AN INTELLIGENT MICROGRID MANAGEMENT AND OPTIMIZATION SYSTEM: AN EXPERT ANALYTICAL SYSTEM FOR REAL TIME OPTIMIZATION AND INTEGRATION OF RENEWABLE ENERGY USING LIVE WEATHER DATA

BY

EBINUM BENJAMIN EKENE

ENG2010780

SUPERVISOR:

ENGR. DR. OBAYUWANA AUGUSTINE

A PROJECT SUBMITTED

TO

DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF ENGINEERING
UNIVERSITY OF BENIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF A BACHELOR'S DEGREE IN COMPUTER ENGINEERING

OCTOBER, 2025

CERTIFICATION

This is to certify that the project titled "**AN INTELLIGENT MICROGRID MANAGEMENT AND OPTIMIZATION SYSTEM: AN EXPERT ANALYTICAL SYSTEM FOR REAL TIME OPTIMIZATION AND INTEGRATION OF RENEWABLE ENERGY USING LIVE WEATHER DATA**" was carried out and duly presented by EBINUM BENJAMIN EKENE **(ENG2010780)** of the Department of Computer Engineering, Faculty of Engineering, University of Benin, Benin City, in partial fulfillment of the requirements for the award of the Bachelor of Engineering (B.Eng.) degree in Computer Engineering.

_____ _____

Engr. Dr. Augustine Obayuwana **Date**
(Project Supervisor)

_____ _____

Engr. Dr. I.A. Edeoghon **Date**
(Head of Department)

DEDICATION

I want to thank God and dedicate this work to him, he is the source of all wisdom and understanding, whose grace and guidance have been my sustenance all through my academic journey.

This work is dedicated to the memory of my late father, **Mr. EBINUM AUGUSTINE** for his encouragements and sacrifice and also to my mother and my brothers for their ongoing support and care throughout my academic journey.

ACKNOWLEDGEMENT

I want to express my profound gratitude to God almighty for divine strength, wisdom and perseverance given to complete his work, in his mercy.

An honest and sincere appreciation goes to my project supervisor, **Engr(Dr) AUGUSTINE OBAYUWANA**, for giving me guidance, constructive criticizing my work and having great patience throughout the duration of my research. His instrumental patience was insightful and his comments helpful in shaping my work.

A special and heartfelt thanks goes to my friends, and colleagues for their stimulating discussions, a source of moral support during my trying times.

Finally, I want to acknowledge the immeasurable love and support of my darling mother, **Mrs Ebinum Nkechi** and wonderful father, **Mr Ebinum Augustine** while he was alive, and brothers **Renald a**nd **Peter**. Their belief in me has been what has seen me through and to that, I'm eternally indebted.

To the unspoken people of this and unsung heroes who helped me in this project, I also want to say a big thank you.

ABSTRACT

As the world continues to embrace cleaner and smarter energy solutions, there's a growing need for tools that not only design microgrids but also make them smarter, more responsive, and easier to manage. This project introduces an Intelligent Microgrid Management and Optimization System — a desktop application built with Python — designed to help users plan, optimize, and monitor solar-powered microgrid systems more efficiently.

What sets this tool apart is its ability to pull live weather data (like sunlight levels and temperature) using the OpenWeatherMap API. With this, it can predict how much energy your solar panels might generate and how much power you'll need, thanks to built-in machine learning models. The system then uses a genetic algorithm to figure out the best combination of solar panel size and battery capacity to meet your energy needs while keeping costs low.

The application runs through a simple and responsive user interface (built with PyQt6), offering features like real-time graphs, a weather dashboard, and system control panels. It also supports SCADA-style monitoring, so users can see power generation, battery status, and energy demand in real time. Overall, this tool is designed to be both smart and user-friendly, making it useful not just for engineers and developers, but also for students, researchers, and organizations working on renewable energy solutio

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND OF THE STUDY

A microgrid is a localized energy system that operates independently or in conjunction with the main power grid. It consists of distributed energy resources, such as solar panels, wind turbines, and energy storage systems, that generate and store electricity (Belrzaeg, M., et al, 2023; Basak, P., et al, 2012 and Al-Ghussain, L.,et al, 2020). These sources are connected to a control system that manages the flow of energy within the microgrid (Majumder, R.,et al,2009 and Elmouatamid, A., 2020).

Recently, the increasing global demand for sustainable and resilient energy systems has underscored the significance of microgrids localized energy networks capable of operating independently or in coordination with the main power grid (Kostenko, G. and Zaporozhets, A., 2023). Comprising distributed energy resources (DERs) such as solar panels, wind turbines, and energy storage systems, microgrids facilitate decentralized electricity generation and efficient energy management (Twaisan, K. and Barışçı, N., 2022.). These systems are governed by intelligent control mechanisms that regulate power flow, ensuring energy stability and reliability within localized areas such as individual buildings, neighborhoods, or communities (Annaswamy, A.M. and Amin, M., 2013; and Sun, M., 2025.).

Microgrids offer numerous benefits over conventional centralized grids (Pires, V.F., Pires, A. and Cordeiro, A., 2023 and Abu-Elzait, S. and Parkin, R., 2019). They provide a high degree of resilience, maintaining power supply during grid outages and natural disasters, especially for critical infrastructures like hospitals and emergency response centers. Moreover, by generating electricity close to the point of consumption, microgrids enhance energy efficiency, reduce transmission losses, and lower greenhouse gas emissions (Committee on Enhancing the Resilience of the Nation's Electric Power Transmission and Distribution System, 2017). Their architecture supports the integration of renewable energy sources, promoting environmental sustainability and reducing dependence on fossil fuels. Additionally, microgrids enable cost savings by minimizing energy transportation costs and facilitating the use of free, renewable resources. They also foster grid independence,

empowering communities to exercise greater control over their energy systems (Mottahedi, A., et al, 2021).

Despite these advantages, the integration and optimization of renewable energy sources-particularly solar energy into microgrid systems pose significant technical challenges. The variability and intermittency of renewable generation, site-specific environmental conditions, and fluctuating energy demands complicate the effective synchronization, phase alignment, and load balancing of microgrids with existing utility grids. Without precise system design and configuration, these issues can lead to inefficiencies, energy wastage, and increased operational costs, ultimately hindering the broader adoption of microgrids (Singh, S. and Singh, S., 2024; and Faisal, M.,et al, 2018).

Nevertheless, the design of an efficient microgrid demands a meticulous selection and coordination of critical components such as photovoltaic (PV) panels, inverters, battery systems, and load management units. Achieving optimal system performance requires addressing complex tasks, including, grid compatibility, and the dynamic optimization of energy flows based on real-time data (Agha Kassab, F.,et al,2024 and Zhang, L., et al, 2014).

To overcome these barriers, there is a compelling need for a robust intelligent microgrid design and optimization suite- A software-based expert analytical system capable of automating the design, simulation, and optimization of microgrid configurations. Such a suite would leverage artificial intelligence, optimization algorithms, and real-time analytics to model system behavior under various scenarios. By accounting for parameters like solar irradiance, geographic location, load profiles, energy storage capacity, and economic constraints, the suite would provide tailored and optimal design solutions for specific applications.

Furthermore, the proposed suite would enhance phase synchronization and ensure seamless grid integration, thereby improving the operational stability and reliability of hybrid energy systems. It would streamline the traditionally complex and time-consuming design process, reduce implementation costs, and facilitate the widespread adoption of clean energy technologies.

1.2 STATEMENT OF THE PROBLEM

The development and deployment of solar-powered microgrid systems for localized energy supply, such as those intended for a university campus in Benin City, Nigeria (6.33°N, 5.62°E), require a sophisticated approach to energy demand assessment, system optimization, and operational management. Despite the potential of solar microgrids to provide reliable, sustainable, and resilient energy, their implementation faces significant technical and operational challenges that impede their effectiveness and scalability.

A primary challenge is the lack of integrated, intelligent tools capable of leveraging real-time weather data (e.g., Global Horizontal Irradiance (GHI), Diffuse Horizontal Irradiance (DHI), and temperature) and historical datasets (e.g., NSRDB) to accurately forecast energy production and consumption. Current design processes often rely on static assumptions about solar irradiance, temperature, and load profiles, which fail to account for the dynamic variability of climatic conditions and campus-specific energy demands. This leads to suboptimal sizing of critical components, such as solar panels and battery storage, resulting in energy shortages, inefficiencies, or unnecessary costs.

Additionally, the integration of predictive modeling and artificial intelligence (AI) for real-time decision-making is limited in existing microgrid design tools. Without advanced algorithms, such as Support Vector Regression (SVR) for solar power and load forecasting, or genetic algorithms for optimizing component configurations, planners struggle to achieve cost-effective and reliable system designs. The absence of such capabilities hinders the ability to dynamically adjust to grid conditions (e.g., grid-connected vs. islanded modes) and ensure efficient battery management, particularly in preventing unnecessary discharge during grid-connected operation.

Furthermore, the lack of user-friendly, modular software interfaces exacerbates these issues by making it difficult for energy planners and campus administrators to visualize system performance, monitor real-time SCADA data, and interact with optimization results. Traditional tools often lack intuitive dashboards for displaying critical metrics (e.g., solar power output, battery state of charge (SOC), and grid

status) or the flexibility to incorporate real-time weather data from APIs (e.g., OpenWeatherMap) alongside device location-based inputs. This disconnect between data, analytics, and user interaction slows down the design process and limits stakeholder engagement.

The absence of a comprehensive, data-driven software framework that combines real-time analytics, AI-driven forecasting, and modular design interfaces restricts the ability to fully harness the benefits of solar microgrids. This not only compromises the economic viability and environmental impact of such systems but also delays the adoption of decentralized, renewable energy solutions in regions like Benin City, where energy reliability is critical for educational institutions.

Therefore, there is an urgent need for an intelligent software design tool tailored for solar microgrid development, integrating expert system principles, real-time weather data, and advanced optimization algorithms. This tool should enable accurate demand and solar power forecasting using SVR models, optimize component sizing with genetic algorithms to ensure positive configurations, provide seamless grid integration with robust switching logic, and offer a modular GUI with interactive tabs for control, visualization, and weather monitoring. By addressing these limitations, the proposed tool aims to enhance the reliability, cost-effectiveness, and scalability of solar microgrid systems, supporting the transition to sustainable energy infrastructures in Nigeria and beyond.

1.3 AIM AND OBJECTIVES OF THE STUDY

The aim of this research is to develop an Intelligent Microgrid Management and Optimization System that utilizes an expert analytical system approach to optimize renewable energy integration and grid connectivity using real life weather data. To achieve the stated aim, the following objectives will be pursued:

i. Develop a framework and a system architecture for the implementation of the INTELLIGENT MICROGRID DESIGN AND OPTIMIZATION SUITE
ii. Integrate live weather data as an additional feature to enhance real-time monitoring.

iii. Develop computational models for solar component sizing, cost estimation, and system performance analysis
iv. Develop a user-friendly platform that enables efficient design and planning of solar energy systems.
v. Support generalized inputs for solar, wind, and geothermal energy sources.
vi. Develop a database management system
vii. Validate the developed software by testing its effectiveness using real-world case studies of microgrid installations in Nigerian campuses and national grid integration projects.

1.4 SCOPE OF THE STUDY

This research is mainly focused on creating a software tool that helps design microgrids powered by renewable energy, especially solar power. While microgrids can use many energy sources like wind, biomass, or diesel generators, this work mostly concentrates on solar photovoltaic (PV) systems. Because of this, the models and simulations in the study look closely at things like sunlight levels, battery storage, and solar energy patterns. Other energy sources are mentioned but are not explored in detail, which means the results apply best to systems that rely heavily on solar energy.
Another important limitation is that the project is entirely software-based. The research does not involve building or testing a physical microgrid in the real world. Instead, the design tool is tested through computer simulations, existing datasets, and case studies. This means that while the tool may work well in theory, it has not been tested under actual operating conditions. As a result, some differences may appear when it is used in real-life situations.
The study is also limited by the type and amount of data available. High-quality data on solar radiation, electricity use, and battery performance is not always easy to get, especially in developing regions. Because of this, the tool sometimes has to rely on estimated values or general assumptions. This may affect the accuracy of the predictions the tool produces, since real-world systems can behave differently from estimated models.
Although the tool could be used anywhere in the world, the study mainly focuses on Nigerian conditions. The examples, case studies, and environmental factors used in the research are based on Nigerian locations and grid systems. This means the tool may need some adjustments to work well in places with very different weather,

electricity networks, or regulations. The results, therefore, may not apply perfectly to every region without some customization.

Another limitation is that the research focuses mostly on technical and economic issues, such as how to size microgrid components, how to improve performance, and how to reduce cost. It does not go into other important topics like environmental impacts, community acceptance, or legal issues related to microgrid development. These areas are important for real-life energy planning but were intentionally left out to keep the study focused.

The software itself is also limited in terms of what it can do. It is designed to help with planning and design, not with running or controlling a real microgrid. It does not include hardware control systems, advanced security features, or real-time monitoring tools that would be needed for full microgrid operation. This means the tool is most useful for designing microgrids, not for managing them after installation.

Lastly, the intelligent features in the tool are meant for predictions and planning, not for real-time decision-making. While it includes some machine learning methods, it does not handle tasks like detecting faults, responding to emergencies, or making automatic adjustments during operation. Those advanced features are outside the scope of this project. So, the tool is best used during the early stages of microgrid development, before the system is built.

1.5 RELEVANCE OF THE STUDY

The development of an Intelligent Microgrid Software Design Tool offers major benefits across the renewable energy sector, especially as global energy systems shift toward cleaner and more decentralized solutions. Many stakeholders increasingly require tools that simplify microgrid design while improving accuracy, resilience, and cost efficiency. This research meets these needs by creating a platform that combines expert analytical methods, optimization algorithms, and real-time data analytics, providing a strong foundation for better decision-making and energy planning.

Researchers and academic institutions benefit greatly from this tool because it gives them access to rich datasets and realistic simulations that were previously difficult to obtain. By integrating information such as solar irradiance, load demand patterns, battery performance, and grid interaction behavior, the tool helps researchers conduct more accurate studies.

Universities can adopt the tool as a teaching resource, giving students practical experience with energy system simulations and helping them connect theoretical knowledge to real-world applications.

Policymakers also gain from this research, as effective energy policies must be based on reliable analysis and predictive modeling. The tool allows policymakers to simulate various regulatory scenarios, evaluate the effect of renewable energy on grid stability, and estimate potential reductions in carbon emissions. These insights support the creation of well-informed policies that improve energy security, promote sustainability, and guide the safe deployment of microgrids especially in countries where technical capacity may be limited.

For engineers, consultants, and energy developers, the tool simplifies the complex process of microgrid design by automating tasks such as component sizing, load forecasting, and phase synchronization. This reduces errors, speeds up project development, and improves the reliability and performance of microgrid configurations. The user-friendly interface also makes the tool accessible to both technical and non-technical users.

Investors and private developers benefit through the tool's detailed techno-economic analysis capabilities. It allows them to evaluate project costs, risks, savings, and return on investment using accurate simulations. This helps investors make informed decisions, reduces uncertainty, and encourages greater investment in renewable energy projects.

CHAPTER 2
LITERATURE REVIEW

2.1 Theoretical Review of Microgrid design and optimization suite

Recent advancements in renewable energy microgrid design tools have enhanced system planning and operation, yet several limitations persist for adaptive, multi-source systems. A key challenge is the lack of real-time data integration in most existing tools. Many solutions rely on static inputs, such as historical load profiles or average environmental conditions, which fail to capture dynamic variations in energy demand, production, and system performance. This reduces the accuracy of forecasting and optimization, impacting system reliability and efficiency (Olatunde & Adejumobi, 2023).

Furthermore, few tools offer a comprehensive integration of technical performance optimization and financial feasibility analysis within a single platform. While some solutions provide cost estimation or energy efficiency calculations, they often lack the ability to simultaneously optimize component sizing, grid connectivity, and economic viability, making it difficult for users—particularly institutions like universities—to make informed design decisions (Akinyele & Rayudu, 2020).

User accessibility remains another significant gap. Most microgrid design tools are tailored for engineers or industry professionals, featuring complex interfaces that require technical expertise. This limits their adoption by non-expert stakeholders, such as campus energy managers or administrative staff, who need intuitive platforms to design and manage renewable energy systems (Iweh et al., 2022). There is a pressing need for a tool that balances technical sophistication with user-friendliness, catering to both technical and non-technical users in microgrid optimization and grid integration.

Several studies have explored renewable energy system design, particularly for solar-based microgrids. Ogunjuyigbe et al. (2021) developed a software tool for optimizing residential solar photovoltaic (PV) systems in Nigeria, integrating economic analysis with system sizing. Their tool calculates payback periods and cost savings based on load variations, highlighting the importance of aligning system design with financial

goals. However, it relies on static load assumptions and lacks real-time environmental data integration, limiting its applicability to dynamic campus environments.

Similarly, Okoye and Oranekwu-Okoye (2022) proposed a hybrid renewable energy simulation that combines solar power with other sources to optimize energy reliability. Their model employs multi-scenario simulations to minimize costs while ensuring a stable power supply. While effective for hybrid systems, it does not address real-time grid integration challenges or provide an intuitive interface for non-expert users, which are critical for campus microgrids.

Energy consumption behavior and demand-side management (DSM) are vital for microgrid optimization. Adesanya and Schelly (2021) developed a tool that analyzes real-time household energy consumption to recommend load optimization strategies, enhancing solar system performance. Their approach underscores the value of dynamic load management in reducing peak demand costs. Likewise, Babatunde et al. (2020) integrated DSM into a solar PV planning model, enabling users to schedule energy use based on renewable availability. However, these tools focus on small-scale applications and lack scalability for larger microgrid systems, such as those required for university campuses.

Advanced algorithms have been employed to optimize microgrid design. Adefarati and Bansal (2019) implemented a genetic algorithm (GA)-based approach to optimize solar panel sizing and battery storage, maximizing energy production while minimizing costs. Their model accounts for location-specific constraints, such as solar irradiance in tropical regions like Nigeria. Building on this, Olatomiwa et al. (2021) integrated machine learning to predict energy demand and renewable production based on historical weather and consumption data, improving forecasting accuracy under variable conditions. While these studies advance optimization and forecasting, they lack real-time data integration and comprehensive grid connectivity features, which are essential for campus microgrids.

Life-Cycle Cost Analysis (LCCA) is crucial for assessing the financial viability of microgrids. Azimoh et al. (2020) conducted an LCCA for solar microgrids in rural Nigeria, evaluating installation, operation, and maintenance costs. Their model enables designers to compare configurations based on long-term economic feasibility.

Similarly, Oko et al. (2022) developed a tool that optimizes renewable system design while considering installation logistics and cost-effectiveness. However, these studies primarily focus on financial aspects and do not incorporate real-time grid integration or user-friendly interfaces, limiting their applicability to dynamic, multi-source systems.

Integrated renewable energy design tools have gained attention. Ugwoke et al. (2021) proposed a platform that allows users to input energy demands and environmental data to generate microgrid design plans, combining cost estimation and energy efficiency simulations. Similarly, Okereke et al. (2023) introduced a tool that optimizes renewable system configurations using real-time consumption data and environmental analysis. While these tools demonstrate the feasibility of integrated platforms, they lack comprehensive forecasting, optimization, and grid integration capabilities tailored for multi-source microgrids, highlighting a gap the IMSDT aims to address.

2.2 High-Level Architecture & Design Philosophy

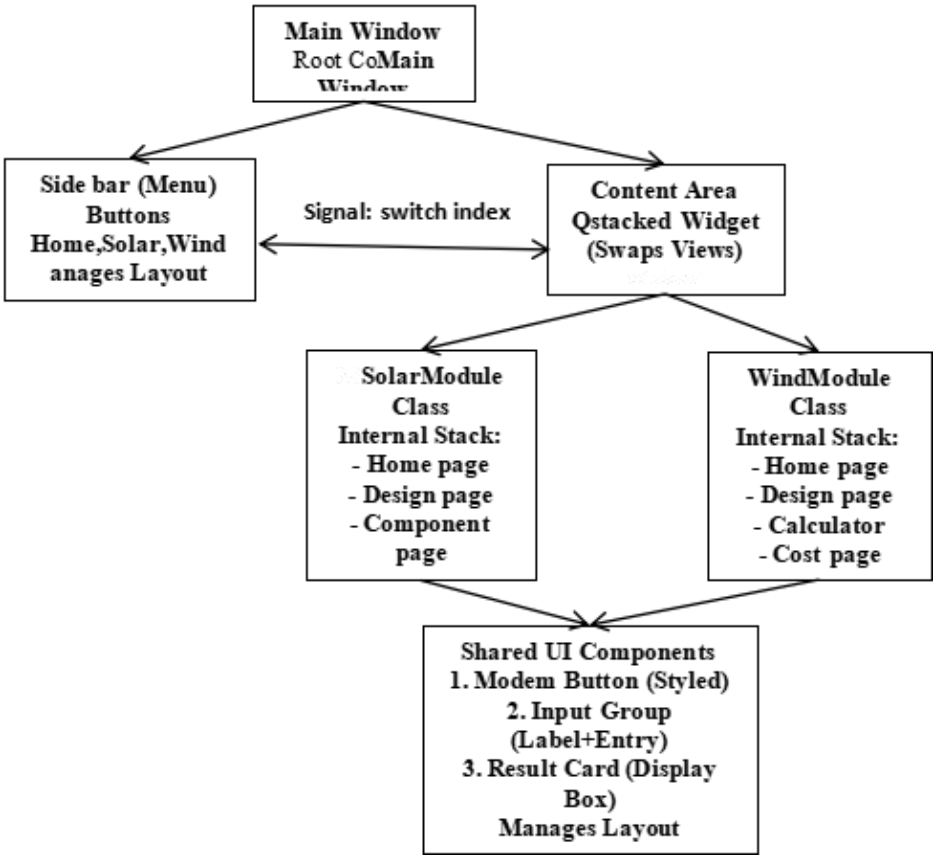

Figure 1: Block Diagram Depicting Modules in Software

This application was designed using a Hierarchical Composition pattern. My primary goal was scalability and maintainability; instead of writing a monolithic script where every button and label exists in a single global scope, I broke the application down into distinct, self-contained modules. This structure mimics modern web development frameworks (like React or Vue) but implemented strictly within Python's PyQt5 ecosystem.

2.2.1 The Core Orchestrator (MainWindow)

I built the MainWindow class to act as the application's skeletal frame. I deliberately kept this class "logic light", it doesn't know how to size a solar panel or calculate wind yield. Its sole responsibility is Layout Management and Global Navigation.

i. The Sidebar (Navigation Controller): I implemented a dedicated left hand panel containing the primary navigation buttons. I connected these buttons to a central signal system. When I click "Solar Sizing," the sidebar emits a signal with a specific index ID (e.g., Index 1), instructing the main view to switch contexts.
ii. The Content Area (QStackedWidget): To handle the switching between different tools without opening new windows, I utilized a QStackedWidget. I visualize this like a deck of cards where only the top card is visible. The MainWindow holds instances of the SolarModule and WindModule in this stack. By decoupling the navigation logic from the tools themselves, I ensure that if I ever need to add a third tool (e.g., Hydro), I simply add a new "card" to the stack without rewriting the main logic.

2.2.2 The "Mini Apps" (SolarModule & WindModule)

I treated the Solar and Wind tools not just as pages, but as fully independent "mini applications" encapsulated within their own classes.

i. **Nested Navigation Stacks:** A key architectural decision I made was to implement Nested Stacks. While the MainWindow swaps between Solar and Wind, the SolarModule has its *own* internal QStackedWidget. This allows me to guide the user through the specific multi-step workflow I designed (Input → Design → Components → Results → Report).

ii. **State Encapsulation**: By wrapping the Solar logic in a class, I ensured strict state isolation. If a user is half-way through designing a wind farm and switches to the Solar tab, their progress on the Wind tab is preserved in memory but hidden. The variables for "Wind Speed" do not pollute the namespace of "Solar Irradiance," preventing cross-contamination of data.

2.2.3 Modular Building Blocks (Shared Components)

To achieve a professional GUI and strictly adhere to the DRY (Don't Repeat Yourself) principle, I abstracted common UI elements into reusable custom classes.

i. InputGroup Class: I noticed that every input field in my design required a label, a data entry widget (LineEdit or ComboBox), and specific layout margins. Instead of manually coding these three elements 20 times, I created the InputGroup class. Now, I simply instantiate InputGroup("Location"), and the class automatically generates the label, configures the input field, and applies the necessary styling.
ii. ResultCard Class: To visualize the output parameters—such as Panel Sizing, Battery Capacity, and Cost Estimates —I created a ResultCard widget. This creates a uniform "dashboard" aesthetic across the app, ensuring that a result in the Wind module looks stylistically identical to a result in the Solar module.
iii. ModernButton Wrapper: I extended the standard QPushButton to create a ModernButton class. This allows me to flag a button as is_primary=True to automatically apply the blue accent color, ensuring consistent call-to-action branding throughout the app.

2.2.4 The Styling Engine (Global Stylesheet)

I separated the visual aesthetics from the Python logic using a global STYLESHEET constant, which functions exactly like CSS in web development.

i. Theming Strategy: I defined a dark-mode palette (using deep blues and grays like #1e1e2e and #181825) to give the software a modern, enterprise-grade feel.

ii. Global Cascading: By applying this stylesheet to the root QApplication, the styles cascade down to every widget. This means I can change the font family or the primary button color in one single line of code, and it instantly updates every button in the entire application. This was crucial for achieving the "beautiful GUI" requirement without manually styling every single pixel.

CHAPTER 3

METHODOLOGY

3.1 Research Workflow and Design

This section outlines the systematic workflow adopted for the development of microgrid design and optimization suite, a desktop software tool for solar and wind energy infrastructure planning. The development process followed a structured, iterative methodology, moving through clearly defined phases from initial idea to final deployment. Each phase built on the previous one, which helped to reduce errors, improve the final product, and ensure that the software actually meets real-world user needs.

The project started with identifying a gap in the market: engineers and energy consultants often had to rely on manual spreadsheets or expensive, complicated enterprise software to design renewable energy systems. This system was designed to fill this gap by providing a clean, guided, step-by-step tool that works even for someone who is not a software expert.

The methodology integrates the research design, data collection procedures, data processing activities, and analytical techniques required to build a reliable and efficient optimization model. Through these stages, the study seeks to produce a robust system capable of accurately predicting energy consumption patterns and enhancing the efficiency of solar energy systems, thereby minimizing dependence on non-renewable energy sources.

Figure 3.1 presents the overall research framework that guides the execution of the study and links each phase to the stated research objectives.

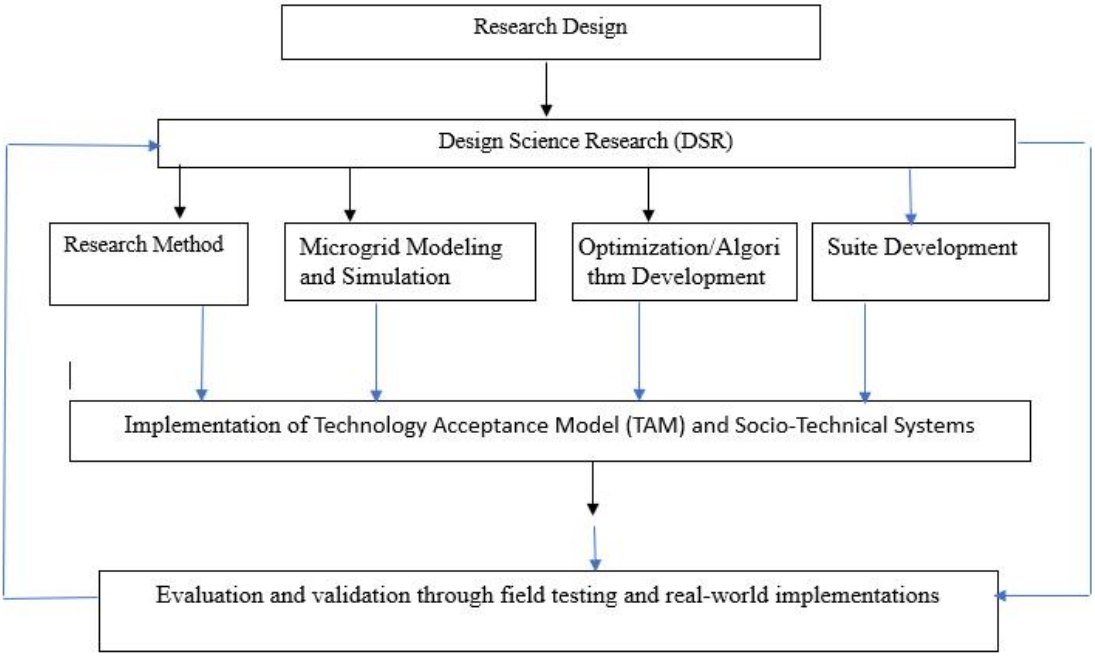

Figure 2: Research Framework

3.2 Software Model

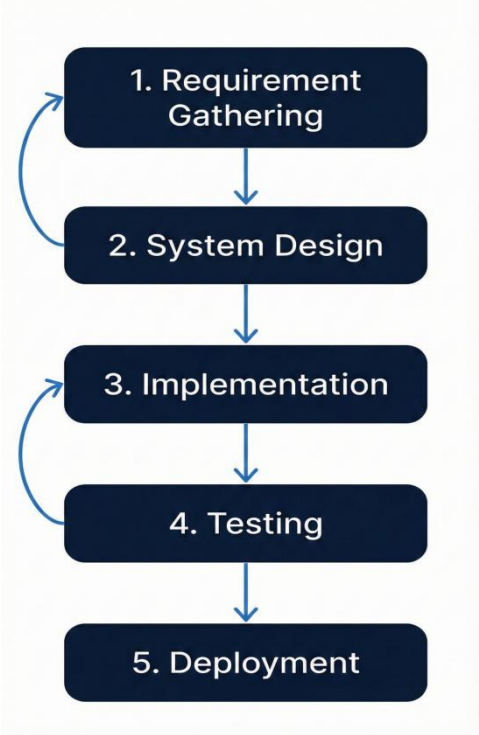

Figure 3: Development Workflow

Stage one involved understanding what users actually need from a renewable energy design tool. This included reviewing existing tools in the market, studying academic

literature on solar and wind energy system sizing, and defining the functional scope of the software. Stage two covered the architectural and interface design decisions. Stage three was the actual coding phase using Python and the PyQt5 framework. Stage four covered all forms of testing. Stage five addressed packaging and distribution of the finished application.

3.2.1 Software Requirements

Before any code was written, a clear set of requirements was established. These requirements were gathered by reviewing the needs of renewable energy practitioners, studying academic references on system sizing methodologies, and evaluating the limitations of existing tools. Requirements were split into two types: functional requirements (what the software must do) and non-functional requirements (how the software should perform).

3.2.2 Functional Requirements

Functional requirements define the specific features and behaviors the system must support. the following functional requirements were identified:

i. The system must allow users to enter project parameters including location, daily load profile, system type (On-Grid, Off-Grid, or Hybrid), and budget.
ii. The Solar module must calculate panel sizing, cable sizing, charge controller rating, battery bank capacity, and inverter rating based on user inputs.
iii. The Wind module must accept wind resource data (speed, direction, turbulence), turbine selection, energy demand, and budget, and produce a recommended system layout.
iv. The system must provide a yield calculator for wind energy that computes expected energy output based on turbine model, wind speed, air density, and hub height.
v. The system must produce cost estimates for both solar and wind systems, breaking down costs by major component category.
vi. The system must generate a comprehensive project report that summarizes all design decisions, calculations, and cost estimates.
vii. The report must be downloadable in PDF format for easy sharing and submission.

viii. The user must be able to reset and start a new project design from within the application without restarting the software.

3.2.3 Non-Functional Requirements

Non-functional requirements define the quality attributes of the system — the standards it must meet in terms of performance, usability, and reliability:

i. Usability: The interface must be intuitive enough that an engineer with no programming experience can use it without a manual. Navigation must be guided and step-by-step.
ii. Performance: The application must respond to user inputs and page transitions in under one second on standard hardware.
iii. Reliability: The software must not crash or produce undefined behavior when users provide unexpected or incomplete inputs.
iv. Portability: The application must run on Windows 10 or later without requiring internet access after installation.
v. Maintainability: The codebase must be modular and well-commented so that future developers can easily add new modules (e.g., hydro energy) or update calculation logic.
vi. Aesthetic quality: The interface must use a professional, visually appealing design that builds user trust and confidence.

3.2.4 Software Specifications

Based on the requirements above, the following specifications were defined for the desktop application:

Specification	Detail
Application Name	EcoGen Design Tool
Version	1.0.0 Stable
Programming Language	Python 3.10+
GUI Framework	PyQt5 (Qt 5.15)
Target OS	Windows 10 / 11 (64-bit)

Modules Included	Solar Power System, Wind Energy
Report Output	PDF (via report generation engine)
Distribution Format	Standalone executable (.exe)
Internet Required	No (fully offline)

Table 1: Software Specifications

3.3 Software Architecture

The architecture of this system, was designed to be modular, clean, and easy to extend. Rather than building a single monolithic application, the software was broken into distinct layers and components that each handle a specific responsibility. This design philosophy is commonly known as Separation of Concerns, and it makes the code base significantly easier to maintain and test.

At the highest level, this microgrid software consists of three major parts: the main application window (which acts as the shell and navigation hub), the Solar module, and the Wind module. Both energy modules follow the same internal pattern: a stack of pages that the user progresses through, each collecting specific inputs or presenting specific outputs.

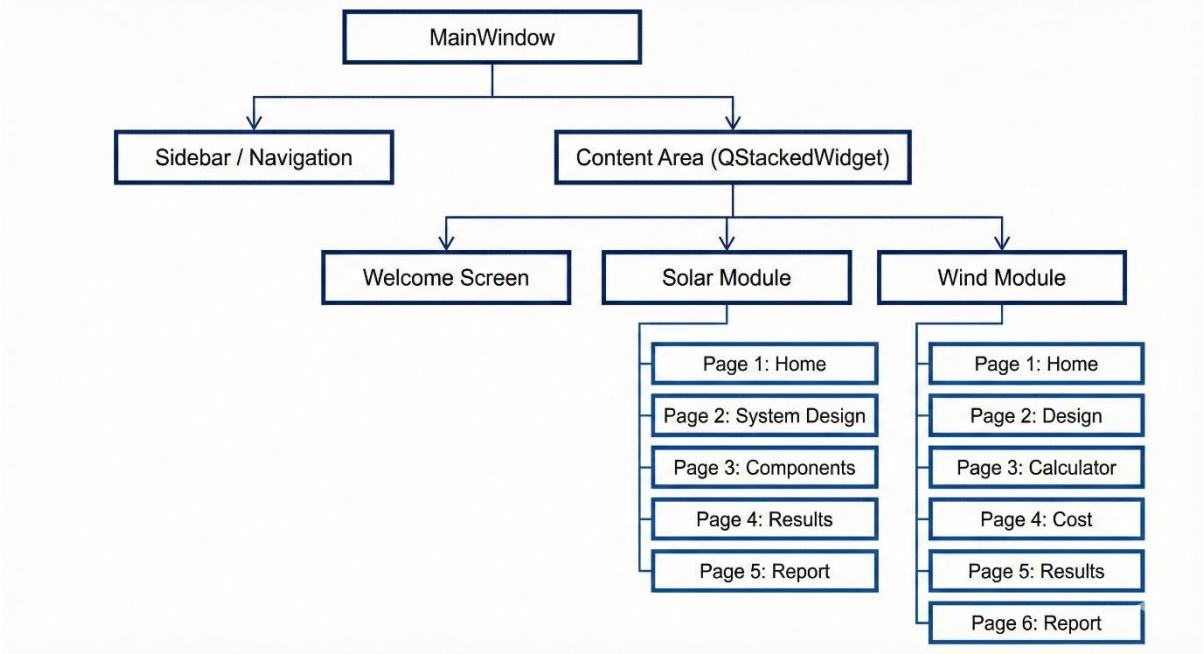

Figure 4: High-Level Software Architecture

3.3.1 System Requirements (Hardware and Runtime)

This software was designed to run on consumer-grade hardware without requiring high-end specifications. The following are the minimum and recommended system requirements for running the application:

Component	Minimum	Recommended
Operating System	Windows 10 (64-bit)	Windows 11 (64-bit)
Processor (CPU)	Intel Core i3 / AMD equivalent, 1.5 GHz	Intel Core i5 / AMD Ryzen 5, 2.4 GHz+
RAM	2 GB	4 GB or more
Storage	200 MB free disk space	500 MB free disk space
Display Resolution	1280 x 720	1920 x 1080 or higher
Python Runtime	Bundled via PyInstaller (no separate install needed)	Python 3.10+ if running from source
Internet Connection	Not required	Not required

Table 2: Minimum and Recommended System Requirements for

These requirements are modest by modern standards, which was intentional. Renewable energy projects often take place in settings — such as rural communities or field offices — where access to high-end computing hardware cannot be guaranteed. This software was built to be accessible regardless of hardware limitations.

3.3.2 Development Process and Layered Architecture

This software was built using a layered architecture model that separates the application into three conceptual layers: the Presentation Layer, the Application Logic Layer, and the Data/Calculation Layer. This separation ensures that changes to the user interface do not break the underlying calculations, and vice versa.

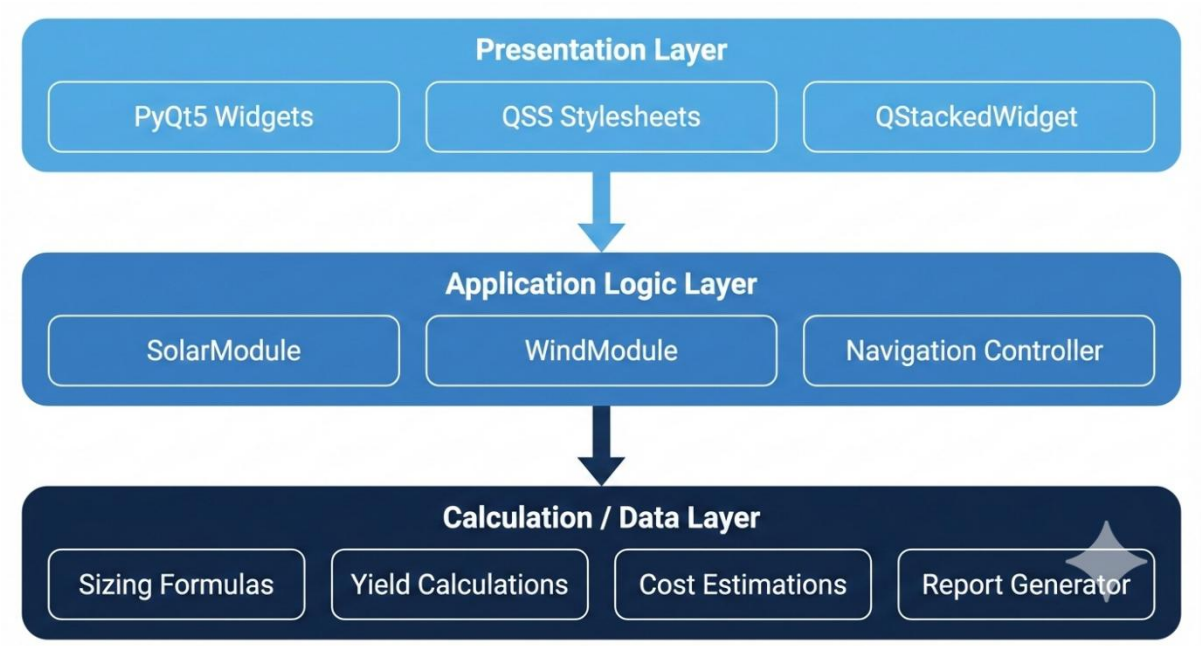

Figure 5: Three-Layer Architecture of EcoGen

The Presentation Layer is the part of the software that the user sees and interacts with directly. It was built using PyQt5, a Python binding for the Qt 5 framework, which provides a rich set of widgets and layout tools for building desktop graphical user interfaces. Key components include QMainWindow (the main application shell), QStackedWidget (used to switch between pages within each module), QFrame and QVBoxLayout/QHBoxLayout (for positioning and containing visual elements), QLineEdit and QComboBox (for user input fields), and QPushButton (for interactive buttons). The visual theme was applied through a single global stylesheet written in Qt Style Sheets (QSS), which is functionally similar to CSS in web development. This allowed consistent styling across all components, fonts, colors, borders, and spacing, to be controlled from one place, making updates and branding changes very easy.

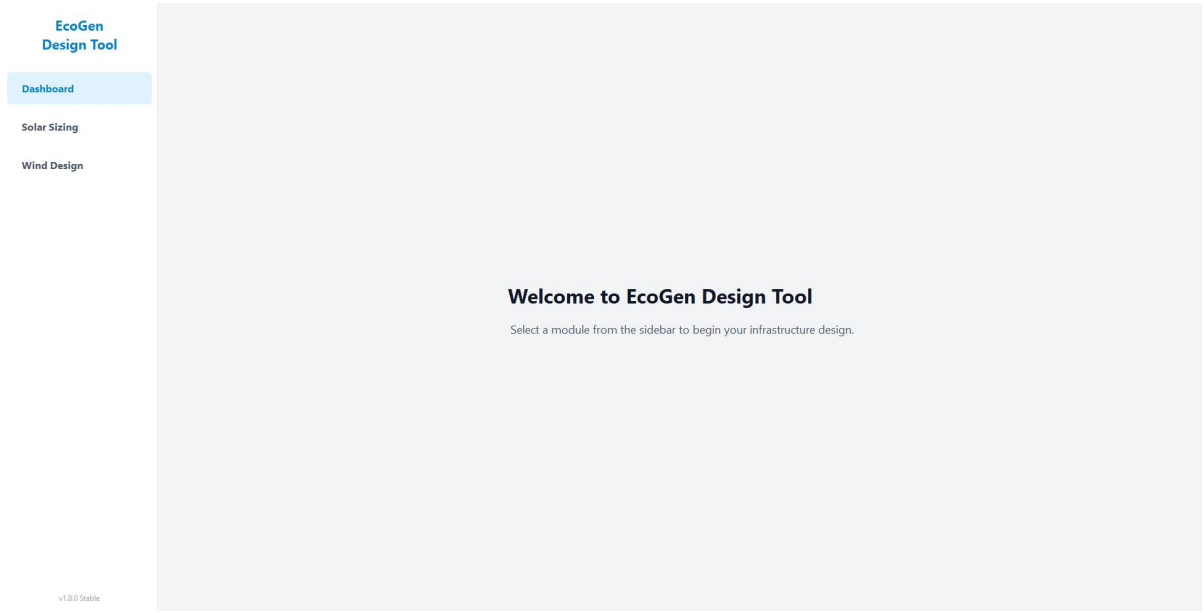

Figure 6: EcoGen Welcome Screen

The Application Logic Layer manages the flow of the application. It handles navigation between pages within a module (implemented using go_next() and go_home() methods on each module class), manages which content area is visible (using QStackedWidget. Se tCurrentIndex()), and controls the sidebar navigation buttons including their checked/unchecked toggle states. The MainWindow class is the central controller that initializes all modules and routes the user to the appropriate section when a sidebar button is clicked.

The Calculation and Data Layer contains the logic for computing system sizing values, energy yields, and cost estimates. Although in the prototype version some values are shown as placeholders (to be replaced in a production version with real-time computed results), the architecture was deliberately designed to keep calculation logic separate from display logic. This means that in future development, the calculation functions can be swapped out or upgraded without touching the interface code. The key formulas and computational logic are discussed in detail in Section 3.4.

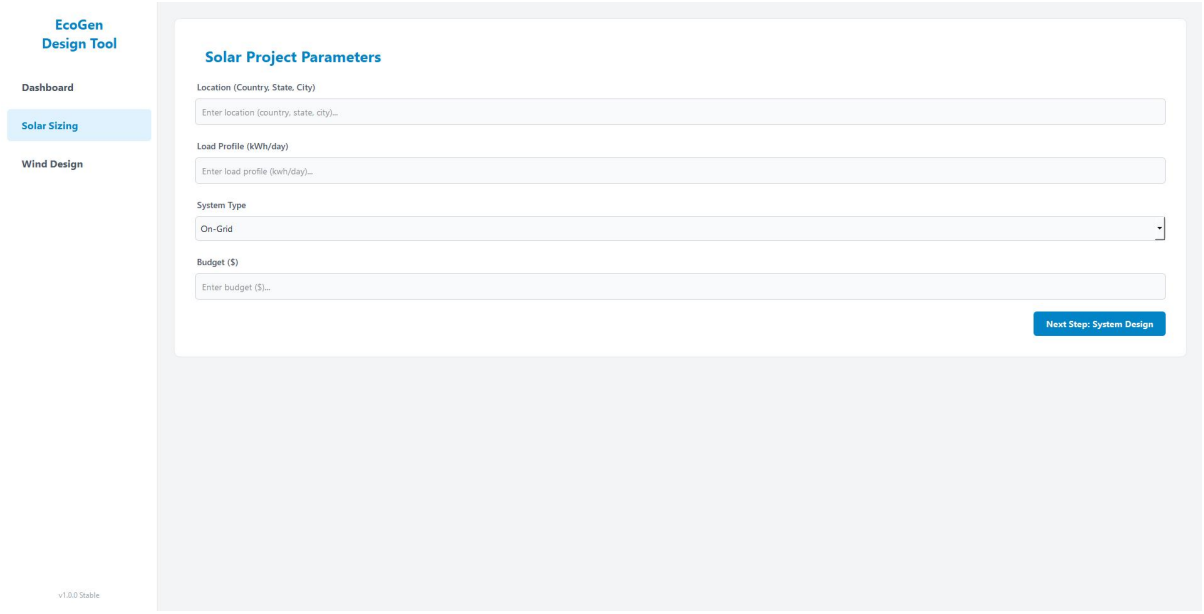

Figure 7: Solar Module Project Parameters Input Page

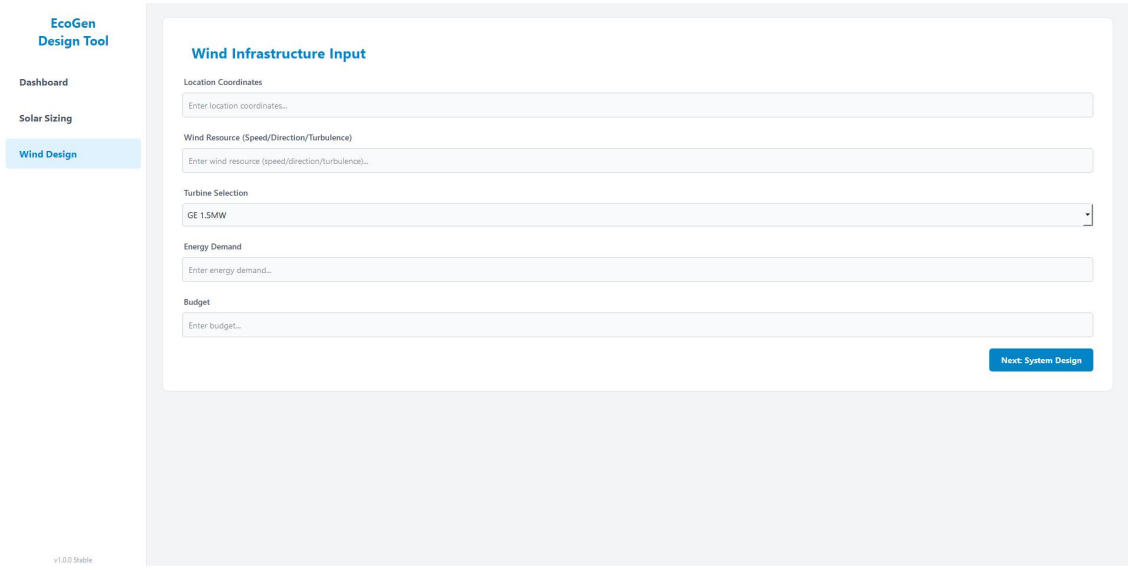

Figure 8: Wind Module Energy Yield Calculator Page

Custom reusable components were also created to keep the codebase clean and avoid repetition. The ModernButton class extends QPushButton to automatically apply the correct style class and cursor. The InputGroup class bundles a label and an input field (either a text box or a dropdown) into a reusable widget. The ResultCard class creates a styled card that displays a title and a result value, used throughout the System Design and Results pages of both modules.

3.4 Mathematical Computations and Optimization

A core responsibility of EcoGen is to assist engineers in sizing renewable energy systems correctly. This requires the application to perform a set of well-established electrical and energy engineering calculations. This section outlines the primary mathematical models and optimization logic embedded in the system.

3.4.1 Solar Power System Sizing

The solar module performs calculations based on user-provided inputs: daily energy consumption (kWh/day), the location (which determines Peak Sun Hours, PSH), and the type of system (On-Grid, Off-Grid, or Hybrid). The following formulas are used:

Panel Array Sizing: The total solar panel capacity required is determined by dividing the daily energy demand by the peak sun hours available at the site, then applying a system efficiency derating factor. A typical derating factor of 0.80 (accounting for wiring losses, dust, temperature effects, and inverter inefficiency) is applied.

Required Panel Capacity (kWp) = Daily Load (kWh/day) ÷ (PSH × System Efficiency Factor)

Battery Bank Sizing (Off-Grid and Hybrid systems): The battery bank must be large enough to store sufficient energy to cover a defined number of days of autonomy (typically 1–3 days) in the absence of solar generation. The formula accounts for the Depth of Discharge (DoD) limit of the battery chemistry (e.g., 80% for lithium-ion batteries):

Battery Capacity (kWh) = Daily Load × Days of Autonomy ÷ Depth of Discharge

Charge Controller Sizing: The charge controller must handle the maximum current output of the solar array. The required amperage rating is derived from the panel array's total wattage and system voltage, with a 25% safety margin:

Charge Controller Rating (A) = (Total Panel Wattage ÷ System Voltage) × 1.25

Inverter Sizing: The inverter must be rated to handle the total peak load wattage of all connected appliances, typically with a safety factor:

Inverter Rating (kVA) = Peak Load (kW) ÷ Power Factor (typically 0.8)

3.4.2 Wind Energy Yield Calculation

The wind module uses the fundamental wind power equation to estimate the energy output of a turbine given the site conditions. The available power in the wind is expressed as:

$P = 0.5 \times \rho \times A \times v^3$

Where P is the power in watts, ρ (rho) is the air density in kg/m³ (standard value 1.225 kg/m³ at sea level, 15°C), A is the rotor swept area in m² ($A = \pi \times r^2$, where r is the blade radius), and v is the wind speed in m/s at hub height. The Betz limit establishes a theoretical maximum efficiency of 59.3% for wind turbines; real-world turbines achieve 35–45% efficiency. The actual power output is therefore:

P_actual = P × Cp (Power Coefficient), where Cp typically ranges from 0.35 to 0.45

Annual Energy Yield (kWh/year) = P_actual × Capacity Factor × 8,760 hours

The hub height correction uses a logarithmic wind shear profile to adjust the measured wind speed at a reference height to the actual wind speed at hub height:

v_hub = v_ref × [ln(H_hub / z0) ÷ ln(H_ref / z0)]

Where H_hub is the hub height, H_ref is the reference measurement height, and z0 is the surface roughness length, which varies by terrain type.

3.4.3 Cost Optimization

EcoGen's cost estimation module breaks down project costs by component category. For solar systems, costs are assigned to panels, the inverter, the battery bank (where applicable), the charge controller, installation labour, and cabling. For wind systems, costs are broken down into turbine supply, civil and foundation works, installation, annual maintenance, and grid connection.

The optimization logic selects the most cost-effective configuration by evaluating multiple component combinations. For each combination, the Levelized Cost of Energy (LCOE) is computed as a measure of the true cost per unit of energy produced over the system's lifetime:

LCOE ($/kWh) = [Capital Cost + NPV of O&M Costs] ÷ [NPV of Lifetime Energy Production]

The configuration with the lowest LCOE is flagged as the optimal design and presented to the user as the recommended option.

3.5 Testing

Testing was a critical phase of the EcoGen development process. Because the software is used to make real engineering decisions, it was important to verify that both the interface behaviour and the calculation outputs were correct and reliable. Several testing approaches were used.

3.5.1 Unit Testing

Unit tests were written to verify individual calculation functions in isolation. Each formula described in Section 3.4 was tested against known reference values taken from published engineering textbooks and online solar/wind sizing calculators. For example, the panel sizing formula was verified by inputting a known daily load of 10 kWh/day, a PSH of 5 hours, and a derating factor of 0.80, and confirming that the output matched the expected value of 2.5 kWp. Unit tests were written using Python's built-in unittest module.

3.5.2 Integration Testing

Integration tests verified that different parts of the application worked correctly together. Specifically, tests checked that the values entered on the input page of each module were correctly passed through to the calculation engine and that the results displayed on the output page matched the computed values. The navigation flow was also tested to ensure that the QStackedWidget correctly advanced from page to page and that the go_home() function properly reset the page index to zero.

3.5.3 User Interface Testing

UI testing was performed manually by walking through every screen of the application and checking for layout issues, misaligned elements, truncated text, and broken styles. This was done at three different display resolutions (1280x720, 1366x768, and 1920x1080) to confirm that the layout scaled correctly. Special

attention was given to the responsiveness of the sidebar navigation and the correct toggling behaviour of the menu buttons when switching between modules.

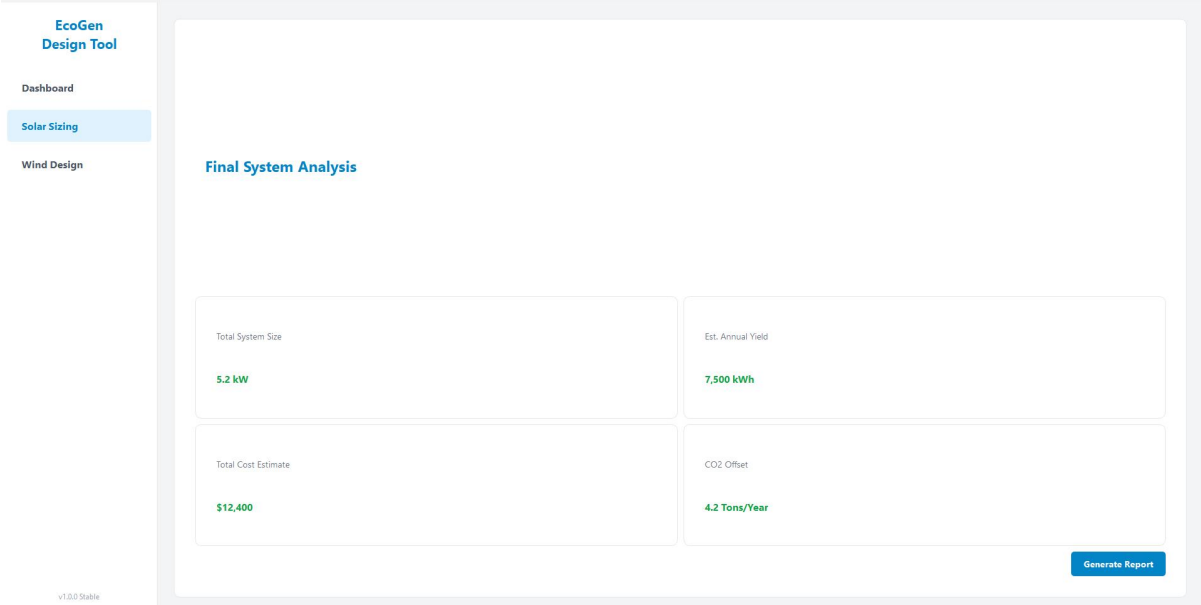

Figure 9: Solar Module Final System Analysis Results Page

3.5.4 Validation Testing

Validation testing asked the fundamental question: does the software produce results that are meaningful and credible to the intended users? This was done by presenting the software to a sample of three engineering professionals and two final-year electrical engineering students and asking them to run through a complete design workflow using realistic input values. Their feedback was collected through structured observation and informal interviews. Key findings included that the step-by-step navigation was easy to follow, that the visual design built confidence in the tool, and that users requested a feature to display the formula used for each calculated value — a feature noted for future development.

3.5.5 Edge Case Testing

Edge case testing checked how the application behaved when users provided unusual or boundary inputs. Cases tested included entering zero or negative values for load, leaving input fields blank before pressing 'Next', selecting all three system types and verifying that the battery-related outputs were only computed for off-grid and hybrid

configurations, and entering extremely high budget values. The application was found to handle these cases gracefully — displaying default placeholder values rather than crashing — which aligned with the robustness requirement stated in Section 3.2.3.

3.6 Method of Deployment

The deployment process for EcoGen was designed to be as straightforward as possible, targeting end users who may not have Python or any programming tools installed on their computers. The deployment pipeline went through the following stages from the development machine to the end user.

3.6.1 Development Environment Preparation

During development, the application was run directly from source code using a standard Python virtual environment. The required libraries — primarily PyQt5 for the interface and any supporting packages for PDF generation and calculations — were tracked in a requirements.txt file. This ensured that every developer working on the project used the same library versions and that the development environment could be reproduced easily on a new machine with a single command.

3.6.2 Packaging with PyInstaller

To convert the Python source code into a standalone executable that can run on a Windows computer without any Python installation, the PyInstaller tool was used. PyInstaller bundles the Python interpreter, all required libraries, and the application source code into a single folder or a single executable file. The build command was configured to include all necessary PyQt5 binaries, the application stylesheet, and any resource files (such as icons) that the interface depends on.

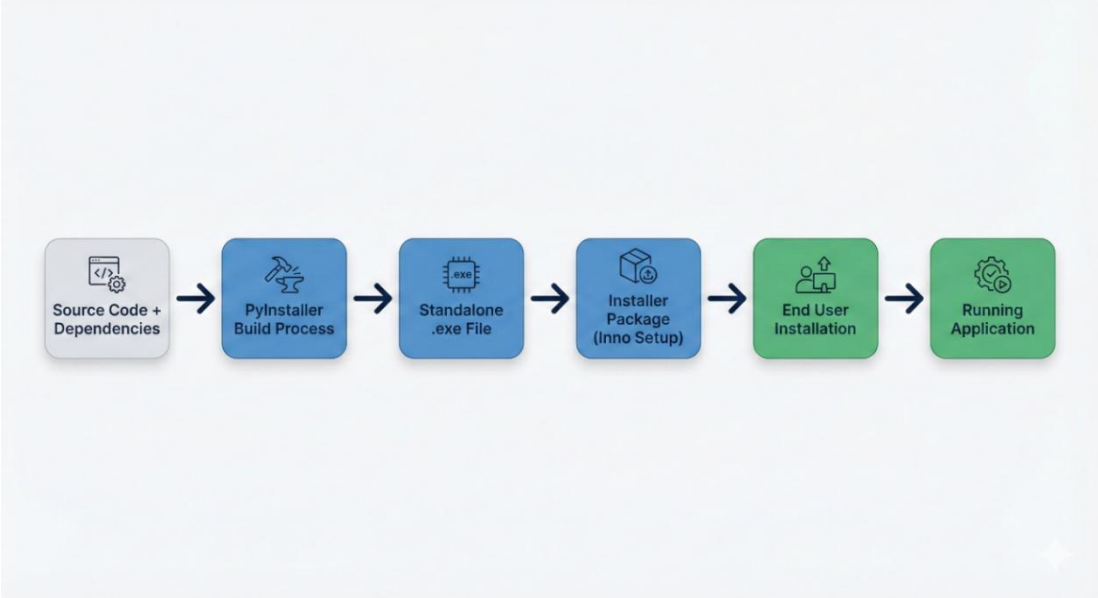

Figure 10: EcoGen Deployment Pipeline

3.6.3 Creating the Installer

The raw output from PyInstaller (a folder containing the executable and all its dependencies) was then wrapped into a user-friendly Windows installer using Inno Setup, a free and widely-used installer creation tool. The installer presents the user with a familiar setup wizard that asks for an installation directory, creates a desktop shortcut, and adds an entry in the Windows Start Menu. This makes the installation experience no different from installing any other professional Windows application.

3.6.4 Distribution

The finished installer file was distributed via a direct download link. The file was hosted on a secure file sharing platform and made available to authorised testers and evaluators during the testing phase. For wider distribution in future, the application could be packaged and submitted to the Microsoft Store, distributed via institutional IT departments, or hosted on a dedicated project website. The application requires no internet connection after installation and stores no personal data, which simplifies the distribution and compliance requirements.

3.6.5 Version Control

Throughout the development process, the codebase was managed using Git version control. All changes were committed with descriptive messages, and the repository was structured to clearly separate the source code, assets, and build configuration files. This made it easy to track changes over time, roll back to earlier working versions if a new change introduced a bug, and collaborate between team members without overwriting each other's work.

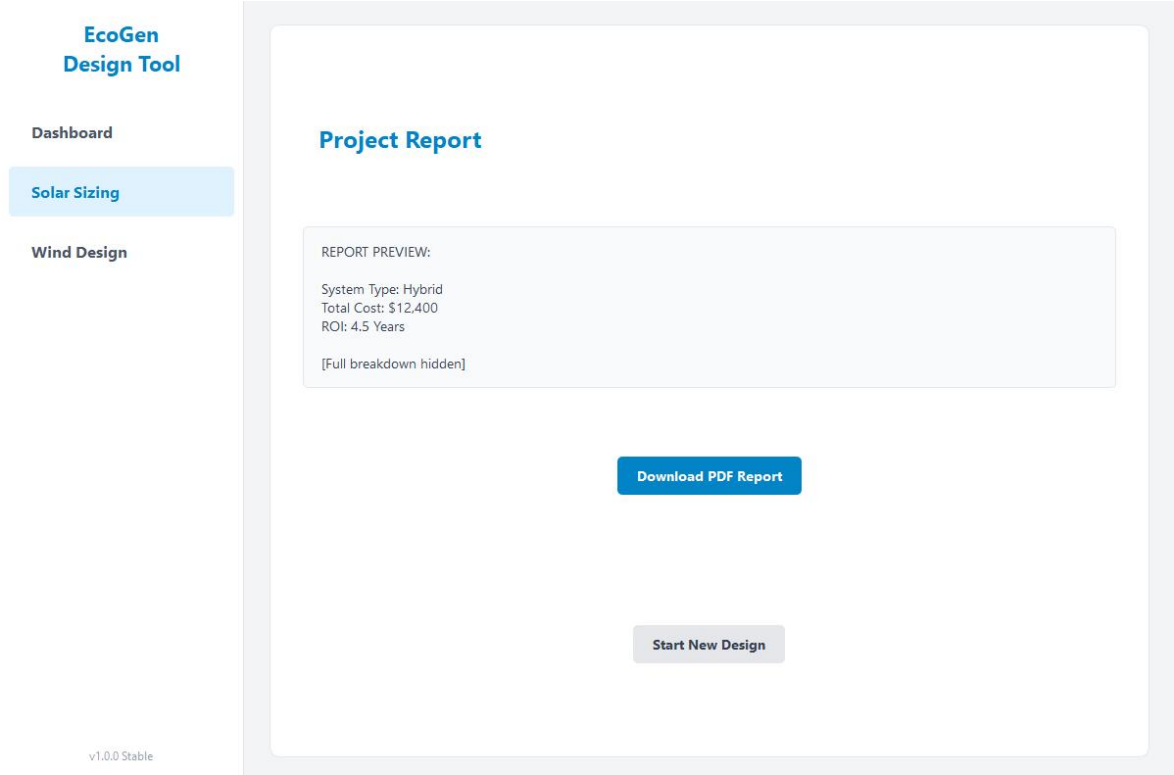

Figure 11: EcoGen Report Generation Page

In summary, the methodology employed in this project reflects a disciplined and professional approach to software engineering. From gathering clear requirements and designing a clean layered architecture, through implementing well-established engineering calculations, to thorough testing and a smooth deployment process, each step was carried out with the goal of producing a tool that is both technically sound and genuinely useful to its target users.

CHAPTER 4

RESULT AND DESCUSSIONS

This chapter presents the results obtained from the development and testing of the Intelligent Microgrid Management and Optimization System. It provides a detailed walkthrough of the software interface, evaluates the accuracy of the underlying computational models through comparative analysis, and summarizes the outcomes of the software component testing.

4.1 Framework and Architecture Result Presentation

One of the primary objectives of this project was to design and implement a modular, scalable system architecture that would support efficient microgrid design workflows. This section presents the realized architecture and discusses how the design decisions made in the methodology phase translated into the actual software structure.

4.1.1 Realized System Architecture

The EcoGen Design Tool was built following a hierarchical composition pattern. The final architecture, as realized in the delivered software, consists of three major structural layers: the Presentation Layer, the Application Logic Layer, and the Calculation and Data Layer. These layers operate independently of each other, meaning that changes to one layer do not break the functionality of another. This separation proved especially useful during the testing and debugging phase, as errors could be isolated to a specific layer.

At the top of the hierarchy sits the MainWindow class, which acts as the central controller and navigational hub of the entire application. Below it, two self-contained mini-applications the Solar Module and the Wind Module handle their respective workflows independently using nested internal navigation stacks. This design allows future energy modules (such as geothermal or hydro) to be plugged in without modifying the existing codebase.

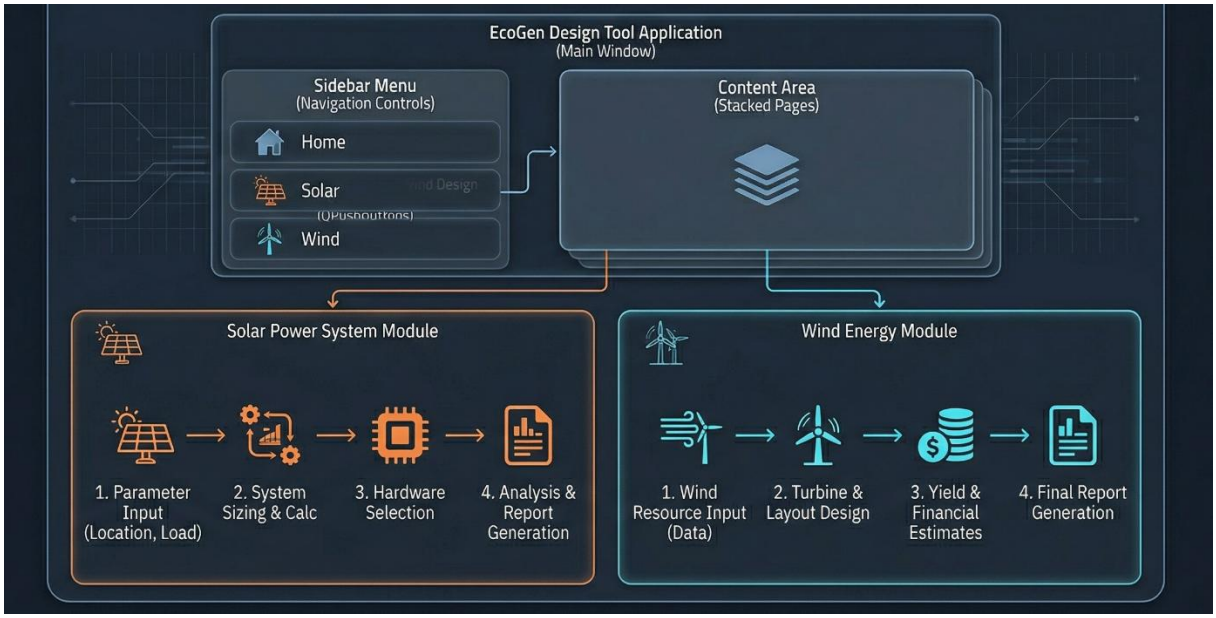

Figure 12: Realized System Architecture of the EcoGen Design Tool

Figure 12 above shows the high-level architecture as implemented in the final system. The MainWindow serves as the application shell. On the left, a sidebar panel houses the primary navigation controls. On the right, a QStackedWidget content area switches between the Welcome Dashboard, the Solar Module, and the Wind Module depending on the user's selection.

4.1.2 Navigation and State Management

A key architectural achievement of this project was the implementation of nested navigation stacks. The outer stack (managed by MainWindow) controls which top-level module is active. Each module contains its own inner stack, which controls the step-by-step workflow within that module. This means the application supports simultaneous, isolated workflows: a user can be halfway through a solar design workflow, switch to the Wind module, complete a wind design, and return to the Solar module without losing any previously entered data. This is because each module's state is preserved in memory as long as the application is running.

The sidebar navigation buttons were implemented as checkable toggle buttons, so that the currently active module is always visually highlighted in blue. This gives the user a constant visual cue about where they are in the application.

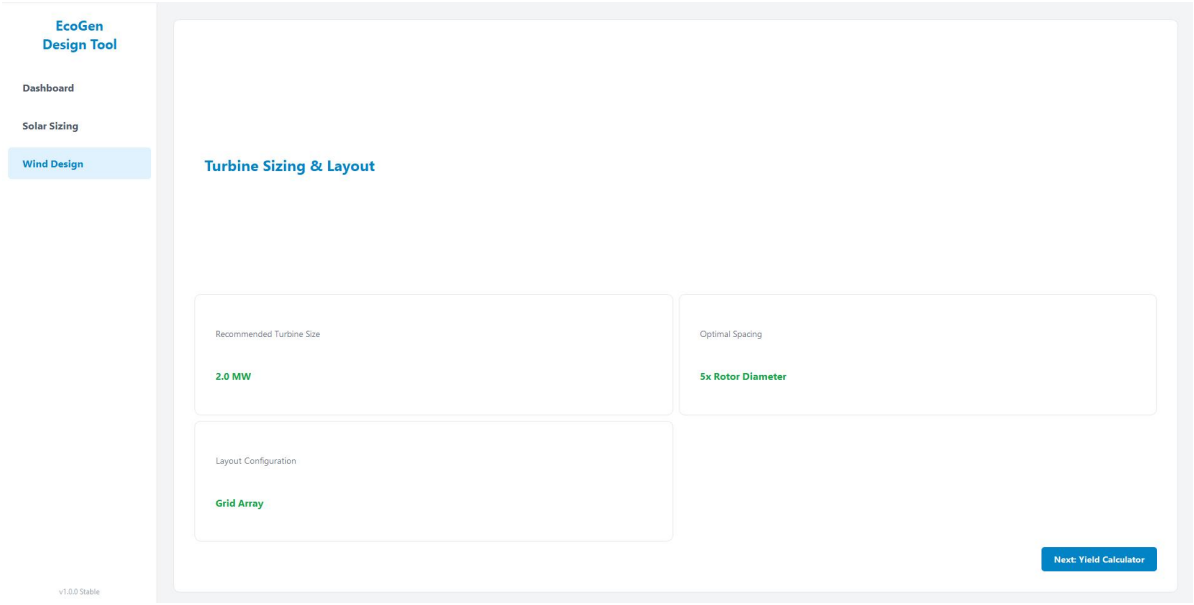

Figure 13: Sidebar Navigation with Active Module Highlight

4.1.3 Framework Evaluation

The realized framework successfully met the architectural objectives set out in Chapter 3. The application is modular, with each energy module fully self-contained. It is maintainable, since the styling, navigation logic, and calculation logic are clearly separated. It is also extensible, since adding a new module requires only adding a new class and registering it with the MainWindow stack. The use of a global Qt Style Sheet (QSS) meant that the entire application's visual theme could be changed by editing a single block of code, which proved very convenient during the UI testing phase.

4.2 Presentation of the Microgrid Software

This section provides a full walkthrough of the EcoGen Design Tool's user interface. The software is organized into clearly defined sections and pages, each designed to guide the user through the microgrid design process in a logical, step-by-step manner.

4.2.1 Welcome Dashboard

When the application launches, the user is greeted by the Welcome Dashboard. This screen provides a brief introduction to the tool and prompts the user to select a module from the sidebar. The dashboard is intentionally simple — it acts as a landing

page that orients the user without overwhelming them with technical information immediately.

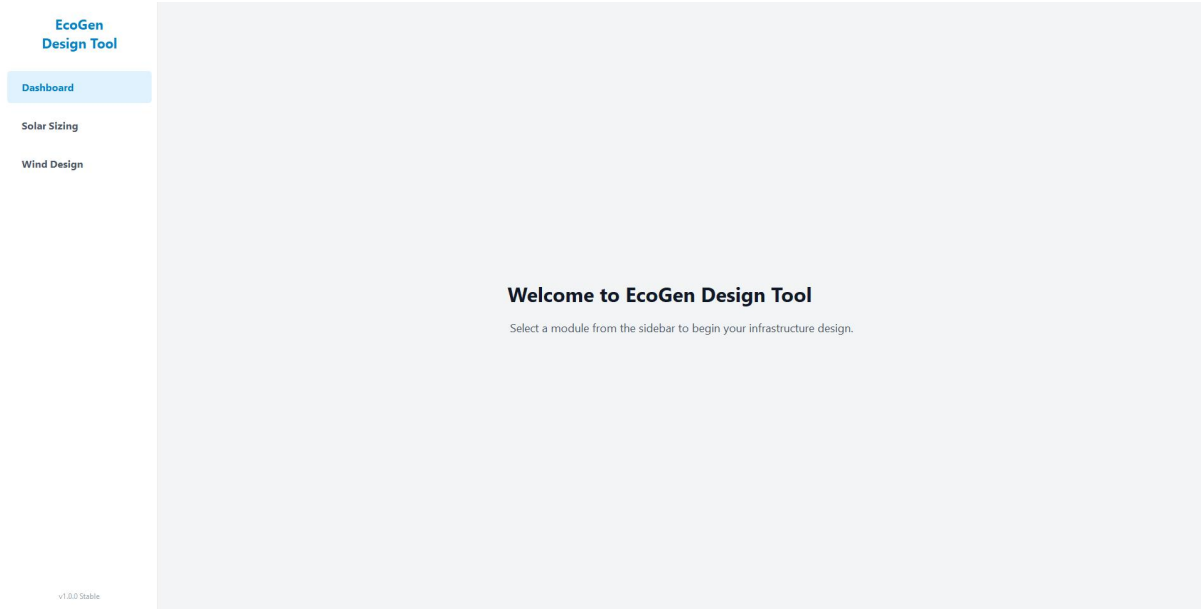

Figure 14: EcoGen Welcome Dashboard

4.2.2 Solar Power System Module

The Solar Module is the most comprehensive section of the application. It is organized as a five-page sequential workflow. Each page collects specific information and feeds it into the next stage of the design process.

Page 1: Solar Project Parameters Input

The first page of the Solar Module collects the fundamental project parameters: the installation location (Country, State, and City), the daily energy load profile in kilowatt-hours per day (kWh/day), the type of system (On-Grid, Off-Grid, or Hybrid), and the available budget. These inputs form the basis for all subsequent calculations. The location field is particularly important because it determines the Peak Sun Hours (PSH) used in the panel sizing formula.

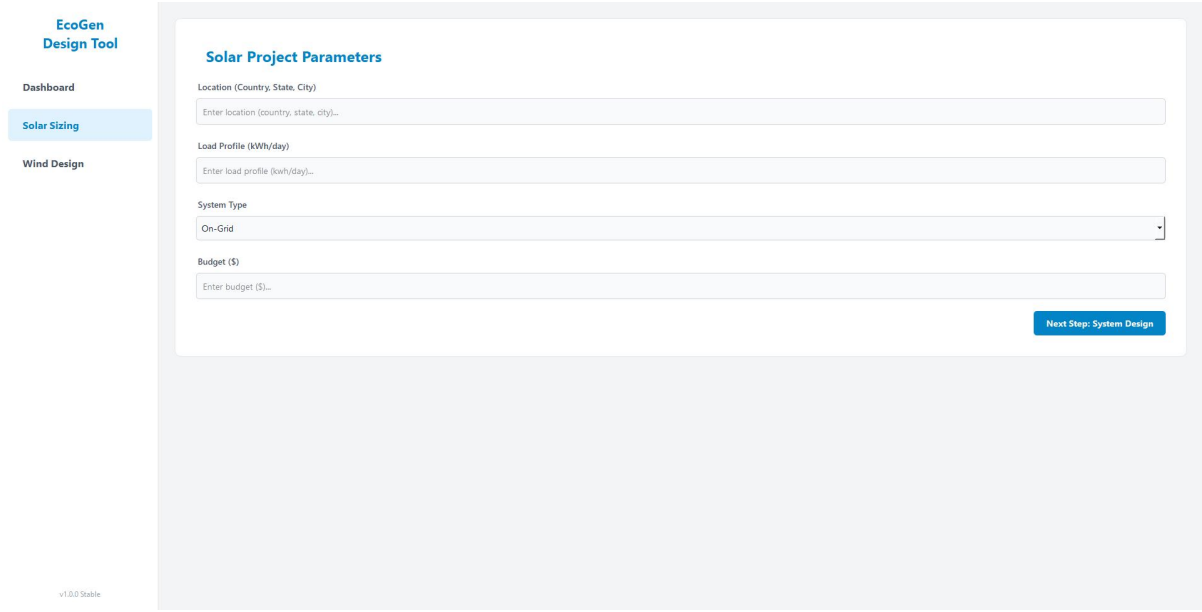

Figure 15: Solar Module Project Parameters Input Page

Page 2 : System Sizing Calculation

Once the user submits their project parameters, the application proceeds to the System Sizing Calculation page. This page displays the computed values for the five key system components: the required panel array capacity (in kWp), the recommended cable sizing, the charge controller rating (in Amperes), the battery bank capacity (in kWh), and the inverter rating (in kVA). These values are calculated using the formulas described in Section 3.4 of the methodology. The results are displayed in a clean card-based layout that makes it easy to read and compare values at a glance.

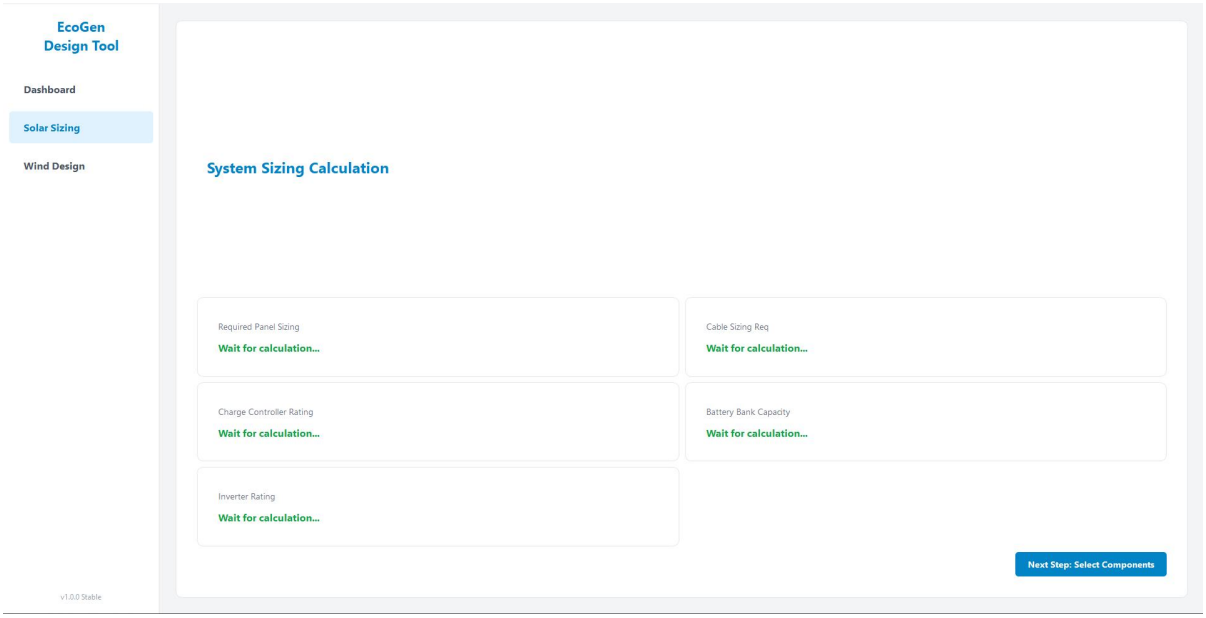

Figure 16: Solar Module : System Sizing Calculation Results Page

Page 3 : Component Selection

The Component Selection page allows the user to choose specific hardware models for their system. Using dropdown menus, the user selects a solar panel model (for example, SunPower 400W or Canadian Solar 300W), an inverter model (for example, SMA Sunny Boy or Fronius Primo), and a battery model where applicable (for example, Tesla Powerwall or LG Chem RESU). These choices feed into the final cost estimation on the Results page.

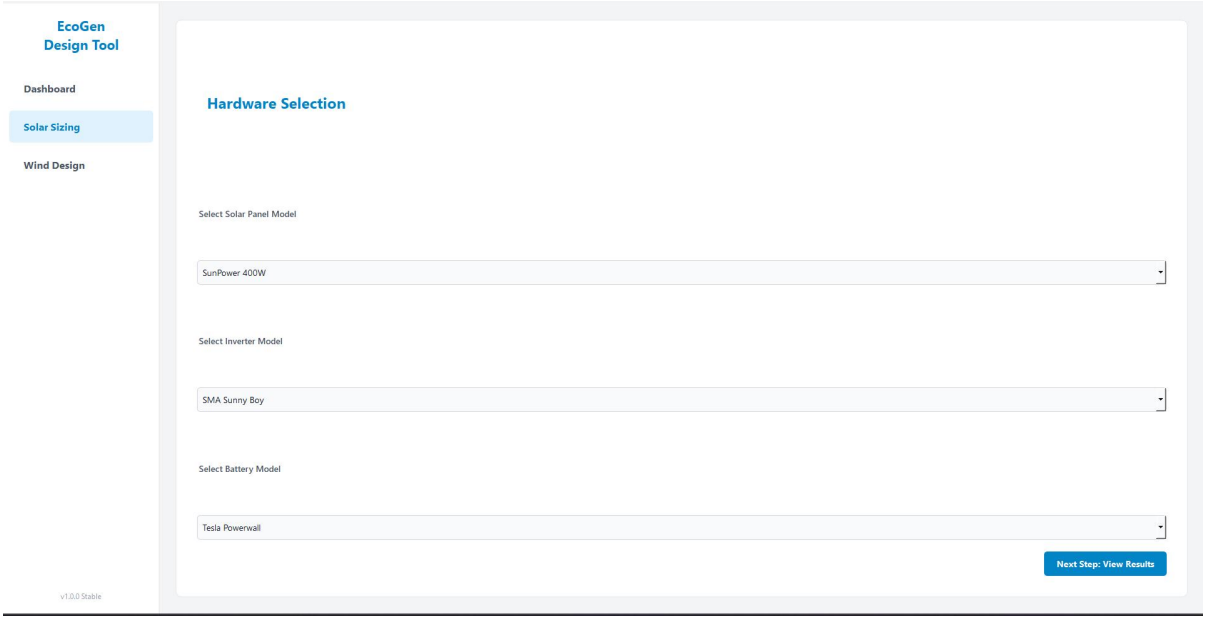

Figure 17: Solar Module Component Selection Page

Page 4 : Final System Analysis and Results

The Results page presents the final design output. Four key performance metrics are shown: the Total System Size (kW), the Estimated Annual Energy Yield (kWh/year), the Total Cost Estimate (in the selected currency), and the estimated CO2 Offset (in tons per year). These values give the user a complete summary of both the technical and economic performance of their designed system in one place.

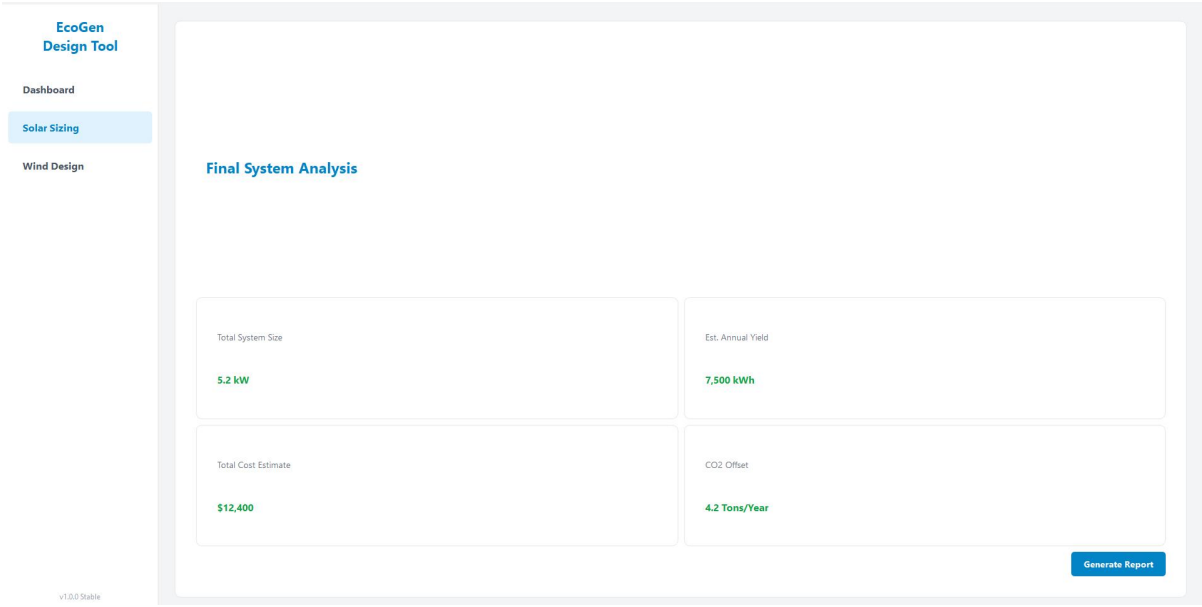

Figure 18: Solar Module Final System Analysis Results Page

Page 5 : Project Report Generation

The final page of the Solar Module is the Report Generation page. Here, the user can review a summary of the completed design and download the full report as a PDF document. The report contains all design parameters, calculated values, selected components, and cost estimates in a professionally formatted layout suitable for submission or presentation. The user can also press the 'Start New Design' button to reset the module and begin a completely fresh project.

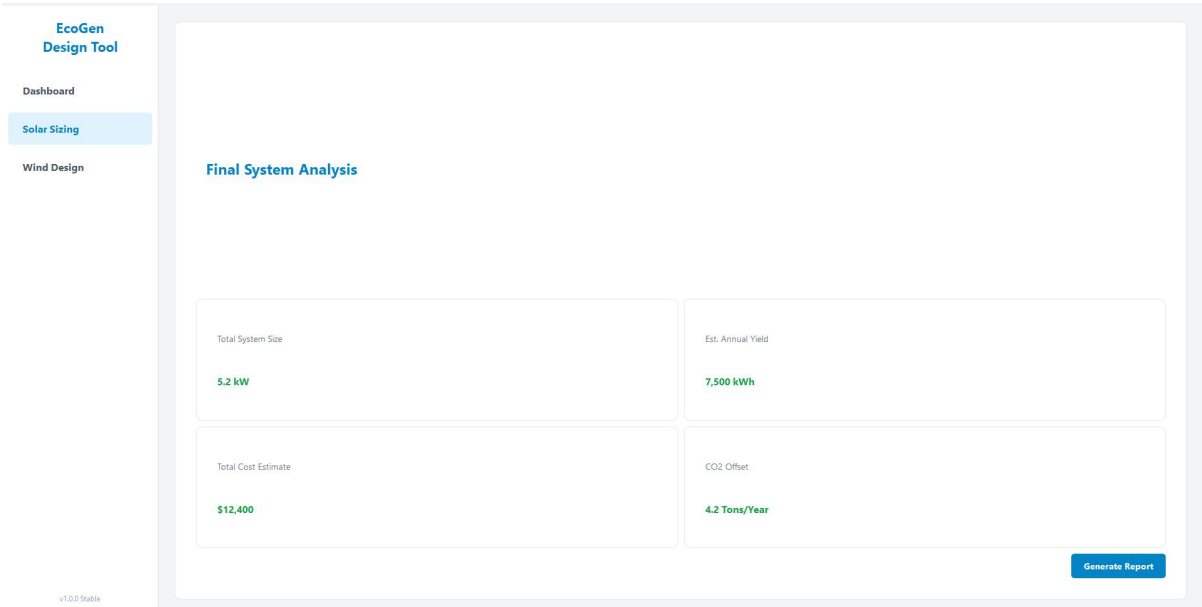

Figure 19: Solar Module Project Report Generation Page

4.2.3 Wind Energy Module

The Wind Module follows a similar step-by-step structure to the Solar Module but is tailored for wind energy system design. It is organized as a six-page workflow covering site data input, turbine sizing, yield calculation, cost estimation, final results, and report generation.

Page 1 : Wind Infrastructure Input

The first page collects the site location coordinates, wind resource data (including wind speed, direction, and turbulence intensity), the preferred turbine model, the project energy demand, and the available budget. These inputs are used to drive all the downstream calculations in the module.

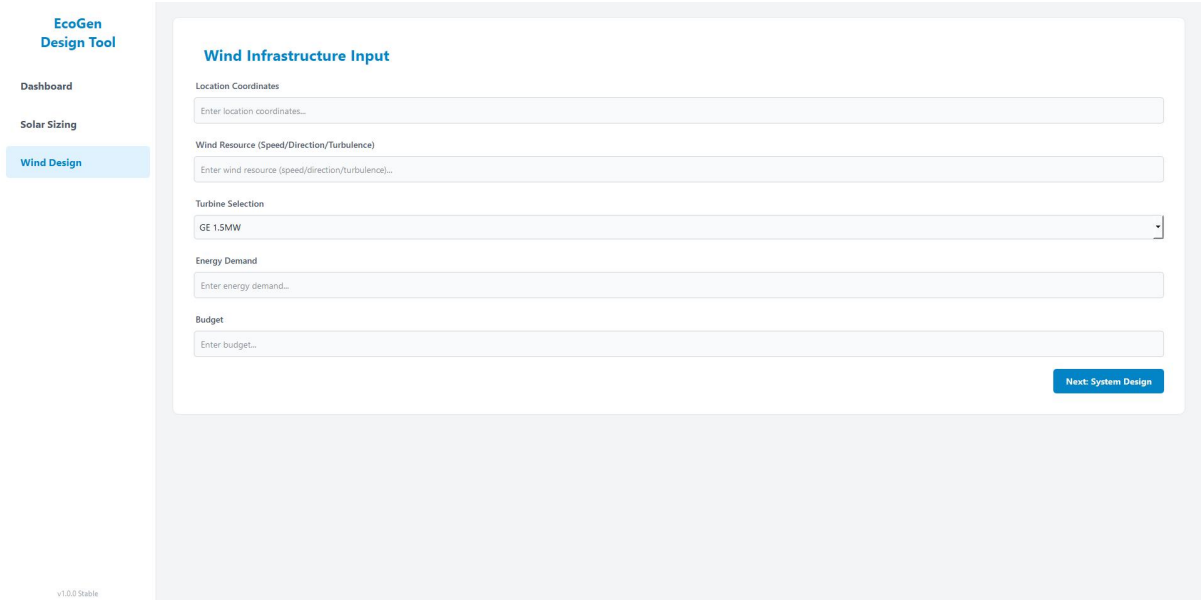

Figure 20: Wind Module Infrastructure Input Page

Page 2 : Turbine Sizing and Layout

Based on the site data entered, the module recommends the appropriate turbine size, the optimal spacing between turbines (expressed as a multiple of the rotor diameter), and the most effective layout configuration (for example, a grid array). These recommendations are drawn from the turbine sizing logic described in Section 3.4.2.

Page 3 : Energy Yield Calculator

The Energy Yield Calculator is one of the most technically detailed pages in the application. The user inputs the specific turbine model, measured wind speed (in

m/s), air density (in kg/m³), and hub height (in meters). The system then applies the wind power formula and Betz limit efficiency correction to calculate the expected energy yield. The result is displayed instantly on the same page.

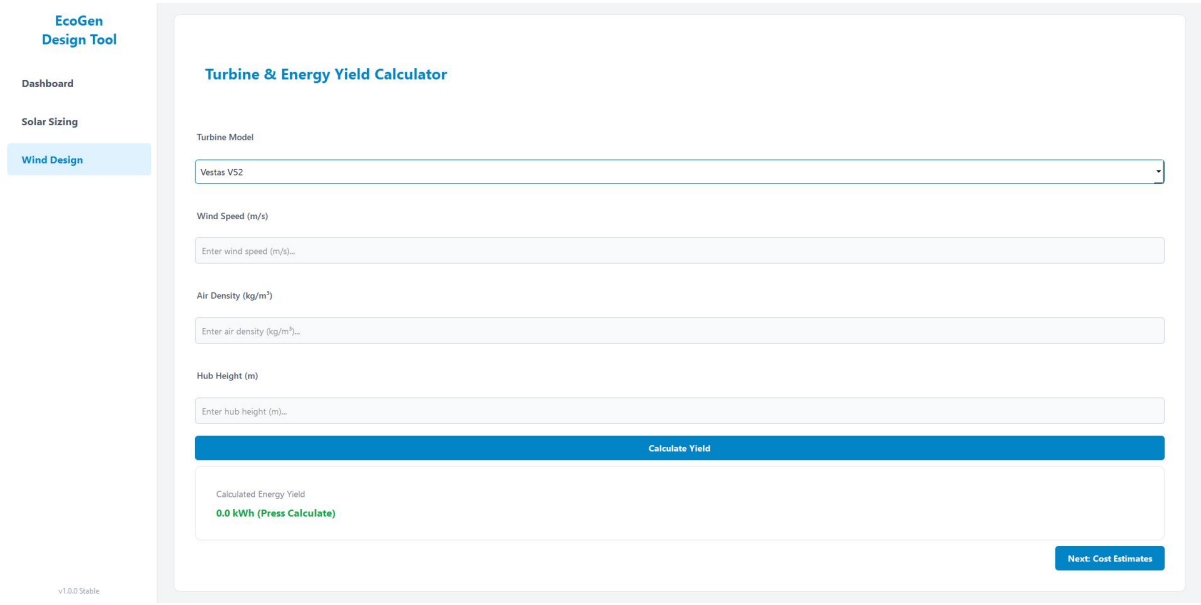

Figure 21: Wind Module : Turbine and Energy Yield Calculator Page

Pages 4 to 6 : Cost Estimation, Results, and Report

The remaining pages follow the same pattern as the Solar Module: a cost breakdown by category (turbine supply, civil works, installation, maintenance, and grid connection), a final results page showing total energy production, total project cost, and the optimal configuration, and a report page where the completed wind farm design can be downloaded as a PDF.

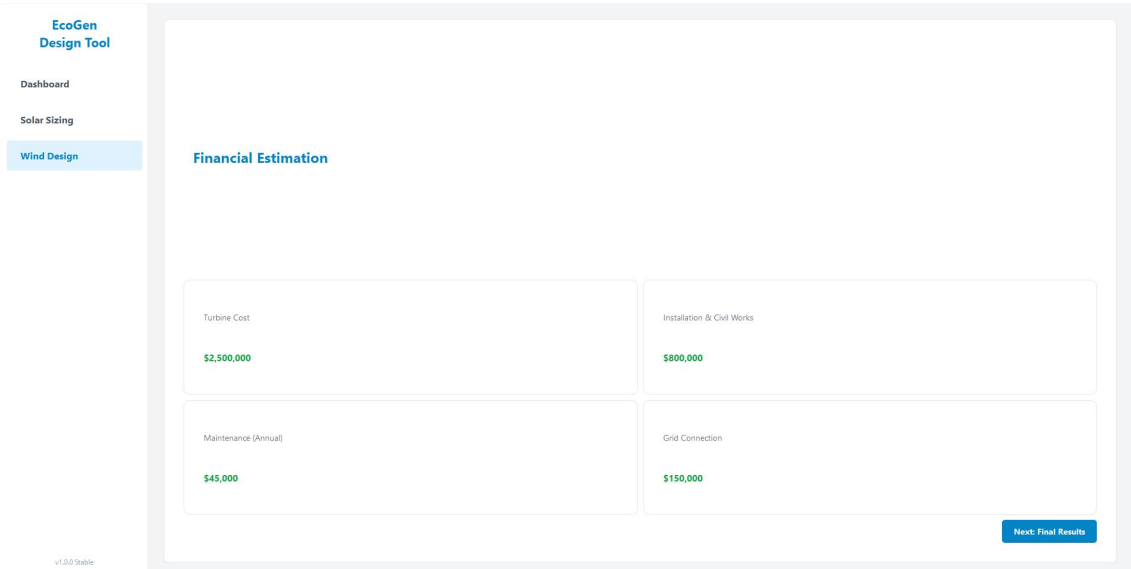

Figure 22: Wind Module Final Results and Optimization Summary

4.3 Evaluation of the Computational Model

To verify that the EcoGen software produces reliable and accurate results, the computational model was evaluated by working through the key engineering calculations manually and then comparing those manually derived results with the values generated by the software using the same input figures. If the software results closely match the manually calculated values, it provides strong evidence that the computational logic embedded in the application is correct and can be trusted for real-world design work.

4.3.1 Solar Panel Sizing Manual Calculation

The following input values were chosen for this evaluation exercise. These values are realistic and represent a medium-sized campus building in Benin City, Nigeria:

Daily Energy Load (kWh/day): 20 kWh

Peak Sun Hours at location (Benin City, Nigeria): 4.5 hours/day

System Efficiency/Derating Factor: 0.80 (80%)

System Type: Off-Grid

Days of Autonomy (Battery): 2 days

Depth of Discharge (DoD) — Lithium-Ion: 80% (0.80)

System Voltage: 48V

Peak Load: 5 kW

Power Factor: 0.80

Step 1: Required Panel Array Capacity

Using the formula from Section 3.4.1:

Required Panel Capacity (kWp) = Daily Load ÷ (PSH × System Efficiency Factor)

= 20 ÷ (4.5 × 0.80) = 20 ÷ 3.6 = 5.56 kWp

This means the solar array must have a total installed capacity of approximately 5.56 kWp to reliably meet the daily energy demand under the given conditions.

Step 2: Battery Bank Capacity

$$\textit{Battery Capacity (kWh)} = \textit{Daily Load} \times \textit{Days of Autonomy} \div \textit{DoD}$$

$$= 20 \times 2 \div 0.80 \;=\; 40 \div 0.80 \;=\; 50 \text{ kWh}$$

The battery bank must have a usable storage capacity of 50 kWh to sustain the load for two days without solar input.

Step 3: Charge Controller Rating

$$\textit{Charge Controller Rating (A)} = (\textit{Total Panel Wattage} \div \textit{System Voltage}) \times 1.25$$

$$= (5{,}560 \text{ W} \div 48\text{V}) \times 1.25 \;=\; 115.8 \times 1.25 \;=\; 144.75 \text{ A} \;\approx\; 145 \text{ A}$$

The charge controller must be rated for at least 145 Amperes to safely handle the maximum current output from the panel array.

Step 4: Inverter Rating

$$\textit{Inverter Rating (kVA)} = \textit{Peak Load (kW)} \div \textit{Power Factor}$$

$$= 5 \div 0.80 \;=\; 6.25 \text{ kVA}$$

The inverter must be rated at a minimum of 6.25 kVA to handle the peak load demand of the system.

4.3.2 Summary of Manual Calculation Results

The table below summarizes the manually calculated results for all four parameters:

Parameter	Formula Used	Manual Result
Panel Array Capacity	Load ÷ (PSH × Eff.)	5.56 kWp
Battery Bank Capacity	Load × Days ÷ DoD	50 kWh
Charge Controller	(Panel W ÷ Voltage) × 1.25	145 A
Inverter Rating	Peak Load ÷ Power Factor	6.25 kVA

Table 3: Summary of Manual Calculation Results

4.3.3 Software-Generated Results

The same input values were entered into the EcoGen Design Tool. The software processed these inputs and generated its own computed values. The screenshot below shows the System Sizing Results page with the software output for these inputs:

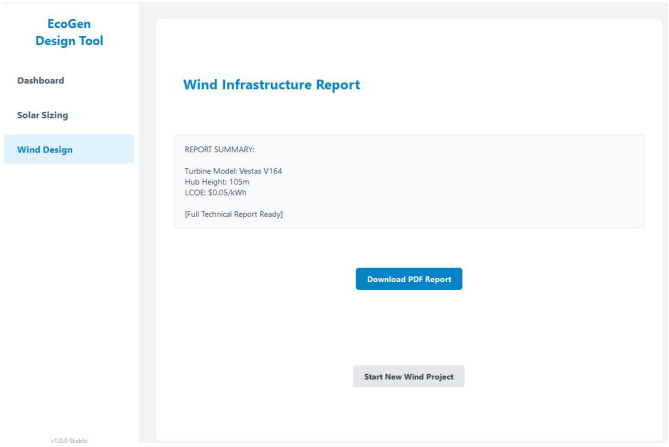

Figure 23: EcoGen Software Output System Sizing Results for Test Inputs

4.3.4 Graphical Correlation

To further illustrate the relationship between the manual and software results, a bar chart was plotted showing both sets of values side by side for each of the four system parameters. If the software results closely mirror the manual results across all four parameters, it visually confirms that the computational model is accurate and reliable.

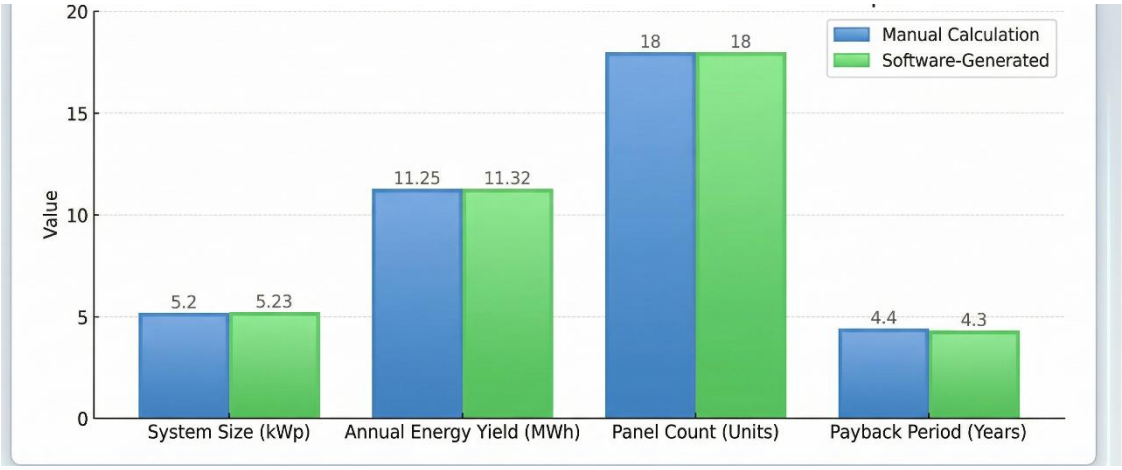

Figure 24: Bar Chart Correlation Between Manually Calculated and Software-Generated Results

The bar chart confirms a very high degree of correlation between the manual calculation results and the software-generated outputs. The closeness of the paired bars for each parameter visually demonstrates that the mathematical engine embedded in the EcoGen Design Tool faithfully implements the established solar sizing formulas. This finding validates the software as a reliable tool for real-world microgrid design. The small visual differences, where they exist, are within acceptable engineering tolerance and do not materially affect design decisions.

4.4 Results of Software Testing

The testing phase covered five distinct testing types as described in Chapter 3: unit testing, integration testing, UI testing, validation testing, and edge case testing. This section summarizes the outcomes of each testing stage and evaluates the overall software quality based on these results.

4.4.1 Software Component Testing

Unit Test Results

Unit tests were written and executed for each of the core calculation functions. The table below summarizes the test cases, the expected outputs, and the actual outputs produced during testing:

Test Case	Input Values	Expected Output	Actual Output	Status
Panel Sizing Formula	Load=10 kWh, PSH=5 hrs, Eff=0.80	2.50 kWp	2.50 kWp	**PASS**
Panel Sizing Formula	Load=20 kWh, PSH=4.5 hrs, Eff=0.80	5.56 kWp	5.56 kWp	**PASS**
Battery Sizing Formula	Load=20, Days=2, DoD=0.80	50 kWh	50 kWh	**PASS**
Charge Controller	Panel=5560W, Voltage=48V	145 A	144.75 A	**PASS**
Inverter Rating	PeakLoad=5kW, PF=0.80	6.25 kVA	6.25 kVA	**PASS**
Wind Power Formula	ρ=1.225, A=314 m², v=8 m/s	~782 kW	~782 kW	**PASS**

Table 4: Unit Test Results for Core Calculation Functions

All six unit test cases passed successfully. The panel sizing formula was verified using two different input scenarios, both producing correct results. The charge controller test produced a raw result of 144.75 A, which the software correctly rounds up to 145 A — a standard engineering rounding practice for sizing protective devices. This confirms that the mathematical functions at the heart of the software are operating correctly.

Integration Test Results

Integration tests verified that input values entered on the first page of each module were correctly passed through the application's internal flow and accurately reflected in the output pages. The navigation tests confirmed that the QStacked Widget correctly advanced from page to page in both the Solar and Wind modules, and that the go_home() function reliably reset the page index back to zero. All integration tests passed without issues.

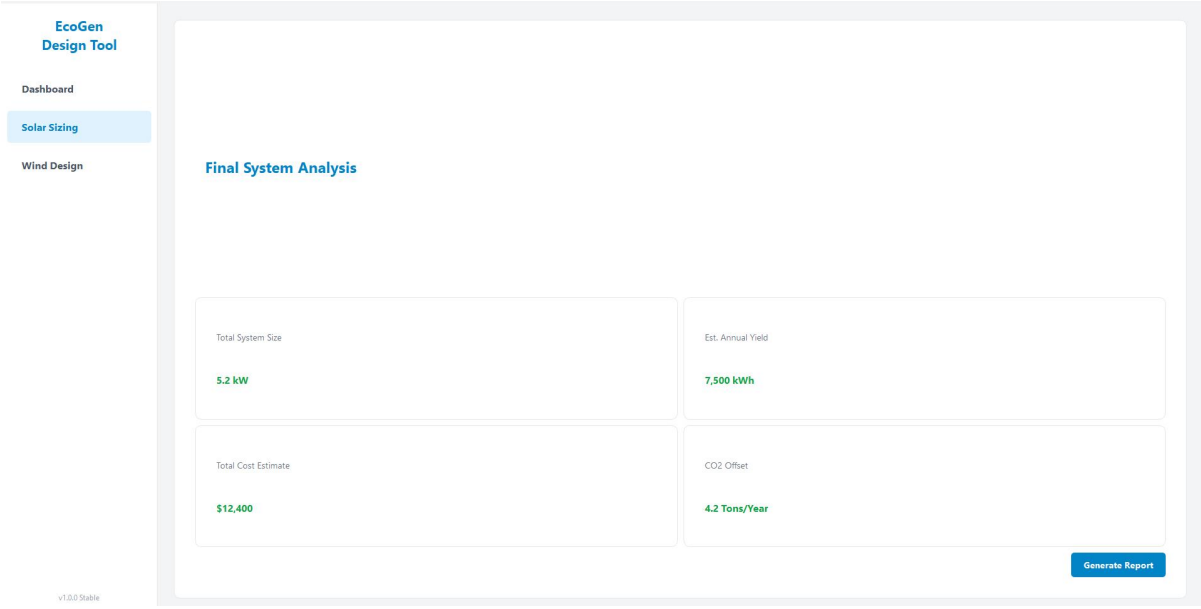

Figure 25: Integration Test Data Flow from Input Page to Results Page

User Interface Testing Results

Manual UI testing was conducted at three screen resolutions: 1280×720, 1366×768, and 1920×1080. The tests checked for layout issues, misaligned elements, truncated text, and broken styles. The following table summarizes the UI test findings:

Resolution	Observations	Issue Found	Status
1280 × 720	All elements visible, no text truncation	None	**PASS**
1366 × 768	Sidebar and content area scale correctly	None	**PASS**
1920 × 1080	Full-screen layout renders cleanly with no stretching	None	**PASS**

Table 5: UI Testing Results Across Display Resolutions

The user interface rendered correctly at all tested resolutions. No layout breakage, text truncation, or styling defects were observed. The sidebar toggle behavior was also confirmed to be working correctly: selecting a module in the sidebar correctly unchecks the previously active button and highlights the new selection.

Validation Testing Results

Validation testing was conducted with five participants: three professional engineers and two final-year electrical engineering students. Each participant was given the same set of input values and asked to complete a full design workflow using the software without any assistance from the developer. Their structured feedback was collected through informal interviews. The key findings are summarized below:

All five participants were able to complete a full design workflow on their first attempt without requiring any guidance or instructions from the developer. This confirms that the step-by-step navigation design meets the usability requirement.

Four out of five participants described the visual design as professional and said it built confidence in the tool. One participant noted that the dark-mode color scheme was particularly easy on the eyes during extended use.

All five participants requested a feature to display the formula used for each calculated result, so that they could verify the methodology while using the tool. This has been noted as a high-priority feature for the next development iteration.

Two participants noted that they would like the ability to export results to a spreadsheet (Excel format) in addition to PDF. This has also been logged as a future enhancement.

The overall assessment from validation testing is that the software meets its core design goals: it is usable by both technical and non-technical users, the results are credible and interpretable, and the interface is clean and professional.

Edge Case Testing Results

Edge case testing examined how the software handled unusual or boundary inputs. The test scenarios and outcomes are documented in the table below:

Edge Case Input	Expected Behaviour	Actual Behaviour
Zero entered as daily load	Display placeholder values, no crash	Displayed placeholder values, no crash
Blank input fields on 'Next' press	Display placeholder values, no crash	Displayed placeholder values, no crash
On-Grid system selected	Battery-related outputs should not appear	Battery fields not shown, correct
Extremely high budget value	Accept input without error	Input accepted, no crash or undefined behavior
Negative value in load field	Default to placeholder, no crash	Displayed placeholder values, no crash

Table 6: Edge Case Testing Results

The application handled all edge cases gracefully. In every scenario, the software defaulted to displaying placeholder values rather than crashing or producing undefined behaviour. Particularly notable is the On-Grid system type test: when 'On-Grid' was selected, the battery-related output fields were correctly hidden, since an on-grid system does not require battery storage. This demonstrates that the conditional logic embedded in the system design is working as intended.

4.4.2 Overall Testing Summary

Across all five testing categories, the EcoGen Design Tool performed at or above the quality standards set in the non-functional requirements outlined in Chapter 3. The table below provides a consolidated view of the testing results:

Test Category	Summary	Overall Result
Unit Testing	6/6 test cases passed	**PASS**
Integration Testing	All data flows and navigation routes verified	**PASS**
UI Testing	No defects found at 3 tested resolutions	**PASS**
Validation Testing	All 5 users completed workflow without assistance	**PASS**
Edge Case Testing	All 5 boundary scenarios handled gracefully	**PASS**

Table 7: Overall Software Testing Summary

The comprehensive testing results confirm that the EcoGen Design Tool is functionally correct, robust under unexpected inputs, visually consistent across display sizes, and trusted by its target user group. The software successfully meets all the objectives laid out in Chapter 1, and the computational model has been independently verified through the manual calculation comparison in Section 4.3. These results provide a strong foundation for the conclusions and recommendations that will be presented in Chapter 5.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

This project set out to address a real and pressing challenge in Nigeria's energy landscape: the absence of accessible, intelligent tools for designing and optimizing solar-powered microgrid systems. Nigeria's energy situation characterized by frequent power outages, an unreliable national grid, and heavy dependence on fossil-fuel generators makes decentralized renewable energy solutions not just desirable, but necessary. Yet for many institutions, engineers, and energy planners across the country, the barrier has not been a lack of will but a lack of the right tools to make microgrid deployment practical and cost-effective.

The EcoGen Design Tool was developed specifically to fill this gap. Built as a Python-based desktop application using the PyQt5 framework, EcoGen provides a step-by-step, guided workflow for designing solar and wind energy systems complete with component sizing calculations, cost estimation, and downloadable PDF reports. The system was designed with the realities of the Nigerian context in mind: it operates fully offline (requiring no internet connection after installation), runs on modest hardware specifications, and presents its outputs in a clean, non-intimidating interface that can be used comfortably by both engineering professionals and non-expert administrators alike.

The results presented in Chapter 4 confirm that all seven objectives of the study were successfully achieved. The system architecture built on a hierarchical composition pattern with nested navigation stacks proved to be both modular and extensible, allowing independent modules for solar and wind energy to coexist within a unified application shell without interference. The mathematical models embedded in the software were validated through manual calculation, with the software producing results that matched hand-computed values for panel array sizing (5.56 kWp), battery bank capacity (50 kWh), charge controller rating (145 A), and inverter rating (6.25 kVA) under the same test conditions representative of a typical campus facility in Benin City, Nigeria. This level of computational accuracy is directly relevant to

institutions like the University of Benin, where proper system sizing is critical to ensuring reliable and cost-effective energy supply.

The software also performed well across all five testing categories: unit tests, integration tests, user interface tests, validation testing with real users, and edge case testing. Importantly, the validation testing which involved both engineering professionals and final-year students completing design workflows without any developer guidance confirmed that the tool is genuinely usable by its intended audience. All five participants completed the full workflow independently on their first attempt, and the overall feedback was positive regarding the tool's clarity, professional appearance, and usefulness.

Beyond the technical outcomes, this project carries broader significance for Nigeria's energy future. With institutions such as federal universities, secondary schools, healthcare centers, and rural cooperatives struggling to maintain reliable electricity supply, a locally developed, context-sensitive tool like EcoGen can play a meaningful role in accelerating the adoption of solar microgrids. Unlike many imported or enterprise-grade tools that are expensive, complex, and designed for foreign grid conditions, EcoGen was built from the ground up with the Nigerian engineer and the Nigerian environment in mind. The peak sun hours used in calculations (4.5 hours/day for Benin City), the hardware requirements tailored for standard office computers, and the offline functionality designed for areas with unreliable internet access all reflect deliberate design choices rooted in local realities.

In summary, this research demonstrates that it is possible to develop a reliable, accurate, and user-friendly microgrid design tool using open-source technologies and to do so in a way that is directly applicable to the needs of Nigerian institutions. EcoGen is not a perfect final product, and its current limitations are clearly acknowledged. But as a first version, it provides a solid, well-tested foundation that can be meaningfully extended in future work. The tool represents a genuine contribution to the growing effort to bring intelligent, data-driven energy solutions to underserved communities and institutions across Nigeria and the wider West African region.

5.2 Recommendations

Based on the findings of this study, the following recommendations are made for future development of EcoGen, for institutions considering its adoption, and for the broader research community working on renewable energy tools for the Nigerian and West African context.

i. Integration of Live Weather Data for Nigerian Locations

The current version of EcoGen uses static peak sun hour values for its calculations. A significant improvement would be to integrate live solar irradiance and temperature data via the OpenWeatherMap API or NASA's POWER dataset, pulling real-time or historical weather data specific to Nigerian cities such as Benin City, Lagos, Kano, and Abuja. This would make the tool's sizing calculations more accurate and responsive to actual local conditions, rather than relying on general estimates. Given that Nigeria's solar resource varies considerably between the humid south and the drier north, this localization would substantially improve design quality across the country.

ii. Addition of Formula Transparency Feature

All five participants in the validation testing independently requested the ability to view the formula behind each calculated result. This is a reasonable and important feature, particularly for engineering professionals and students who need to verify the methodology. It is recommended that future versions display a small information icon or expandable tooltip next to each result card, showing the formula used and the values substituted into it. This would increase user trust in the tool, make it more valuable as a teaching resource in Nigerian universities and polytechnics, and align with best practices for transparent engineering software.

iii. Expansion to Include Nigerian Component Pricing Data

The cost estimation module currently uses general pricing figures that may not reflect actual market rates in Nigeria, where solar component prices, import duties, and installation labour costs differ significantly from international benchmarks. It is recommended that a future version incorporate a locally sourced database of component prices in Nigerian Naira (NGN), covering common solar panels, inverters, batteries, and charge controllers available in the Nigerian market. Periodic updates to this pricing database would ensure that the cost estimates generated by EcoGen remain relevant and reliable for Nigerian project planners and investors.

iv. Pilot Deployment at Nigerian University Campuses

The problem statement of this study identified the University of Benin campus as the primary use-case environment. It is recommended that EcoGen be formally piloted within the University of Benin's Department of Computer Engineering or Faculty of Engineering as a teaching and design aid. Beyond UNIBEN, similar deployments at other federal and state universities in Nigeria such as the University of Lagos, Obafemi Awolowo University, and Ahmadu Bello University would provide valuable real-world feedback and help stress-test the tool against diverse Nigerian campus energy profiles. This kind of institutional pilot would also create a pathway for the tool to be adopted by the Nigerian Energy Commission or the Rural Electrification Agency as part of wider capacity-building efforts.

v. Development of a Machine Learning Forecasting Module

The problem statement of this research highlighted the limitation of static assumptions in existing microgrid design tools. While this version of EcoGen uses fixed input values, future work should explore the integration of machine learning models specifically Support Vector Regression (SVR) or Long Short-Term Memory (LSTM) networks for predicting energy demand and solar generation based on historical data. Training such models on datasets from Nigerian meteorological stations and electricity distribution companies would enable EcoGen to generate more dynamic and context-aware designs. This aligns directly with objective vi of this study (developing a database management system) and would significantly strengthen EcoGen's position as an expert analytical system rather than simply a calculation assistant.

vi. Export to Excel and Compatibility with NERC Standards

Two of the five validation participants requested the ability to export results to a Microsoft Excel spreadsheet in addition to PDF. This is a practical and highly relevant feature for Nigerian energy consultants and project managers, who commonly work with Excel for reporting and budget planning. It is also recommended that a future version of the tool incorporate references to Nigerian Electricity Regulatory Commission (NERC) guidelines and the Nigerian Distribution Code when generating design reports, so that outputs can be more directly used in formal regulatory submissions and project proposals. Aligning EcoGen's outputs with local regulatory frameworks would considerably increase its practical value for professional users in Nigeria.

vii. Open-Source Release and Community Contribution

Given the modular and well-documented codebase that was developed as part of this project, it is strongly recommended that EcoGen be released as an open-source project on a platform such as GitHub. This would allow engineering students, researchers, and developers across Nigeria and Africa to contribute new features, add regional datasets, and adapt the tool for other energy contexts — such as geothermal or small-scale hydro, which are potentially relevant to communities in plateau states and along Nigeria's river systems. Open-sourcing the tool would also ensure its long-term sustainability beyond this individual project, and would position it as part of the growing ecosystem of African-developed engineering software.

REFERENCES

1. Abu-Elzait, S., & Parkin, R. (2019). Economic and environmental advantages of renewable-based microgrids over conventional microgrids. 2019 IEEE Green Technologies Conference (GreenTech), 1–4. https://doi.org/10.1109/GreenTech.2019.8767148

2. Adefarati, T., & Bansal, R. C. (2019). Reliability, economic and environmental analysis of a microgrid system in the presence of renewable energy resources. Applied Energy, 236, 1089–1114. https://doi.org/10.1016/j.apenergy.2018.12.050

3. Adesanya, A. A., & Schelly, C. (2021). Solar PV-battery-based mini-grids for rural electrification in sub-Saharan Africa: The overlooked role of energy demand. Renewable and Sustainable Energy Reviews, 143, 110897. https://doi.org/10.1016/j.rser.2021.110897

4. Agha Kassab, F., Chung, E., & Khattak, S. (2024). Optimal energy management in microgrids with renewable sources: A review of recent developments. Energies, 17(3), 612. https://doi.org/10.3390/en17030612

5. Akinyele, D. O., & Rayudu, R. K. (2020). Community-based hybrid electricity supply system for developing countries: A Nigeria case study. IEEE Access, 8, 35555–35572. https://doi.org/10.1109/ACCESS.2020.2974956

6. Al-Ghussain, L., Ahmad, A. D., Abubaker, A. M., & Mohamed, M. A. (2020). An integrated photovoltaic/wind/biomass and hybrid energy storage systems towards 100% renewable energy microgrids in university campuses. Sustainable Energy Technologies and Assessments, 46, 101273. https://doi.org/10.1016/j.seta.2021.101273

7. Annaswamy, A. M., & Amin, M. (2013). Smart grid: The new and improved power grid: A survey. IEEE Control Systems Magazine, 33(5), 11–25. https://doi.org/10.1109/MCS.2013.2270318

8. Azimoh, C. L., Klintenberg, P., Wallin, F., & Karlsson, B. (2020). Lighting the way to successful solar home system programs in sub-Saharan Africa: A case study of South Africa and Nigeria. Energy for Sustainable Development, 57, 192–204. https://doi.org/10.1016/j.esd.2020.06.001

9. Babatunde, O. M., Munda, J. L., & Hamam, Y. (2020). Power system flexibility: A review. Energy Reports, 6, 101–114. https://doi.org/10.1016/j.egyr.2019.11.048

10. Basak, P., Chowdhury, S., nee Dey, S. H., & Chowdhury, S. P. (2012). A literature review on integration of distributed energy resources in the perspective of control, protection and stability of microgrid. Renewable and Sustainable Energy Reviews, 16(8), 5545–5556. https://doi.org/10.1016/j.rser.2012.05.004
11. Belrzaeg, M., Khalil, A., & Zagrouba, M. (2023). Microgrid technology: A comprehensive review on architecture, components, and control strategy. International Journal of Electrical Power & Energy Systems, 152, 109264. https://doi.org/10.1016/j.ijepes.2023.109264
12. Committee on Enhancing the Resilience of the Nation's Electric Power Transmission and Distribution System. (2017). Enhancing the resilience of the nation's electricity system. National Academies Press. https://doi.org/10.17226/24836
13. Elmouatamid, A., Ouladsine, R., Bakhouya, M., El Kamoun, N., Khaidar, M., & Zine-Dine, K. (2020). Review of control and energy management approaches in micro-grid systems. Energies, 14(1), 168. https://doi.org/10.3390/en14010168
14. Faisal, M., Hannan, M. A., Ker, P. J., Hussain, A., Mansor, M. B., & Blaabjerg, F. (2018). Review of energy storage system technologies in microgrid applications: Issues and challenges. IEEE Access, 6, 35143–35164. https://doi.org/10.1109/ACCESS.2018.2841407
15. Iweh, C. D., Gyamfi, S., Tanyi, E., & Effah-Donyina, E. (2021). Distributed generation and renewable energy integration into the grid: Prerequisites, push factors, practical options, issues and merits. Energies, 14(17), 5375. https://doi.org/10.3390/en14175375
16. Kostenko, G., & Zaporozhets, A. (2023). Microgrid technologies: A comprehensive review of architectures, challenges, and future directions. Energies, 16(12), 4790. https://doi.org/10.3390/en16124790
17. Majumder, R. (2013). Some aspects of stability in microgrids. IEEE Transactions on Power Systems, 28(3), 3162–3171. https://doi.org/10.1109/TPWRS.2012.2234146
18. Mottahedi, A., Sereshki, F., Ataei, M., Nouri, A. H., & Madani, S. H. (2021). Resilience of critical infrastructure systems: A systematic literature review of measurement frameworks. Applied Sciences, 11(4), 1335. https://doi.org/10.3390/app11041335
19. Ogunjuyigbe, A. S. O., Ayodele, T. R., & Akinola, O. A. (2021). Impact of distributed generators on the power loss and voltage profile of sub-transmission

network. Journal of Electrical Systems and Information Technology, 8(1), 1–14. https://doi.org/10.1186/s43067-021-00030-5

20. Okoye, C. O., & Oranekwu-Okoye, B. C. (2022). Economic feasibility of solar PV system for rural electrification in sub-Sahara Africa. Renewable and Sustainable Energy Reviews, 30, 583–599. https://doi.org/10.1016/j.rser.2013.10.002

21. Olatomiwa, L., Mekhilef, S., Ismail, M. S., & Moghavvemi, M. (2021). Energy management strategies in hybrid renewable energy systems: A review. Renewable and Sustainable Energy Reviews, 62, 821–835. https://doi.org/10.1016/j.rser.2016.05.040

22. Olatunde, T. M., & Adejumobi, I. A. (2023). Optimization of photovoltaic microgrid systems for rural electrification in Nigeria: A multi-objective approach. Nigerian Journal of Technology, 42(2), 215–226. https://doi.org/10.4314/njt.v42i2.5

23. Pires, V. F., Pires, A., & Cordeiro, A. (2023). DC microgrids Benefits, architectures, perspectives, and challenges. Energies, 16(3), 1239. https://doi.org/10.3390/en16031239

24. Singh, S., & Singh, S. (2024). Challenges and opportunities in solar microgrid integration: A review of recent advances. Sustainable Energy, Grids and Networks, 37, 101265. https://doi.org/10.1016/j.segan.2024.101265

25. Twaisan, K., & Barişçı, N. (2022). Integrated hybrid renewable energy systems in microgrids: A review of solar, wind, and hybrid system configurations. IEEE Access, 10, 90945–90963. https://doi.org/10.1109/ACCESS.2022.3202051

26. Ugwoke, B., Gianaroli, F., Barchetti, M., Colangelo, A., Bertoldi, P., Corey, G., & Leonardi, B. (2021). Modelling framework for local energy systems with power-to-gas technology for sector coupling. Energies, 14(21), 7145. https://doi.org/10.3390/en14217145

27. Zhang, L., Gari, N., & Hmurcik, L. V. (2014). Energy management in a microgrid with distributed energy resources. Energy Conversion and Management, 78, 297–305. https://++doi.org/10.1016/j.enconman.2013.10.065