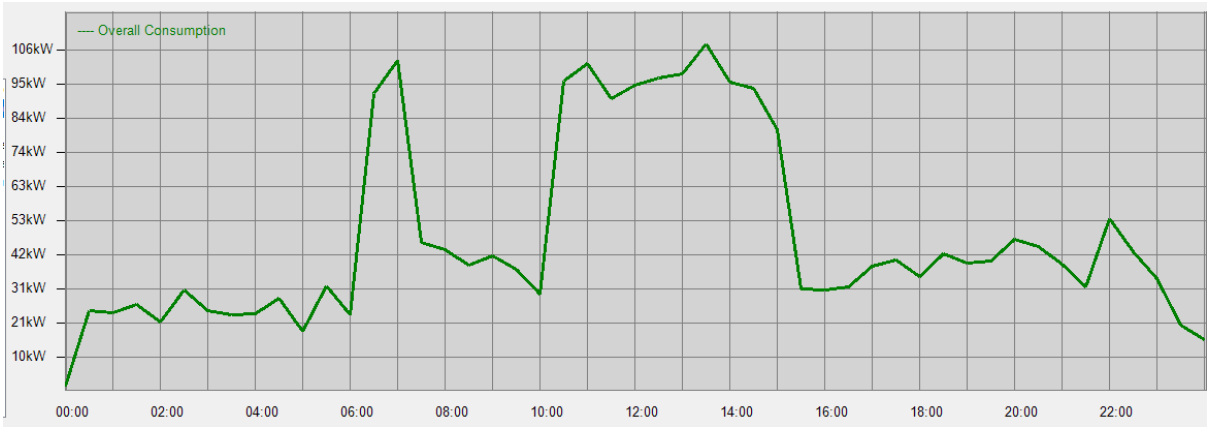


Figure 71 - Daily report for Aug 1, case 4

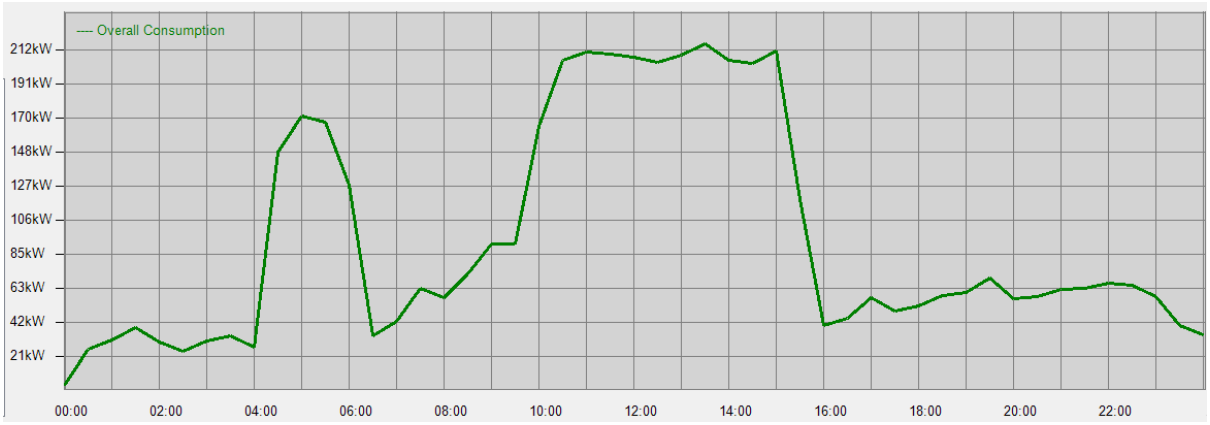


Figure 72 - Daily Report for July 21, case 4

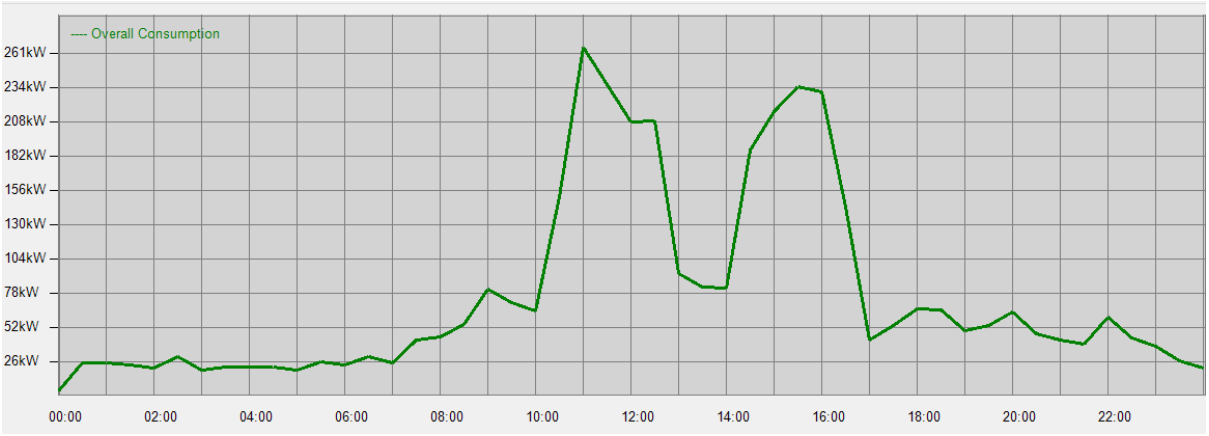


Figure 73 - Daily Report for Dec 22, case 4

Figure 70, Figure 71, Figure 72 and Figure 73 show the daily loads for each day associated, to give a base understanding of the microgrid's behaviour. The consumption for the four days averages at 3.7MWh. Based on the four sample reports, the daily consumption can be expected to sit between 50 and 80 kWh when the factory is not running its high-power equipment and shoot right up to 300-400 kWh when the factory is working. A yearly report (Figure 74) will now be generated to obtain some insights into yearly power usage. Note, the initial generation time for the first-year report was around 60s, as factory data was read in as it was processed, this will be much less on concurrent generations due to caching.

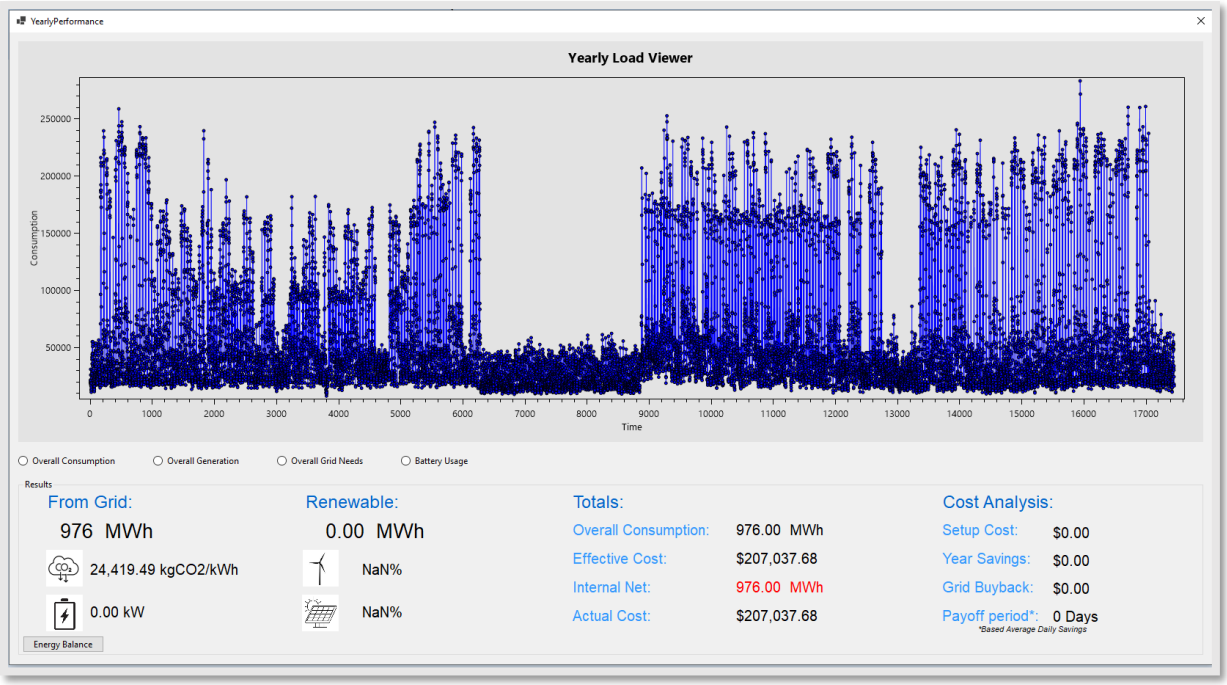


Figure 74 - Initial Yearly Power usage for case 4

There are a few interesting things to note about this yearly report before progressing. First, the gap in factory data that was identified earlier is present as expected, so it can be presumed that during the period of missing data, the peaks of power consumption would meet similar heights to those present in other areas of the report. Generally speaking, the power consumption of the factory is quite consistent across the board, but interestingly there appears to be a lower amount of consumption in the earlier months of the year. The other gaps in factory data are assumed to be public holidays, or other days in which the factory is not operating, which are important considerations when analysing on a broad scale.

The stakeholders and residents of the microgrid have agreed to invest in a large-scale wind turbine, which they believe they can place near to their shoreline for optimum wind utilization. Within MicroSim Dev, they see that the Enercon E82, 2300kW turbine is available from the pre-loaded wind turbine samples (Figure 75). At an estimated price of $2,645,000, this type of renewable generation is a large cost, and can be considered as a risk, but with reliable power curve data associated with this model, its performance can be accurately modelled. Since its rated power will likely provide much more power overall than what is being consumed, the community agree to sell excess power to the grid where applicable. In this scenario, it is assumed to be best case grid buyback, and while this is not realistic, it should be stated again that grid buyback fluctuation is not something incorporated into MicroSim Dev.

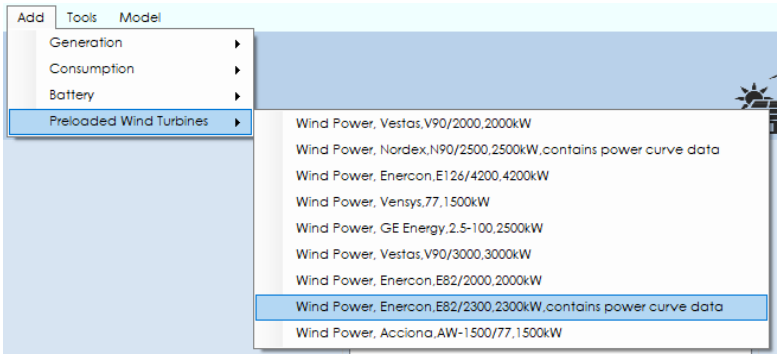


Figure 75 - Adding preset wind power

Some general tests to see the turbines performance are undertaken using the load comparison tool, and windspeed tool. Since the highest-consuming day from the sample day reports was July 21, this day will be used to explore the performance of the desired wind turbine.

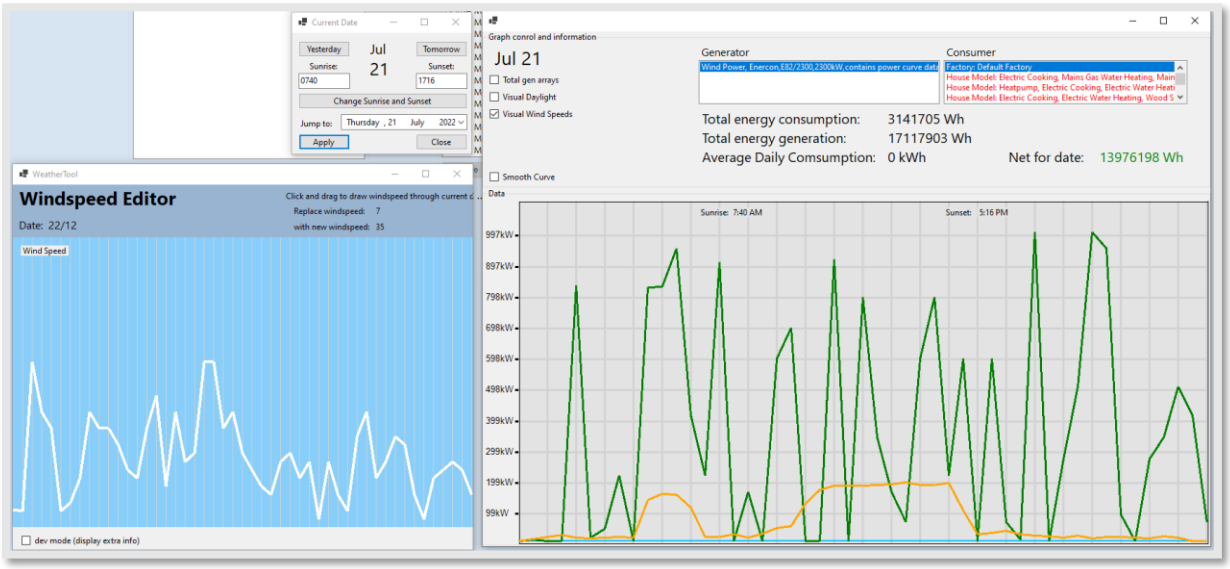


Figure 76 - Windspeed editor and date changer used to alter the simulation conditions

Figure 76 demonstrates the tools mentioned, alongside the daily performance of the desired wind turbine against the factory load. It is clear the turbine performs well, however the wind for this day is favourable. To get an idea of a low-wind day alongside this, the wind speed editor is utilized to draw a much lower distribution of windspeeds throughout the day.

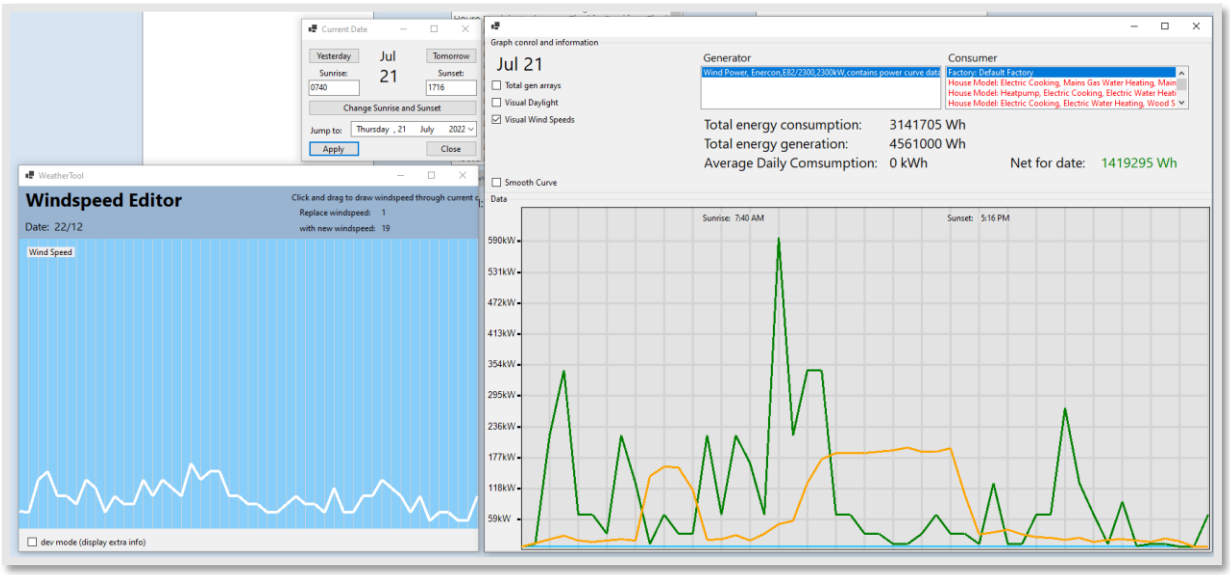


Figure 77 - windspeed editor is used to lower windspeeds for July 21

Figure 77 illustrates how using the windspeed tool to lower the windspeeds have a real-time, direct impact on how well the wind turbine performs in the load comparison tool. Even when there is a surplus in net generation, there is still a period of time when demand for consumption exceeds supply. The neighbourhood also agrees to invest in 5kWh of power storage for each home to ensure that power does not go out if they were to rely solely on the efforts of this wind turbine. This is to account

for the unavoidable times of the year when demand for power is greater than supply. Each household agrees to consider installing 6kW batteries, utilizing the same model that was used in Case 2. With this, 54 6kW Lithium-Ion Batteries are added to the grid, to effectively total a large shared 324kWh of capacity. With these batteries incorporated in the microgrid, a daily report for July 21 (with the changes to wind retaining) can be viewed to see the effect the batteries will have on the grid. The daily report is shown in Figure 78.

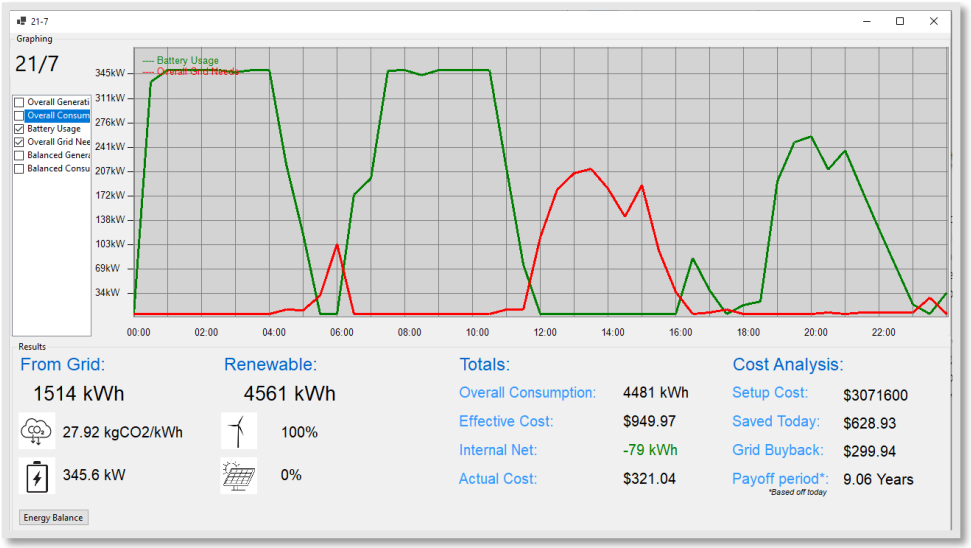


Figure 78 - Day report showing battery impact on microgrid

The report shows the battery level time-series alongside the grid needs time-series. Comparing these two time-series, the microgrid community could deduce that the storage holds up mostly well on a low-wind day. Unfortunately, during a time of high demand, there was no incoming wind power, and due to the extended period of demand from 12:00 – 16:00, the storage would likely run dry as depicted by the time-series. The community decide to re-generate the yearly statistics again, to see how things line up in comparison to earlier. The generated annual report (Figure 79) presents a promising outlook, requiring only 1.5 seconds to generate.

The reports cumulative data reveals that the selected wind turbine, operating within the context of an average distribution of wind speeds modelled through the Weibull distribution, significantly exceeds the energy demands of both the community and factory by approximately 100 MWh (Internal Net). If the microgrid were to rely solely on grid power, the estimated power cost for this projected year would have amounted to approximately $200,000. However, by harnessing wind power and effective energy storage, this cost is reduced to a mere 5% of the original estimate. This finding indicates that the current microgrid setup, despite not being fully optimized in terms of cost and battery models, is already 95% sustainable. This insight is invaluable in understanding the potential feasibility of creating a sustainable community with similar aspirations.

The grid buyback is currently assumed to be best-case, where power can be sold for its buying price, which puts the payoff period at a small 5.8 years. Even if a more conservative estimate of $100,000 is assumed for grid buyback, the expected payback period for the grid assets (batteries and turbine) would be approximately 10 years. This aligns with the community's goal of establishing a sustainable, cost-effective long-term solution.

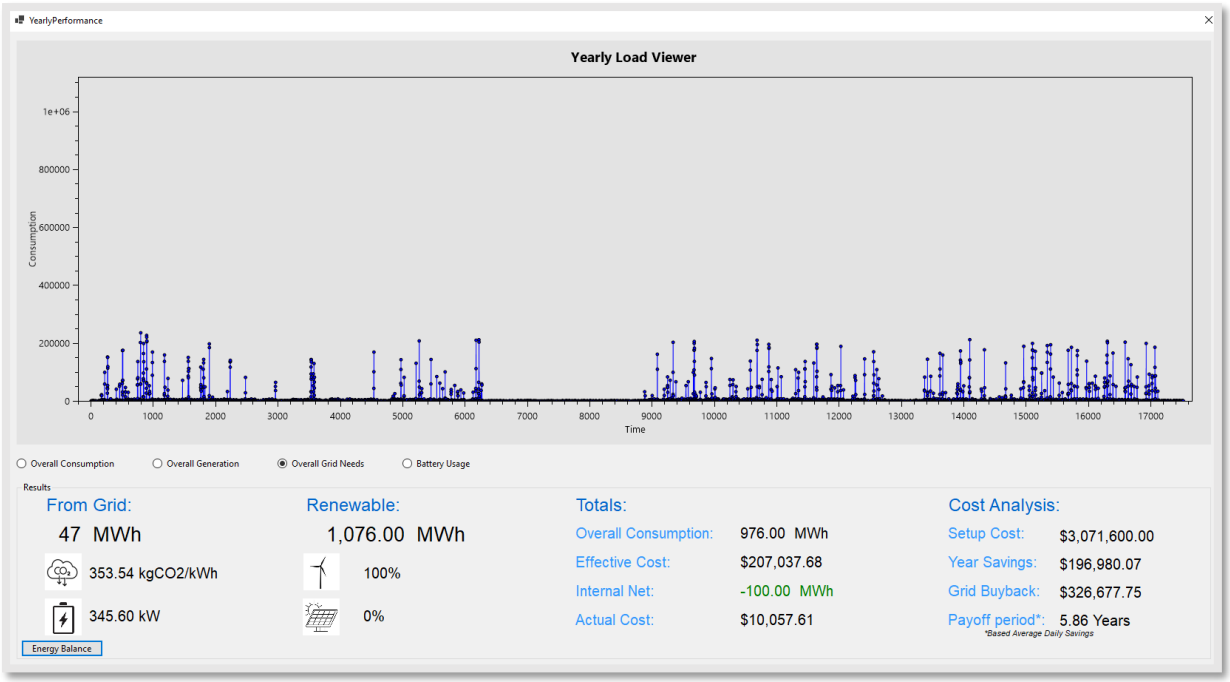


Figure 79 - Year report with grid needs after adding storage and wind power

This case study underscores several key insights and capabilities of MicroSim Dev in the context of microgrid design and exploration. Firstly, it highlights the power of trend identification without the need for intricate design changes. By thoroughly analysing load data and associated metrics, users can glean valuable insights into the performance and efficiency of their microgrid setups. The study also emphasizes the accuracy and effectiveness of modelling wind turbine performance, particularly when a turbine model is equipped with an associated power curve. This level of detail allows for precise predictions of energy generation, enabling users to make informed decisions about turbine selection and placement within their microgrid. Alongside this is MicroSim Dev's ability to incorporate weather variations into the scenario and persist these changes between report generations. This dynamic weather simulation allows users to assess how microgrid adjustments fare under different weather conditions, providing a valuable tool for scenario testing and optimization.

In summary, MicroSim Dev serves as an exceptional platform for initiating and exploring microgrid designs. Users can easily manipulate assets, make scenario-specific adjustments, and regenerate reports to explore and optimize their microgrid configurations. Whether users seek to meet specific energy needs or gain a deeper understanding of their data, MicroSim Dev empowers them to embark on a journey of discovery within the realm of microgrid design and sustainability, and aids users in uncovering more about the feasibility of a sustainability focused microgrid, and furthermore a sustainability focused New Zealand, thus meeting the criteria of this thesis.

6 CONCLUSIONS

6.1 SUMMARY

In the pursuit of advancing sustainable and efficient microgrid design, this thesis has designed, developed, and explored the MicroSim Dev application. With a meticulous approach to development, this software tool offers a versatile and user-centric platform for the modelling, analysis, and exploration of microgrid configurations. By drawing upon background research and carefully implementing a thorough design and wide array of components, MicroSim Dev facilitates the comprehensive exploration of microgrid design aspects. The thesis commenced by defining the fundamental objectives and challenges of microgrid design, emphasizing the need for innovative solutions to meet modern energy demands while reducing environmental impact. The direction of research within this field was also heavily influenced by the Ahuora initiative, particularly within Aim 3, which focuses on renewable energy potential, factory load profiles and renewable energy balance. It was within this context that MicroSim Dev emerged as a sophisticated solution to address the initial criteria and issues.

Throughout this discourse, the thesis meticulously detailed the architecture and functionality of MicroSim Dev. It delved into the creation of house model, factory model, solar panel, wind turbine, and energy storage unit objects, allowing for a granular representation of assets within the microgrid. The software's flexibility was underlined, enabling users to customize these objects to align with their specific requirements, and retain as much user freedom as possible. The discussion expanded to include the representation of daylight patterns and wind speed distribution, recognizing the pivotal role that environmental factors play in microgrid performance. This thesis showcased MicroSim Dev's adaptability by providing users the freedom to incorporate real-world data for precise modelling or changing the data within the application to meet their specific desires.

One of the prominent features unveiled was MicroSim Dev's support for load profiles, offering users the ability to define and customize energy consumption patterns. The software introduced factory models, enhancing the applicability of the tool to broader industrial settings. Users are empowered to explore the microgrid's behaviour at a high level, with the only limit being their own understanding. Energy balance plots also emerged as a valuable analytical tool within MicroSim Dev, providing a visual representation of energy generation and consumption trends. These plots offer a unique perspective on the dynamics of microgrid operation, and how well renewable energy is distributed and used within the microgrid. This thesis also highlights the reporting and analysis capabilities of MicroSim Dev, allowing users to produce analytical reports that provide in-depth insights into microgrid performance. Users have access to a variety of indicators, including cost analysis, consumption and generation data, grid usage, and related emission variables, all of which promotes well-informed decision-making.

Finally, case studies were presented as an evaluative testament to MicroSim Dev's utility, demonstrating its effectiveness in addressing real-world microgrid challenges, and simulating real world scenarios in which MicroSim Dev may be used for. From optimizing a single house's energy consumption to analysing the impact of renewable energy sources on a full microgrid, the software showcased its versatility and practicality.

6.2 SUGGESTED IMPROVEMENT

This study, while comprehensive in its exploration of MicroSim Dev and microgrid analysis, acknowledges certain limitations and identifies avenues for future refinement. The development of MicroSim Dev, along with the broader research, establishes a solid foundation for microgrid analysis and exploration. However, given the ambitious goals and expansive scope of this study, it is important to recognize that some aspects of microgrid modelling may not fully encapsulate real-world dynamics. The intricacies of digital twin design become particularly challenging when counterparts in the physical world are limited or absent, as evidenced by the custom wind turbine and solar panel designs, which may have certain inaccuracies.

It is crucial to reiterate for the final time that the overarching objective of this thesis is to create an application that empowers microgrid experts to investigate configurations tailored to their specific needs and concepts. With this goal carefully and accurately considered during implementation, the result primarily needs to be an environment that does not limit the users and is set up so the internal calculations and functions of each object can be refined to closely mirror their real-world counterparts.

In light of these considerations, a primary recommendation for improvement is the acquisition of more comprehensive real-world data. Collaborative efforts with subject matter experts and rigorous user testing, guided by experts, are essential for ensuring that the microgrid modelling aligns more closely with real-world scenarios. MicroSim Dev's success lies in its well-established and robust backend architecture. The reporting architecture stands to thrive from the enhancements that site matter experts and real-world data will bring, as they would require little to no adjustments. The foundational objects and structure within MicroSim Dev not only support the current application in its juvenility but also lays the groundwork for future enhancements by other developers or researchers to further advance its capabilities. Ultimately, MicroSim Dev is poised to evolve and thrive as a valuable tool for microgrid analysis and sustainable energy exploration.

6.3 FOLLOW ON WORK

The following suggestions are all features that were heavily considered when creating MicroSim Dev for this thesis but had to be rationalized out of this phase of development to contain the size and time it would take to complete this project.

6.3.1 Machine Learning

The integration of machine learning and artificial intelligence algorithms to fine-tune microgrid performance is the next obvious step. By assimilating real-time data and anticipatory analytics, these algorithms hold the potential to empower microgrid operators with astute decision-making capabilities concerning energy utilization and production.

6.3.2 Internal Microgrid Optimization

Internal optimization stands to be a powerful future development within MicroSim Dev. The utilization of well-known algorithms such as PSO, users would be able to obtain another metric of assistance outside static statistics and be nudged in the right direction of microgrid optimization when they are feeling stuck.

6.3.3 Locational objects

A heavier focus on locational aspects of MicroSim Dev would benefit the visual layout of the microgrid. Through location metadata for objects and associating objects on a real-world map, users can gain

another level of realism to their exploration, and take more dimensions into account, such as space for wind farms or solar panel locations at a start.

6.3.4 Closer Electronic analysis

Further development into electronic operations, such as power converters are an essential component that are not included in this stage of MicroSim Dev's lifecycle, but would again enhance the depth of realism and accuracy of modelling within MicroSim Dev.

6.4 CLOSING REMARKS

In conclusion, this thesis has not only introduced MicroSim Dev as a powerful tool for microgrid design exploration but has also demonstrated its real-world applicability and potential for driving sustainable energy solutions. The software's meticulous development, user-centric features, and flexibility position it as a valuable asset in the pursuit of designing efficient, eco-friendly microgrids. By fostering innovation, analysis, and informed decision-making, MicroSim Dev contributes significantly to the advancement of sustainable energy systems, marking a noteworthy contribution to the field of microgrid design and renewable energy integration.

7 REFERENCES

Altin, N., & Eyimaya, S.E. (2021). A Review of Microgrid Control Strategies. In 2021 10th International Conference on Renewable Energy Research and Application (ICRERA) (pp. 412-417). Istanbul, Turkey. doi: 10.1109/ICRERA52334.2021.9598699.

Amoura, Y., Pereira, A., & Lima, J. (2021). Optimization methods for energy management in a microgrid system considering wind uncertainty data., 117-141. https://doi.org/10.1007/978-981-16-3246-4_10

Apperley M. (2017). Modelling energy balance and storage in the design of smart microgrids. Proc Energy 2017, Barcelona, 40-45.

Aquino, C., & Unsihuay-Vila, C. (2021). A Hybrid Deep Learning Model for Solar Panel Performance Prediction Using Real Environmental Data. Energies, 14(11), 3273.

Badruhisham, S. H., Hanifah, M. S. A., Yusoff, S. H., Hasbullah, N. F., & Yaacob, M. (2022). Pi controller for hybrid biomass- solar photovoltaic- wind in microgrid: a case study of mersing, malaysia. IEEE Access, 10, 95151-95160. https://doi.org/10.1109/access.2022.3204671

Bai, X., Tsai, W. T., Paul, R., Feng, K., & Yu, L. (2002). Scenario-based modeling and its applications. In Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002) (pp. 253-260). San Diego, CA, USA. doi: 10.1109/WORDS.2002.1000060

Bazmohammadi, N., et al. (2022). Microgrid Digital Twins: Concepts, Applications, and Future Trends. IEEE Access, 10, 2284-2302. doi: 10.1109/ACCESS.2021.3138990.

Bhoi, S.K., Kasturi, K., & Nayak, M.R. (2023). Optimization of microgrid operation with renewables and battery storage. e-Prime - Advances in Electrical Engineering, Electronics and Energy, 4, 100159. https://doi.org/10.1016/j.prime.2023.100159.

Cieslak, K. and Dragan, P. (2018). Comparison of the existing photovoltaic power plant performance simulation in terms of different sources of meteorological data. E3s Web of Conferences, 49, 00015. https://doi.org/10.1051/e3sconf/20184900015

Edrisi, F., et al. (2021). EA Blueprint: An Architectural Pattern for Resilient Digital Twin of the Organization. In R. Adler, et al. (Eds.), Dependable Computing - EDCC 2021 Workshops (pp. 1-9). Communications in Computer and Information Science, vol 1462. Springer, Cham. https://doi.org/10.1007/978-3-030-86507-8_12.

Ferreira, W.M., Meneghini, I.R., Brandao, D.I., & Guimarães, F.G. (2020). Preference cone based multi-objective evolutionary algorithm applied to optimal management of distributed energy resources in microgrids. Applied Energy, 274, 115326. https://doi.org/10.1016/j.apenergy.2020.115326.

Guzman Razo, D. E., Müller, B., Madsen, H., & Wittwer, C. (2020). A Genetic Algorithm Approach as a Self-Learning and Optimization Tool for PV Power Simulation and Digital Twinning. Energies, 13, 6712. https://doi.org/10.3390/en13246712

Han, Y., Zhang, K., Li, H., Coelho, E., & Guerrero, J. M. (2018). Mas-based distributed coordinated control and optimization in microgrid and microgrid clusters: a comprehensive overview. IEEE Transactions on Power Electronics, 33(8), 6488-6508. https://doi.org/10.1109/tpel.2017.2761438

Jahn, U. and Nasse, W. (2004). Operational performance of grid-connected PV systems on buildings in Germany. Progress in Photovoltaics Research and Applications, 12(6), 441-448. https://doi.org/10.1002/pip.550

Jha, S. K., & Kumar, S. (2023). A Comparative Study on Wind Energy Assessment Distribution Models: A Case Study on Weibull Distribution.

Michalík, J., Danihelka, P., & Kulla, J. (2022). Wind Speed Modeling using Weibull Distribution: A Case of Liptovský Mikuláš, Slovakia.

Minerva, R., Lee, G. M., & Crespi, N. (2020). Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. Proceedings of the IEEE, 108(10), 1785-1824. doi: 10.1109/JPROC.2020.2998530

Mo, W., Lu, Z., Dilkina, B., Gardner, K.H., Huang, J-C., & Foreman, M.C. (2018). Sustainable and Resilient Design of Interdependent Water and Energy Systems: A Conceptual Modeling Framework for Tackling Complexities at the Infrastructure-Human-Resource Nexus. Sustainability, 10(6), 1845. https://doi.org/10.3390/su10061845.

Phommixay, S., Doumbia, M.L., & St-Pierre, D.L. (2019). Review on the cost optimization of microgrids via particle swarm optimization. Received: 15 July 2019 / Accepted: 14 December 2019.

Verma, A.K., Singh, S.K., & Garg, V.K. (2022). Modelling of Wind Speed Data using Weibull Distribution Function in Dewas.

Wang, Y., Sun, W., Liu, L., Wang, B., Bao, S., & Jiang, R. (2023). Fault Diagnosis of Wind Turbine Planetary Gear Based on a Digital Twin. Applied Sciences, 13, 4776. https://doi.org/10.3390/app13084776.

8 APPENDICES

GitHub Repository:

https://github.com/weirdoliam/Microgrid-Prototype-Design-Tool

OxyPlot can be found here:

https://github.com/oxyplot/oxyplot

https://oxyplot.github.io

Various products were referred to during case studies, which are noted below informally.

Example panel from case study 1:
Product Name: Solar Panel 460W Half-Cut Mono PERC
Manufacturer or Supplier: Fazcorp
Product URL:
https://fazcorp.co.nz/products/solar-panel-460w-half-cut-mono-perc?variant=39916990136433&currency=NZD&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic&utm_campaign=gs-2020-04-01&utm_source=google&utm_medium=smart_campaign&gclid=CjwKCAjwyNSoBhA9EiwA5aYlbxs5w9Qp-dPLXc1VA8VPtCim2n9LYTn96zTZXxsaqURR8SVQI9J79hoCxowQAvD_BwE

Example storage in case study 1:
Product Name: Power-UR-Wall 6.4kW Daily Cycle Solar Storage Battery
Manufacturer or Supplier: Hitek Solar
Product URL: https://www.hiteksolar.co.nz/products/power-ur-wall-6-4kw-daily-cycle-solar-storage-battery-new

Example wind for case study 2:
Web Page Title: Small Wind Turbine Size by Power Rating
Website: Attainable Home
URL: https://www.attainablehome.com/small-wind-turbine-size-by-power-rating/