Rose D. Mohammad

Electricity Generator Automation Prototype

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master’s Thesis

4 Nov 2023

PREFACE

I am proud to be a graduate of Metropolia University, and I extend my heartfelt gratitude for welcoming me into this esteemed academic community. My journey here began in 2021 when I arrived in Finland as a diplomat's family member. Fortunate to commence my master's degree in September 2022, I aimed for a subject that seamlessly blends my interest with a positive impact on my home country's daily electricity service.

Gratitude and thanks to my dedicated tutor, Mr. Ville Jääskeläinen, whose unwavering support has been invaluable from the outset of this project. Equally, I appreciate Miss. Saana Vallius, whose class in the second semester sparked the idea for my thesis. Her guidance and passion amplified my love for the master's program.

Special acknowledgment goes to Mr. Eetu Jääskeläinen for his assistance with schematic diagrams. I extend my gratuities to all my professors at Metropolia University whose teachings made this significant day possible.

I extend my deepest appreciation to my mother, Ms. Tania Hussein, Deputy Charge of the Iraqi Embassy in Finland, and Estonia. Her unwavering support and guidance throughout my academic journey have been instrumental.

Special thanks to my father, Mr. Dislahd Mohammad Sh, Deputy Minister of Electricity in KRG, Iraq, whose support and encouragement, both financially and professionally, steered me towards this thesis idea. Being an electrical engineer and an expert in Electricity Generators, his insights have profoundly shaped this project. The installation of this automated generator system will take place in my personal home in Iraq as a start before it will be introduced in market.

21/Spe/2023

Rose D. Mohammad Sh

Abstract

Author:	Rose D. Mohammad
Title:	Electricity Generator Automation
Number of Pages:	xx pages + x appendices
Date:	1/March/2023

The persistent shortage of electricity in numerous Middle Eastern nations, notably Iraq, is a challenging predicament. The state sector provides power supply in most Middle Eastern countries, including Iraq, and the government funds power plants. Despite Iraq's substantial oil reserves, it grapples with the challenge of satisfying its electricity demand, stemming from multifaceted factors.

Local electricity generators have emerged as a prevailing solution for bridging the electricity deficit in Iraq. The government and local people depend on these electric generators for everyday life, and the private sector owns them.

The prevalent use of Iraq's types of Electricity Generators (EG) adds complexity, entangled with challenges, including manual operation, the exigency of maintenance, and the necessity for vigilant monitoring of engine oil quality for local EG. These electric generators are small-sized local generators that can supply electricity for nearly 200-400 homes.

The core aim of this thesis is to forge an automated operation system granting a solution to govern these local Electricity Generators, the central tenets of this objective include the automation of operational control, seamless integration of sensors, facilitation of a user-friendly interface, and the implementation of automated responses.

This project details the creation of a conceptual framework and methodology for a project involving designing and implementing a prototype with specific hardware components, including microcontrollers. It emphasizes the explanation of functions for these elements and their integration into the prototype.

The thesis process includes schematic diagrams, hardware and sensor testing, software development, and coding for Raspberry Pi Pico and a user-friendly interface. Testing procedures encompass both hardware and coding, leading to a summary of crucial findings for automating Electricity Generator operations and future upgrades.

Keywords: Electricity Shortage, Iraq Energy Crisis, Energy sector in Iraq, Electricity Generation, Generator Automation, Raspberry Pi Pico, Generator Monitoring System, Microcontrollers, User-Friendly Interface, Automated Control Systems, Electronics, MicroPython.

Contents

List of Abbreviations

AC	Alternate Current(electric current continuously changes direction)
DC	Direct Current(electrical current that always flows in one direction)
EG	Electricity Generators.
GPIO	General-Purpose Input/Output
HTML	Hyper Text Markup Language
IoT	Internet of Things
IC	Integrated Circuits
IDE	Integrated Development Environment
kW	Kilowatt
kVA	kilo-Volt-Amperes (a unit of power eq ual to (1000 * 0.8) watts)
SBC	Single Board Computers
TW	Terawatt
UML	Unified Modelling Language

1 Introduction

Most Middle Eastern countries, especially Iraq, suffers from chronic electricity shortage. While Iraq is rich in oil, is a member of OPEC and has 8% of the global oil reserve yet it does not have sufficient electricity power, this is for varied reasons (1). Electricity in Iraq depends on natural resources such as Diesel at 8.83%, Hydro 13.71%, Thermal 29.75%, and Gas Turbine 47.70%.

The demand for electricity in the past years in Iraq has increased due to the population increase and the consequences of climate change. The temperature in Iraq reaches above 50 degrees Celsius during summer, and the climate change that resulted in sandstorms most of the days during the summer season has increased electricity demand. In Iraq, the government produces and sells electricity and the price is subsidised by national income.

Most of the cities in Iraq have power plants that can provide a limited electricity supply, which is not enough for 24 hours a day. For this reason, the private sector sells electricity by operating small generators, covering the power supply deficiency by 50% of locally requested electricity needs.

These generators are spread in all cities in Iraq and are favoured by local users to cover the daily need for electricity. Take one city as an example, if the population of a city is one million, the number of generators needed is 2500 of 1000kVA output power. This requires at least ten thousand operators over two shifts, and two operators cover each working shift.

Since these generators are used widely everywhere in Iraq, this project aims to help the generator operators to have an automated control and the ability to switch on, off, restart, standby and reading water level, temperature, and engine oil quality.

1.1 Electricity Production in Iraq

Iraq's energy sector has faced challenges due to its limited and old infrastructure, resulting in electricity shortages and frequent power outages. Iraq's overburdened infrastructure needs help to keep up with the country's rising electrical demand. The government predicted the peak demand climbed 1.4 GW annually to a new record 34.18 GW for summer 2023. According to Ministry of Electricity records, demand rose 8.8% in the last three years to an average of 19.81GW.

The Ministry of Electricity records an estimated available electricity capacity raised to 22GW last year, and it will be a challenge to meet the demand in the following years. According to the documents, the predicted gap between the supply that can meet demand at the summer peak has increased from 9GW to 10.8GW in the last three years. The difference grows to roughly 12.2 GW, up from 11.66 GW when transmission losses are considered(2).

Conflicts, political instability, and other unforeseen circumstances have further exacerbated the situation, making it challenging to maintain a consistent and uninterrupted energy supply. In 2004, the country's electricity demand exceeded the installed capacity by 3,950 MW. From 2010 to 2020, Iraq witnessed an average annual increase of 7% in net energy generation, producing an estimated 92-terawatt hours of electricity. However, despite this growth, the rising electricity demand has outpaced the expansion of generating capacity, leading to frequent power disruptions experienced by most homes in Iraq.

The country relies on renewable energy for only 19% of its electricity production, further exacerbating the gap. As a solution to this problem, many households have resorted to personal generators, which offer a convenient and cost-effective means of compensating for the electricity shortfall. These generators are readily available and require minimal maintenance.

District generators include one or more large-scale generators supplying electricity to multiple households within a district, run by the private sector. These systems involve heavy equipment and require a substantial transmission capacity and a dedicated maintenance team.

On the other hand, individual home generators are typically smaller and are powered by fuels such as gasoline, diesel, natural gas, or propane and serve to provide electricity to individual households. Personal generators are more cost-effective, requiring less maintenance and offering ease of installation and operation. However, their capacity is limited by the generator's power output.

In both cases, there is an abundance of generators in the country that require regular maintenance. The workers overseeing these generators must allocate time to ensure proper supervision and prevent system breakdowns or disruptions. Unfortunately, despite these efforts, power outages are still common occurrences.

1.2 Business challenges

The Electricity Generator (EG) system presents several challenges for its users. Firstly, users must stay alert when the system stops or shuts down due to overload or extended operation. In such cases, users must manually switch it on or restart the system, which can be inconvenient and requires constant attention.

In addition to that, maintaining proper levels of engine water and fuel is crucial for the system's smooth operation. Users need to be mindful of the engine water level, ensuring it stays within accepted level, and monitor the fuel level to prevent it from running too low. Neglecting these levels can lead to disruptions and adversely affect the performance of the EG system.

Moreover, the quality of the engine oil plays a vital role in engine performance. Regularly checking the oil levels and ensuring their quality daily is essential for keeping the engine well-lubricated, reducing friction, and minimizing unnecessary power loss. Failing to monitor and maintain the oil levels and quality can result in inefficient operation and potential system failures.

1.3 Objectives

This project aims to implement an automated system capable of monitoring and controlling the switch on/off functionality and the standby mode operations of an electricity generator. This system will also collect data from various sensors, measuring water level, engine oil quality, and temperature levels.

The primary purpose of this system is to establish an efficient and comprehensive control platform that empowers users to maintain optimal control over the system's various functions. Additionally, it can spot problems, such as sudden temperature increases or declines in oil quality and respond quickly.

To ensure user-friendliness and ease of operation, the system will feature a graphical user interface that simplifies the management of functions and provides users with a comprehensive overview of the system.

The system will deliver real-time readings from the various sensors, enabling users to promptly identify any abnormal readings and take appropriate action.

In summary, the objective of this automated system is to provide an efficient and secure platform for managing and monitoring the switch on/off and standby modes of an electricity generator. This system is an excellent option for users looking for a trustworthy and effective control system because of its user-friendly interface and extensive security features.

1.4 Study plan

The project plan contains the following steps:

1. Identifying the key elements of Electricity Generators
2. Selection of suitable prototype platform
3. Implementation of Hardware
4. Implementation of Software
5. Testing hardware and software of the prototype

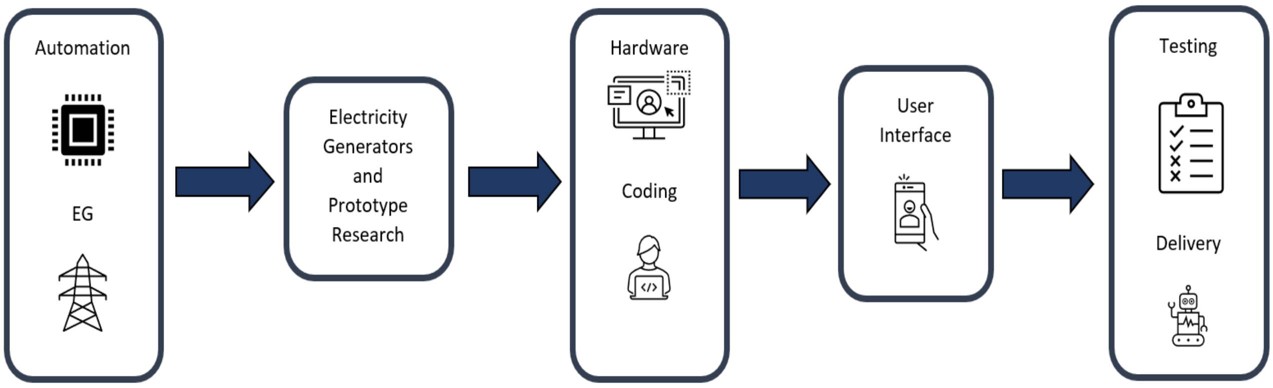


Figure 1 Thesis Flow

This thesis is divided into six sections; the first includes a brief overview of the Middle East's chronic electricity shortage, focusing on Iraq. The introduction explains the significance of the problem and the challenges the region faces. It also outlines the project's primary objective: to develop an automated system for controlling electricity generators. The section concludes with a summary of what to expect in the report.

In the second section, the report delves into the theoretical background of the project. It discusses the current manual methods used to operate and maintain electricity generators. The importance of electricity generators is highlighted, along with the disadvantages associated with them. The section also explores the components and operation of these generators. Additionally, it provides insights into hardware design research.

The third section focuses on microcontrollers and their types, evaluation of the existing system, hardware design, and explanations of selected components. Also, schematic diagrams, UML diagrams, and the printed prototyping process are explained.

The fourth section focuses on the implementation and practical aspects of the project, followed by the software design, emphasising the coding for Raspberry Pi Pico and the development of a user-friendly interface.

The fifth section of the research outlines the testing procedures, it includes hardware testing, which involves evaluating the Raspberry Pi Pico and sensors. The research describes how the hardware was evaluated to ensure successful functionality. Testing also included step-by-step code testing and user interface testing.

2 Electricity Generator

The history of electricity dates back to ancient times when people discovered that certain materials, such as amber, could generate a static electric charge when rubbed against fur or wool. However, only in the late 18th century was progress made in understanding and harnessing electricity.

In 1752, Benjamin Franklin conducted his famous kite experiment, where he flew a kite with a metal key attached during a thunderstorm. This experiment demonstrated the connection between lightning and electricity and laid the foundation for further exploration.

In the early 19th century, scientists such as Alessandro Volta and Michael Faraday contributed significantly to understanding electricity. In 1800, Volta created the first electric battery, the Voltaic Pile, which generated an uninterrupted flow of electric current. Faraday's experiments with electromagnetic induction led to the development of the electric generator, transforming mechanical energy into electricity.

Based on information from the Institute of Electrical and Electronic Engineers (IEEE), Iberdrola, a Spanish global energy leader, generated a photo illustration in Figure 2 (3).

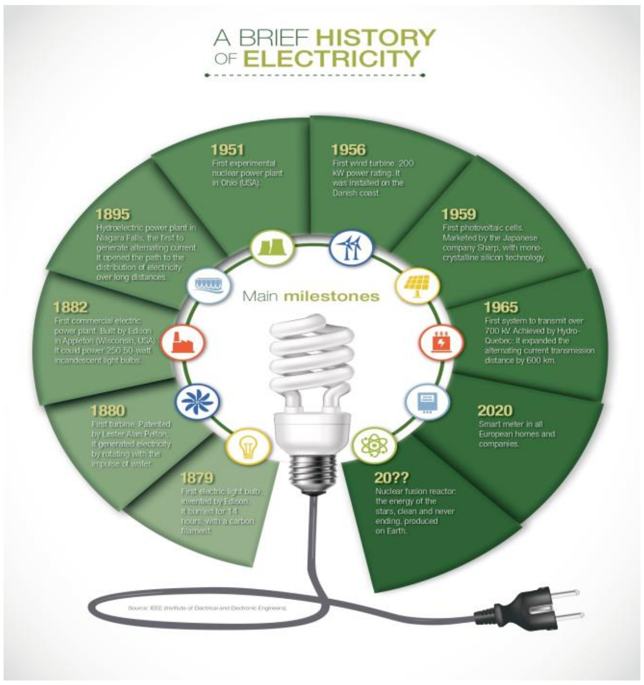

A BRIEF HISTORY OF ELECTRICITY
Main milestones
1951 First experimental nuclear power plant in Ohio (USA).
1956 First wind turbine. 200 kW power rating. It was installed on the Danish coast.
1959 First photovoltaic cells. Marketed by the Japanese company Sharp, with mono-crystalline silicon technology.
1965 First system to transmit over 700 kV. Achieved by Hydro-Quebec; it expanded the alternating current transmission distance by 600 km.
2020 Smart meter in all European homes and companies.
20?? Nuclear fusion reactor: the energy of the stars, clean and never ending, produced on Earth.
1879 First electric light bulb, invented by Edison. It burned for 14 hours, with a carbon filament.
1880 First turbine. Patented by Lester Alan Pelton. It generated electricity by rotating with the impulse of water.
1882 First commercial electric power plant. Built by Edison in Appleton (Wisconsin, USA). It could power 250 50-watt incandescent light bulbs.
1895 Hydroelectric power plant in Niagara Falls, the first to generate alternating current. It opened the path to the distribution of electricity over long distances.
Source: IEEE (Institute of Electrical and Electronic Engineers).

Figure 2 “History of Electricity- by Iberdrola

Thomas Edison's discovery of the light bulb in 1879 was a ground-breaking innovation that transformed how we illuminate our environment. Edison's light bulb utilized a filament inside a glass bulb, which emitted light when an electric current passed through it.

As the electricity demand grew, Power generation and distribution systems used to fulfil the increasing energy demand as the need for electricity increased. The first water turbine created by Lester Alan Pelton in 1880. The first centralized power station was established by Edison in New York City in 1882, providing electricity to a small area, which marked the beginning of the modern electrical grid system.

The late 19th and early 20th centuries saw rapid advancements in electricity generation. Nikola Tesla's work on alternating current (AC) systems and George Westinghouse's support led to the widespread adoption of AC power distribution, which offered advantages in long-distance transmission.

The development of power plants using various energy sources accelerated throughout the twentieth century. Coal-fired power plants became prevalent, using natural gas, nuclear energy, and renewable sources such as hydroelectric, wind, and solar power.

Renewable energy sources have gained significant attention with technological advancements and the need for cleaner and more sustainable energy. Worldwide renewable energy installations have increased significantly in the twenty-first century as nations work to cut their carbon footprints and make the transition to a greener future.

Today, electricity plays a central role in our lives, powering our homes, businesses, industries, and technology. It has become indispensable to modern society, driving economic growth, innovation, and improved living standards.

The journey from the discovery of static electricity to the development of advanced power generation and distribution systems has been a testament to human ingenuity and scientific progress. As we progress, the focus remains on enhancing efficiency, sustainability, and reliability in electricity generation, distribution, and utilization.

Iraq Electricity History:

The history of electricity in Iraq stretches back several decades, with notable improvements and problems. Early developments in the introduction of electricity in Iraq may be linked back to the early twentieth century. The first hydroelectric plant in Iraq was built in 1915, using the country's water resources to create electricity. It was based on a steam turbine which used steam to boiler and turn a turbine connected to an electrical generator and convert the rotational energy from the turbine into electrical energy. In addition, a British engineer made an electric machine in Baghdad's 'Khan Dala' building, signaling the beginning of the capital city's energy infrastructure.

The systems were upgraded over the years, and the electricity network grew as per population demands in Iraq. Power generation stations were upgraded, and the electrical grid expanded to cover more regions nationwide to support the development of many sectors, such as industry, commerce, and residential regions.

Despite efforts to expand the electricity infrastructure, Iraq has faced significant challenges in meeting the increasing electricity demand. Population growth, conflicts, and limited production capabilities have resulted in electricity shortages and frequent power outages. The inadequacy of the electrical power generation stations to meet the demand has been a persistent issue(4).

Diversification of Energy Sources in Iraq's electricity generation has traditionally relied heavily on fossil fuels, particularly oil and natural gas. However, there has been a push towards diversifying the energy mix and incorporating renewable energy sources in recent years.

Iraq has over fifty electricity power plants; the government has installed several gas turbine power plants to minimize the gap; power suppliers' companies such as (Stratford, Siemens, Caterpillar, and GE) have installed gas turbine power plants across the country.

These installations have helped to supplement the electricity supply and reduce the dependency on conventional power sources. Iraq has also utilized hydropower as a significant source of electricity, providing electricity over 70%. However, the reliance on hydropower has faced challenges due to climate change and changing water resources.

As a result, the capacity of hydropower plants in Iraq has significantly decreased, impacting the overall electricity supply. Three of those power plants have been installed in the north region of Iraq, that area has hydropower plants.

In Iraq, electricity demand has continually surpassed available generation capacity, resulting in ongoing power outages. Many households and businesses rely on personal generators as an alternate source of electricity to make up for the gap.

It is important to note that the electricity situation in Iraq is dynamic and subject to numerous factors, including political, social, and economic conditions. Efforts are ongoing to address the challenges and improve the electricity infrastructure to provide a stable and sustainable power supply for the people of Iraq.

Although the electricity distribution always had its limitations, the quantity of demands changes per season of the year. The Iraq weather is a reason for the increase in electricity demands(5).

2.1 Current Used Method

A power source generates electricity using the electromagnetic induction principle, which relies on the interaction of electrical conductors and a magnetic field to generate current. Depending on the design and arrangement, the generator produces alternating (AC) or direct (DC) energy by revolving a rotor inside a still stator.

Electricity generators are crucial in various sectors to ensure continuous power supply, support critical operations in commercial, industrial, and residential demands, prevent disruptions, and maintain productivity and quality of life. They serve as reliable backup power sources, bridging the gap during power outages or in areas without reliable grid infrastructure.

2.1.1 Importance of Electricity Generators

The importance of electricity generators in various sectors can be summarized as follows(6):

- **Residential Sector:** Electricity generators provide continuous power supply to homes, ensuring that essential appliances like refrigerators, heating/cooling systems, and security systems remain operational during power outages. They offer comfort and convenience by maintaining lighting and enabling electronic devices, allowing residents to continue with their daily activities uninterrupted.

- **Commercial and Business Sector:** Generators are critical for businesses to maintain uninterrupted operations. They ensure that essential equipment such as computers, servers, communication systems, and cash registers remain functional during power disruptions. By avoiding downtime and maintaining business continuity, generators enable businesses to serve customers, protect valuable data, and avoid financial losses.
- **Industrial Sector:** Electric generators are a reliable backup power source in industries to prevent production losses and maintain smooth operations. They provide power during grid failures, ensuring that critical machinery, manufacturing processes, and industrial equipment continue to function, which helps avoid costly interruptions, equipment damage, and delays in production schedules.
- **Healthcare Sector:** Hospitals, clinics, and medical facilities require a continuous and reliable power supply to operate life-saving equipment, maintain critical care units, and support patient care. Electricity generators offer backup power during power outages or emergencies, ensuring that medical services, surgeries, and patient care are not compromised.
- **Data Centers and IT Sector:** Data centers and IT infrastructure rely on uninterrupted power to safeguard sensitive information and maintain operational efficiency. Generators provide backup power during grid failures, allowing data centers to continue operating, preventing data loss, and avoiding disruptions to online services and digital infrastructure.
- **Emergency Services:** In emergency service buildings like fire stations, police stations, and emergency response centers, generators are necessary. They ensure continuous power supply to communication networks, emergency lights, and crucial equipment, allowing for effective emergency response and coordination during critical situations.
- **Events and Entertainment Sector:** Outdoor events, concerts, and entertainment venues often require a reliable power source in areas

where grid supply may be limited or unavailable. Generators power the stage lighting, sound systems, food stalls, and other event infrastructure, ensuring the smooth and uninterrupted operation of the event.

- **Remote Areas and Construction Sites:** Generators are vital in remote areas with limited or nonexistent grid access. They provide power for construction sites, remote research facilities, off-grid residences, and rural communities, enabling basic amenities, infrastructure development, and economic activities.
- **Energy Resilience and Grid Stability:** Generators contribute to energy resilience by providing backup power during grid failures, natural disasters, or emergencies. They help maintain grid stability by supporting it during peak demand periods or providing ancillary services such as frequency regulation. Generators function as a safety net, preventing widespread outages and minimizing the impact of power disruptions.
- **Disaster Relief and Humanitarian Aid:** In disaster-stricken areas, electricity generators are crucial for powering relief efforts, emergency shelters, field hospitals, and communication systems. They facilitate essential services, medical care, and aid distribution during humanitarian crises, contributing to relief and recovery.
- **Off-Grid and Renewable Energy Systems:** Generators play a role in off-grid locations with limited access to the electricity grid. They provide backup power during periods of low renewable energy generation or high demand, ensuring a stable and reliable electricity supply. Generators can be integrated into off-grid renewable energy systems, enabling a balanced and sustainable energy mix.
- **Economic Growth and Development:** A reliable electricity supply catalyzes economic growth and development. Generators support productivity, attract investments, create job opportunities, and drive economic progress in urban and rural areas by providing consistent power to businesses, industries, and infrastructure projects.
- **Environmental Applications:** Generators powered by cleaner fuels or renewable energy sources contribute.

2.1.2 Disadvantages of Electricity Generators

While electricity generators offer numerous benefits, they also have some disadvantages that should be considered:

- **Fuel Dependency:** Most power plants use fossil fuels like natural gas, diesel, or gasoline. It can also make generators susceptible to fluctuations in fuel prices and availability.
- **Environmental Impact:** Using fossil fuel-powered generators contributes to air pollution, emitting carbon dioxide (CO2), nitrogen oxides (NOx), and particulate matter. These emissions can adversely affect air quality, contributing to air pollution.
- **Noise Pollution:** Generators, especially older or larger models, can be noisy. The constant noise can be disruptive, particularly in residential areas, quiet environments, or during night-time operations. However, newer generator models and soundproofing techniques are available to reduce noise levels.
- **Maintenance and Operation:** This includes frequent oil changes, filter replacements, and fuel management. Additionally, gasoline expenses can be substantial, particularly during prolonged power outages or high demand. Buying and installing one can be expensive based on the generator type.
- **Limited Fuel Storage:** Generators that rely on fuel require adequate fuel storage capacity. It can be challenging when access to fuel sources is limited, such as during natural disasters or in remote areas. Ensuring a sufficient fuel supply is essential to maintain generator operation during extended periods without grid power.
- **Emissions Regulations:** Generators may be required to comply with emissions rules and get permits, depending on the jurisdiction. These restrictions are intended to reduce the environmental impact of generator emissions, but they may increase compliance costs and administrative burdens.
- **Safety Hazards:** Generators involve fuel combustion and power generation, which can offer safety risks if not managed and maintained

appropriately. Fire, carbon monoxide poisoning, electric shocks, and fuel leaks are all possibilities. Following safety requirements, providing sufficient ventilation, and inspecting and servicing the generator regularly can all assist in reducing these dangers.

- **Limited Runtime:** Generators have a limited runtime based on fuel capacity and power demand. Extended power outages may require periodic refueling, which can cause interruptions in the power supply. Additionally, standby generators that run on natural gas or propane may depend on a continuous supply of these fuels.

Despite these disadvantages, advancements in technology are addressing some of these concerns. The development of cleaner and more efficient generator models, increased use of renewable energy sources, and the integration of energy storage systems are improving the sustainability and reliability of electricity generation(7).

2.1.3 Components and Operation of Electricity Generator:

as per the following Figure 3 from ADE shows the functionality and components of Generators (8):

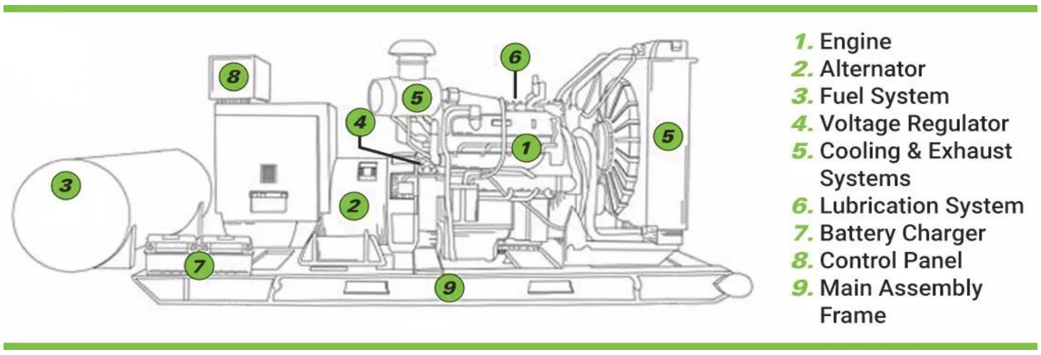


Figure 3 The Main Fueatuers of Electricity Generator

Several engines are used in Iraq, such as (Caterpillar, Cummins, MTU, Perkins, Volvo, MQV, Mitsubishi, etc). Take the Diesel Powered Generating Set (SPG1000-C1), provided by Sakr Power Generation(9), as an example to get a better understanding of the features and components of generators and how they operate:

A. Standard Generation Set Features:

- Water Cooled Diesel Engine
- Oil and fuel filter
- Lube-oil drain valve
- 24V D.C. Electric Starter & Charge Alternator
- Normal duty air filter
- Single-bearing alternator
- Standard Voltage 230/400 Volts 50 Hz
- Digital Voltage Regulator
- 3-pole Circuit Breaker with shunt trip
- Auto Start AMF Module DEIF GC1F as standard, other controllers are available for different applications.
- Fitted with low coolant level shutdown.
- Welded steel base frame with integral
- Anti-vibration mountings
- Industrial type silencer: Noise attenuation
- Stainless steel exhaust bellow
- Set mounted starting batteries.
- Operation & Maintenance Manual
- Parts Manual
- Standard set of labels
- The generating set and its components are factory-built, type and production-tested.

B. The essential Engine components of most common components of any Electricity Generator are:

- **Rotor:** usually referred to as an armature, typically made of copper wire coils.
- **Stator:** which surrounds the rotor, it is a stationary element.

- **Magnetic field:** electricity production requires the creation of a magnetic field by use of an electromagnet or a magnet. The rotor is typically equipped with electromagnets or contains permanent magnets.

C. Operation of an Electricity Generator:

1. **Rotation**: is a fundamental process in an electricity generator that involves the connection of a mechanical power source to the rotor. The crucial generator component rotor is designed to rotate at high speeds when the power supply is activated.

   An automatic power source is coupled to the rotor. This power source can vary depending on the type of generator and the specific application. Examples include steam turbines, gas engines, or water turbines, which can harness different forms of energy to drive the rotor.

   When turned on, the mechanical power source transfers its significance to the rotor, causing it to spin rapidly. This rotational motion is vital in generating electricity, as it sets the subsequent processes of electromagnetic induction and power generation in action. The rotor's rotation generates a changing magnetic field, which induces an electric current in the rotor's wire coils.

   This electric current is then harnessed and converted into usable electrical energy for various applications. A mechanical power source, such as a steam turbine, gas engine, or water turbine, can aid in the rotation of an electricity generator's rotor. This rotational motion is essential for generating electricity and is a crucial step in the overall operation of the generator.

2. **Magnetic induction:** Magnetic induction is a crucial process in electricity generators, occurring as the rotor rotates. As the rotor spins, it generates a dynamic and changing magnetic field.
   This changing magnetic field interacts with the copper wire coils in the rotor, resulting in the induction of an electric current. According to Faraday's law of electromagnetic induction, an induced voltage or

electromotive power (EMF) occurs when a conductor, like copper wire coils, passes along magnetic lines of force(10).

In the case of an electricity generator, the changing magnetic field created by the rotor's rotation allows the copper wire coils to cut through the magnetic lines of force, generating an electric current. The copper wire coils, being conductors, act as pathways for the induced electric current to flow. Then, this electric current can be captured and used for various tasks, such as powering electronics or distributing electricity via power lines.

As a result of the rotor spinning and producing a fluctuating magnetic field, magnetic induction occurs in an electricity generator. According to Faraday's law of electromagnetic induction, this fluctuating magnetic field causes an electric current to be induced in the copper wire coils of the rotor.

3. **Alternating current (AC) or direct current (DC) output:** This is the form in which the rotor of an electricity generator produces electricity. For some uses, the AC output is used directly; however, when direct current (DC) is needed, the generator's AC output must be rectified into DC using a rectifier.

   A rectifier is a device that transforms AC power into DC. It typically consists of diodes arranged in a specific way. The rectifier produces a DC output when the AC output from the generator passes through it.

   The diodes make sure that the current only flows in one direction. When using electronics, batteries, or other electrical systems that require DC power, such as DC motors, it is necessary to convert the AC output to DC. The rectifier is essential to this conversion process because it alters the rotor's generated power into a suitable form for the intended application.

   In conclusion, an electricity generator uses the rotation of its rotor to create electricity in the form of alternating current (AC). The AC output is sent to a rectifier, which turns it into direct current (DC) by permitting the wind to flow in one direction if a direct current is required. The electricity generated is compatible with systems and equipment that need DC power.

4. **Control and Regulation:** The control and regulation systems in an electricity generator, including voltage regulators and governors, are crucial parts that guarantee the stability and quality of the electricity generated, making it suitable for various applications and fulfilling the unique needs of electrical systems.
5. **Power Regulation:** It is critical for an electricity generator to maintain a consistent output voltage and frequency. Automatic voltage regulators (AVRs) are crucial in achieving this regulation. AVRs continuously monitor and adjust the output voltage to keep it within the desired range. By maintaining a stable output voltage, AVRs ensure compatibility with the electrical grid and enable the generator to supply consistent and reliable power. Additionally, frequency regulation ensures that the generated electricity matches the desired frequency of the electrical grid, allowing for seamless integration and operation of electrical devices.
6. **Control and Protection Systems:** are integral components of an electricity generator, ensuring safe and reliable operation. These systems incorporate a range of sensors, instruments, and protection devices to monitor various variables and safeguard against potential issues. Sensors measure essential parameters such as voltage, current, temperature, and speed, providing real-time data on the generator's performance.

   Instruments interpret the sensor readings and provide critical information for monitoring and control purposes. Protection devices are implemented to prevent damage to the generator from electrical faults, overloads, and abnormal operating conditions. These devices include circuit breakers, fuses, relays, and surge protectors designed to interrupt the electrical flow or trigger protective actions when necessary. By detecting anomalies and responding swiftly, the control and protection systems help to minimize the risk of equipment damage, ensure operator safety, and maintain the overall integrity of the generator.

7. **Maintenance and Servicing:** An electrical generator's reliable and effective operation depends on regular maintenance and service. This process entails several standard duties and actions to maintain the generator in top shape. Regular checks are carried out to evaluate the generator's general condition, spot any wear or damage, and handle any possible problems before they become more serious. Lubrication keeps the moving parts in good working order, lessens friction, and increases component longevity.

   A vital maintenance component is testing, which enables the evaluation of the generator's performance and the detection of any functional abnormalities: load testing, voltage testing, frequency checks, and evaluation of safety measures to confirm their effectiveness.

   Operators can reduce the danger of unexpected failures, maximize the generator's performance, and lengthen its operational life by following a thorough maintenance and repair schedule. This proactive strategy ensures the generator stays in top shape and is prepared to provide dependable electricity.

Overall, The basic idea of electromagnetic induction applies to all electrical generators, including diesel generators, gas turbines, and wind turbines(11). The type and use of the generator will determine the size, capacity, and components. Through the technique of electromagnetic induction, an electricity generator transforms mechanical energy into electrical energy, offering a dependable supply of power for numerous purposes.

2.2 Generator Control Panel

A control panel of an electricity generator is a centralized device or interface that allows operators to monitor, regulate, and manage the generator's operation. It often comprises switches, buttons, meters, indicators, and displays that provide information about the generator's performance and allow modifications.

The control panel serves as the generator's command centre, allowing operators to start, stop, and control the generator's output. It offers critical information such as voltage, current frequency, and power output, letting operators ensure that the generator operates within the parameters set.

The control panel may also include safety measures and protection mechanisms to avoid generator damage and ensure safe operation. Examples are circuit breakers, overload protection, emergency stop buttons, and alarms to inform operators of any problems or irregularities.

The purpose of this thesis is to have remote control over the operation. A microcontroller can be attached to the control panel for remote control and monitoring. Integrating a microcontroller with the control panel can be automated and regulate numerous generating functions remotely.

The microcontroller receives commands or instructions data from a remote device or system, such as a computer, smartphone, or dedicated control interface. It can then transmit signals to the control panel to start or stop the generator, change parameters like voltage and frequency, and monitor its performance.

Additionally, the microcontroller is connected to sensors that collect real-time data from the generator, such as temperature, fuel level, and power output. This data can be transmitted wirelessly or via a network connection to a remote monitoring system or user interface, enabling remote monitoring, diagnostics, and generator troubleshooting.

Several types of microcontrollers are used in remote controlling an Electricity generator, depending on the operators' or users' complexity of tasks.

3 Microcontrollers and Sensors

Microcontrollers or Single Board Computers (SBCs) are integrated circuits (ICs) with a microprocessor core, memory, and peripherals on a single chip. They are miniature computers designed to do specific tasks with minimal external components and power consumption(12). Microcontrollers are widely employed in various applications requiring control, automation, and embedded systems. The Figure 4 represents Raspberry Pi 4 Model B(13).

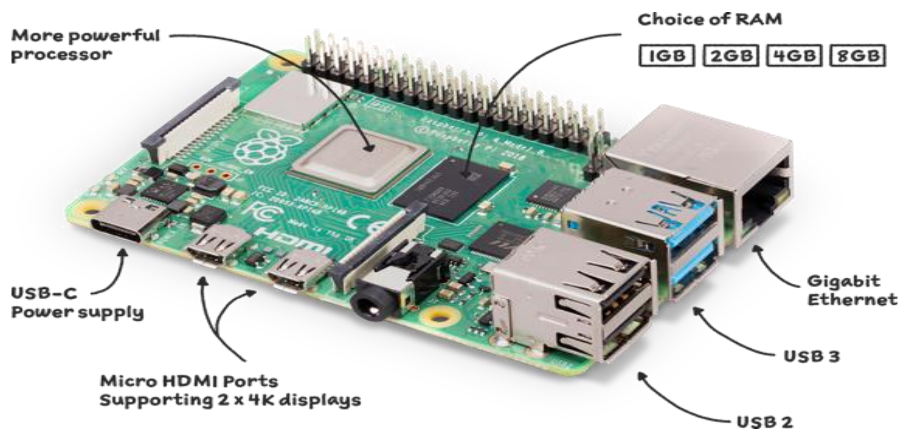


Figure 4 Microcontroller Example (Raspberry Pi 4 model b)

Some of the most essential qualities and features of microcontrollers are as followings:

- **Microprocessor Core:** A microcontroller's central processing unit (CPU) or microprocessor core executes instructions and conducts computations.
- **Memory:** They often have built-in memory, such as read-only memory (ROM) for storing firmware or program instructions and random-access memory (RAM) for storing temporary data.
- **Peripherals:** On-chip peripherals include timers, analogue-to-digital converters (ADCs), digital-to-analogue converters (DACs), input/output (I/O) ports, serial communication interfaces (UART, SPI, I2C), and interrupt controllers. These add-ons enable interaction with external devices and sensors.

- **Low Power Consumption:** Microcontrollers are designed to function with low power consumption, making them ideal for battery-powered or energy-efficient applications.
- **Real-Time Capabilities:** Many microcontrollers have real-time features like timers and interrupt controllers that enable accurate timing and reaction to external events.
- **Programming:** Specific programming languages, such as C or assembly language, are used to program microcontrollers. Microcontroller code is written, compiled, and debugged using development tools and compilers.
- **Embedded Systems**: Microcontrollers are primarily used in embedded, dedicated computer systems built to perform specific jobs or functions within larger systems. Robotics, industrial automation, consumer electronics, automotive systems, medical gadgets, and household appliances are among the examples.

3.1 Microcontroller Types

There are many types of microcontrollers that can be used for this project that involves automation control of a generator with sensors and monitoring. Several Single-Board Computers (SBCs) could be suitable depending on specific requirements and preferences. Many microcontrollers are available in the market, each with unique features and capabilities(12). Some of the most used types of microcontrollers or SBCs are:

1. 8-bit Microcontrollers: These are some of the simplest and most cost-effective. They are often used in simple embedded systems that require basic functionality, such as switching on/off LEDs or sensors.
2. 16-bit Microcontrollers: These are more powerful than 8-bit microcontrollers and suitable for applications requiring more processing power and memory. They are used frequently in applications such as motor control and home automation.
3. 32-bit Microcontrollers: These SBCs are useful for computationally intensive applications and provide significantly more processing and

memory than 16-bit microcontrollers. They are extensively employed in applications for robotics, sophisticated motor control, and audio/video processing.

4. ARM-based Microcontrollers: These are based on the ARM architecture and are frequently utilized in applications requiring high performance while consuming little power. They are prevalent in applications like smartphones, tablets, and other portable devices.
5. FPGA-based Microcontrollers: These are based on Field Programmable Gate Arrays (FPGAs) and offer high flexibility and customization. They are commonly used in signal, image, video, aerospace, and defense systems.

3.2 Microcontroller Models

Many microcontroller models are available, each with unique features and capabilities. The choice of microcontroller depends on the application's specific requirements, such as processing power, memory, and power consumption. Some brand examples of SBCs are:

1. Raspberry Pi Pico: Raspberry Pi Pico is versatile and suitable for various applications, such as robotics, automation, home electronics, and educational projects. Its compact size, powerful microcontroller, and affordable price make it a popular choice among hobbyists, educators, and professionals alike, and it can be programmed using MicroPython, C/C++, or other programming languages. Inexperienced and seasoned developers can use it because it supports well-known development tools and environments.
2. Raspberry Pi 4: More specifically, Raspberry Pi 4 model B is a well-liked option for automation projects because of its priced, low power-consuming, and extensive user and development communities. Connecting to sensors and other peripherals is simple because of its numerous USB ports, Ethernet, Wi-Fi, and Bluetooth connectivity.

Raspbian, Ubuntu, and other Linux variants are among the operating systems that the Raspberry Pi 4 can run.

3. Beagle-Bone Black: The Beagle-Bone Black is another popular choice for automation projects due to its powerful processor, large memory capacity, and high-speed connectivity(14). It has multiple USB ports, Ethernet, and HDMI (High-Definition Multimedia Interface) connectivity, making it easy to connect to sensors and other peripherals. The Beagle-Bone Black runs on a Debian-based Linux operating system and is compatible with many programming languages, such as Python, C++, and Java.
4. Odroid XU4: Equipped with an octa-core processor and 2GB of RAM, the Odroid XU4 is a high-performance SBC. Thanks to its many USB ports, Ethernet, and HDMI connectivity, it is simple to connect to sensors and other peripherals(15). The Odroid XU4 is compatible with numerous programming languages, including Python, C++, and Java, and it is compatible with various Linux distributions, including Ubuntu, Debian, and Arch Linux.

Regardless of the SBC chosen, the system needs to connect the sensors to the SBC's GPIO (General-Purpose Input/Output) pins, write code to read sensor data and control the generator based on the readings. Programming languages like Python, C++, or Java can be utilized to build up the user interface to display readings of the water or fuel level and notify the user when they cross a predetermined threshold.

3.3 Microcontroller Selection

The choice of which board to use for the Prototype depends on various factors, including the project requirements, desired features, and budget. A comparison of the options provided are:

1. Raspberry Pi Pico:

- **Pros:** It is a low-cost microcontroller board with a powerful processor and many GPIO pins. It is suitable for small-scale projects and applications that require real-time control and fast response times.
- **Cons:** It has limited memory and lacks some features found in full-fledged single-board computers, such as built-in networking or multimedia capabilities.

2. Raspberry Pi 4:
- **Pros:** It is a powerful single-board computer with a quad-core processor, ample RAM, built-in networking (Ethernet and Wi-Fi), and multimedia support. It offers many connectivity options and can handle more complex tasks than microcontrollers.
- **Cons:** It may cost more than the Raspberry Pi Pico and has fewer GPIO capabilities.

3. BeagleBone Black:
- **Pros:** It is a single-board computer that mixes performance and cost well. It has a potent processor, plenty of RAM, and many GPIO pins. It also includes networking and supports a variety of operating systems.
- **Cons:** It may not be as popular or widely supported as the Raspberry Pi boards and its community, and documentation may be less extensive.

4. Odroid XU4:
- **Pros:** It is a powerful single-board computer with an octa-core processor and ample RAM. It offers more processing power compared to the other options listed.
- **Cons:** It is relatively more expensive, and its GPIO capabilities may not be as extensive as the Raspberry Pi boards.

Considering the component required for the Prototype (water level sensor, temperature level sensor, oil quality sensor, LED control), the Raspberry Pi Pico, more specifically (Raspberry Pi Pico W), is a suitable choice due to its low cost, sufficient processing power, and GPIO capabilities. However, if it requires additional features, networking capabilities, or more processing power for advanced simulations, consider using the Raspberry Pi 4, BeagleBone Black, or Odroid XU4.

3.4 Components Selection

This phase included the identification and selection of the hardware components used in the system and other necessary elements based on the requirements for the Prototype, such as:

1 Microcontroller:

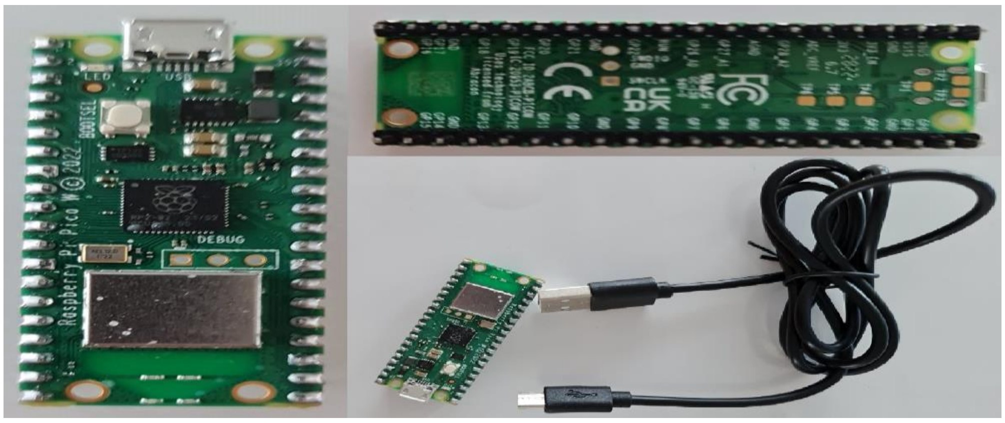


Figure 5 Raspberry Pi Pico W (original photo)

The key features and specifications of the selected Raspberry Pi Pico include:

a. **Microcontroller:** It is built on the RP2040 microcontroller chip developed by Raspberry Pi. The RP2040 features a dual-core Arm Cortex-M0+ processor running up to 133MHz, providing sufficient processing power for various applications.
b. **Memory:** Raspberry Pi Pico has 264KB of SRAM for data storage and 2MB of onboard flash memory for programming storage.
c. **GPIO Pins:** The board has 26 multi-function GPIO pins for digital input and output, analogue input, and other devices. These pins can be programmed to interface with sensors, actuators, displays, and other components.

d. **Programming:** Raspberry Pi Pico can be programmed using MicroPython, C/C++, or other programming languages. It supports popular development tools and environments, making it accessible to developers of various skill levels.
e. **Connectivity:** The board offers USB 1.1, Wi-Fi, and Bluetooth enabling to connect to other devices and computers.

Raspberry Pi Pico is known for its low cost, high performance, and flexibility. It suits various projects, including robotics, home automation, IoT (Internet of Things) devices, and educational applications. With its compact size and extensive GPIO capabilities, Raspberry Pi Pico is popular among makers, hobbyists, and educators. The Prototype need Raspberry Pi Pico W and a micro-USB cable for power and programming.

2 Breadboard: A breadboard is a standard board used in electronics prototyping. It is a reusable device that allows quick building and testing of electronic circuits without soldering (16). It consists of a rectangular plastic board with a grid of interconnected metal sockets or holes, one of the tools used as a base in the prototype.

3 LED Lights: Light-emitting diodes, or LEDs, are frequently employed with microcontrollers for various tasks, including visual feedback, status signalling, and user interface components(17). LEDs are compact, low-power gadgets emitting light when an electric current flows through them. Some key points regarding the use of LED lights with microcontrollers:

- **Visual Feedback:** LEDs often provide visual feedback to users or indicate the status of a system or a specific function. For example, an LED can be programmed to light up when a condition blinks at a specific rate to indicate a specific mode or operation.
- **GPIO Pins:** Microcontrollers typically have General-Purpose Input/Output (GPIO) pins that can be configured as digital output pins. These pins can be connected to an LED with a series resistor to control illumination.

- **Circuit Connection:** Because LEDs have polarity, they must be connected in the proper direction for them to work. The LED's shorter leg (cathode) should be linked to the microcontroller's GPIO pin through a current-limiting resistor. The LED's longer leg (anode) should be connected to the positive voltage supply (VCC).

- **Current-Limiting Resistors:** LEDs need current-limiting resistors to avoid an excessive current flow that could harm the LED or the microcontroller. Ohm's Law can calculate the resistor's value based on the LED's forward voltage and the desired current.
- **Multiplexing:** Microcontrollers with limited GPIO pins may utilize multiplexing techniques to control multiple LEDs using smaller pins. This involves rapidly switching between LEDs to create the illusion of simultaneous illumination.

- **PWM Control:** A LED's brightness can be adjusted using pulse width modulation (PWM). By modifying the duty cycle of a PWM signal given to an LED, it is possible to change how bright the LED perceives itself to be.
- **External LED Drivers:** For more advanced LED applications or when driving multiple high-power LEDs, external LED driver ICs or modules may be used. These devices provide features such as constant current control, dimming capabilities, and higher power handling.

4 Sensors: several sensors will be used in the prototype implementation, such as:

a) Water Level: The engine coolant is an essential part to control the overloaded shutdown caused by rising temperature in the country during summer season . The Water amount Sensor DC3V-5V is a sensor that detects water in a tank or container(18). It is extensively used in automatic water level control systems, aquariums, industrial tanks, and water storage systems.

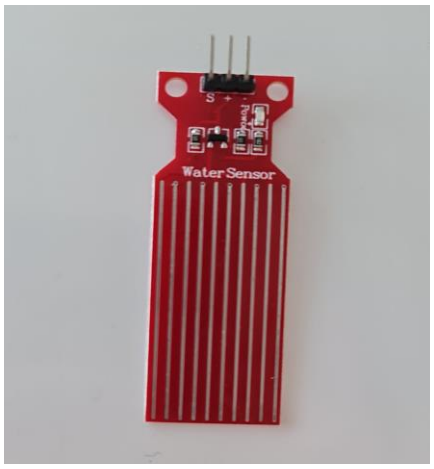


Figure 6 Water level Sensor DC3V-5V

The sensor often comprises several exposed metal probes or electrodes set at various levels along the tank's height. The number of probes varies according to the model of the water level sensor. The sensor module is attached to these probes.

When the water level rises and encounters the probes, an electrical circuit is completed, and the sensor module detects the change in conductivity. This change in conductivity is used to determine the water level.

The Water Level Sensor DC3V-5V is designed to operate within a 3V to 5V DC voltage range, making it compatible with most microcontrollers and development boards. It typically provides a digital output signal (such as HIGH or LOW) to indicate the presence or absence of water at a specific level.

b) Temperature/Humidity sensor: This sensor measures the digital temperature and humidity. The sensor model is DHT11, often used in projects and applications that require ambient temperature and humidity monitoring(19). The sensor gives accurate and trustworthy measurements within a specific range and is affordable and widely available. A specific voltage range is required to operate the DHT11 temperature and humidity sensor. The DHT11 sensor operates with a 3.3V to 5V DC source voltage. Most popular microcontrollers and development boards operating at 3.3V or 5V logic levels are compatible with this voltage range.

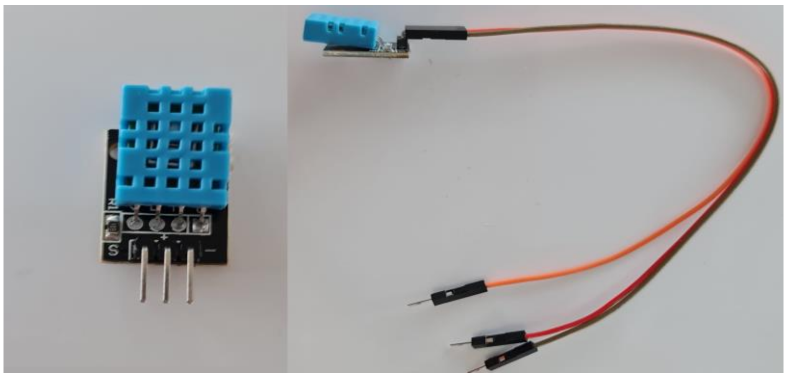


Figure 7 Temperature/Humidity Sensor DHT11

- **Temperature Measurement:** The DHT11 sensor has a limited range of 0 C^o to 50 C^o, and the precision is 2 C^o.
- **Humidity Measurement:** With an accuracy of 5%, the sensor can monitor humidity in the 20% to 90% range.
- **Low Power Consumption:** The DHT11 operates on low power and requires minimal external components, making it suitable for battery-powered applications.
- **Simple Interface:** The sensor has three pins - VCC (power supply), GND (ground), and OUT (data output). It can be connected to microcontrollers or development boards Raspberry Pi Pico.

c) Turbidity Sensor TSD-10: A turbidity sensor measures the clarity or cloudiness of a liquid by determining the number of suspended particles or solids present in the liquid(20). It is commonly used in water quality monitoring, environmental monitoring, and industrial applications such as food and beverage production, pharmaceutical manufacturing, and chemical processing where the particulate matter level in a liquid must be assessed. They also are used in research studies and scientific experiments involving particulate matter analysis in liquids.

Turbidity sensors work based on the principle of light scattering. They emit light into the liquid sample and measure the intensity of the scattered or reflected light to the sensor. The light scatters when there are suspended particles in the liquid, lowering the measured intensity. It is connected to a single connector board (LMV358), a three-pin interface is provided by the LMV358 IC-based module to connect to microcontroller, and the module also has an analogue/digital selector switch to switch between analogue and digital output mode, also contain calibrator that user can calibrate the sensor when needed.

The turbidity sensor provides a quantitative measurement of turbidity, typically in units of nephelometric turbidity units (NTU) or Formazin turbidity units (FTU). The NTU and FTU are standard units used to express turbidity levels.

It is crucial to remember that the voltage range can vary depending on the demands and specifications of the turbidity sensor. Others may function at more comprehensive voltage ranges, while specific turbidity sensors may have a smaller voltage range, such as 4.5V to 5.5V.

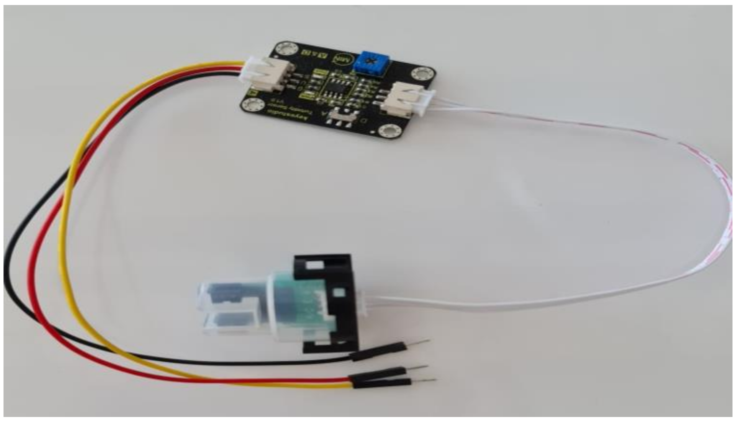


Figure 8 Turbidity Sensor

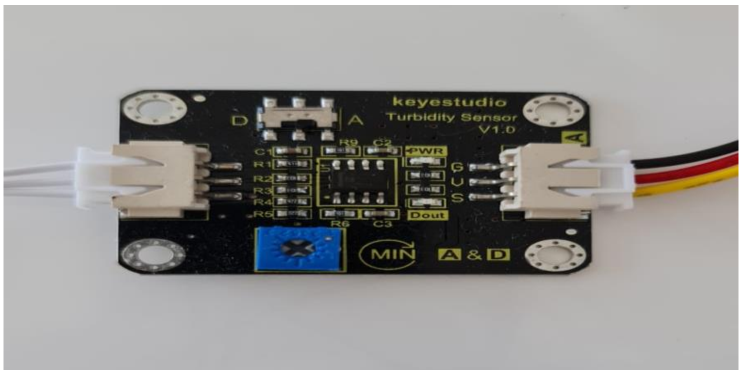


Figure 9 Single Conector Board

A turbidity sensor used for engine oil quality testing, should include the following features and considerations:

- **Oil Compatibility:** The sensor should be compatible with engine oils and able to operate reliably in oil environments. It should be resistant to oil contamination and designed to withstand the properties of engine oil, such as temperature, viscosity, and chemical composition.
- **Particle Size Detection:** The sensor needs to be able to identify a variety of particle sizes that are important for engine oil analysis. Engine oil can contain varying-sized particles, from large metal fragments to microscopic contaminants, so the sensor should have appropriate sensitivity and measurement range.
- **Accuracy and Sensitivity:** The sensor should provide accurate and reliable measurements capable of detecting even low levels of turbidity or suspended particles. It should be sensitive enough to detect changes in oil cleanliness over time.
- **Robust Construction:** The sensor should be built to withstand the harsh conditions of engine environments, including temperature variations, vibrations, and potential exposure to oil contaminants. Robust construction ensures long-term durability and consistent performance.
- **Calibration and Calibration Standards:** The turbidity sensor may require calibration to establish a correlation between turbidity measurements and oil quality parameters. Calibration standards or reference samples can configure the sensor and ensure accurate readings.

5 Jumper Wires: Jumper wires are essential in microcontroller circuits. They establish electrical connections between various components on a breadboard or a circuit board, such as Component connections, Microcontroller pin connections, Breadboard connections, Prototyping, and External circuit connections(16).

Depending on the type of connections needed, they are typically available in different lengths and can be male-to-male, male-to-female, or female-to-female. Jumper wires provide the flexibility and convenience required modifications, controller circuit development and testing phases, allowing for easy connection modifications, and troubleshooting.

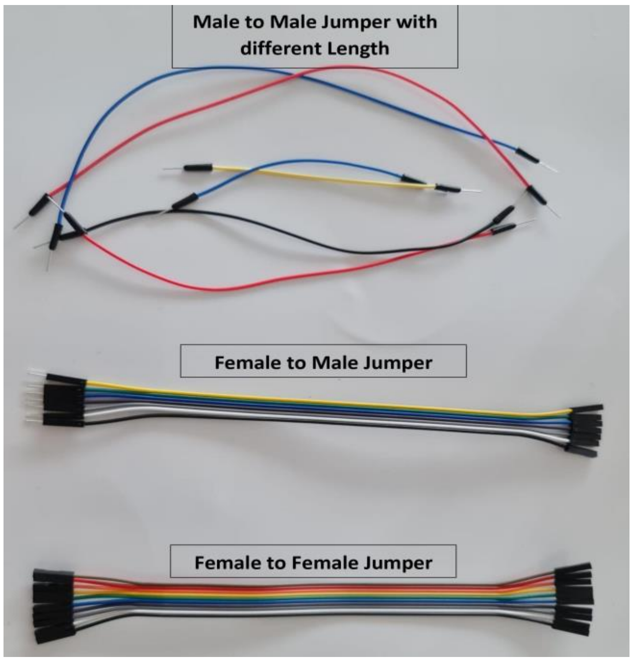


Figure 10 Jumper Wires

6 Resistors: Resistors are adaptable parts used in a variety of electronic applications, and they are essential for shaping electrical signals, regulating current flow, and setting voltage levels in electronic circuits, such as:

- **Current Limiting:** One of the primary functions of resistors is to limit the flow of electric current in a circuit. By resisting the current flow, they prevent excessive current from damaging sensitive components.
- **Voltage Division:** Resistors are used in voltage dividers, circuits that divide a voltage into smaller parts. This is useful in various applications, including setting reference voltages, level shifting, and biasing transistors.

- **Biasing:** In transistor circuits, resistors are used to provide the necessary bias voltage to ensure the proper operation of the transistor.
- **Signal Conditioning:** Resistors can modify and shape electrical signals in a circuit by filtering high-frequency noise or attenuating signals.
- **Pull-Up and Pull-Down:** The resistors ensure that digital inputs to microcontrollers or other digital circuits are at a known voltage level when not actively driven.
- **Current Sensing:** Resistors been used in current sensing circuits to measure the amount of current flowing through a particular circuit part.
- **Voltage Dropping:** In power supply circuits, resistors can drop voltage and regulate current in certain circuit parts.
- **Timing:** Resistors, in combination with capacitors, are used to create timing circuits, such as in oscillator circuits or timing delays.
- **Temperature Sensing:** Some types of resistors, like thermistors, change their resistance with temperature. These are used in temperature sensing applications.
- **Matching and Calibration:** In precision circuits, resistors may be used for calibration and matching to ensure accurate and consistent performance.

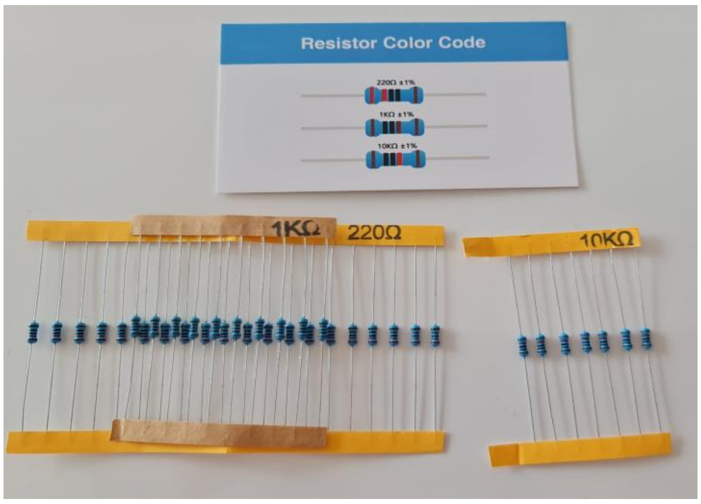


Figure 11 Resistors

4 Implementation

The implementation section contains hardware diagram design, connecting prototype components, microcontroller coding, and user interface design.

4.1 Schematic Diagram Design

A schematic design, a circuit diagram, is a graphical representation of an electronic circuit. It uses standardized symbols to depict the various electronic components and their interconnections within the circuit(21). Schematic designs serve as blueprints or visual representations of the circuit's functionality, allowing engineers, designers, and technicians to understand how the circuit works without the need to examine physical components. Key features of a schematic design include:

1. Symbols: Each electronic component in the circuit is represented by a specific symbol. These symbols are standardized and universally recognized, making conveying the circuit's functionality easier regardless of language or region.
2. Component Labels: Each component is typically labelled with its reference designator, such as R1 for a resistor or C2 for a capacitor. These labels help identify and reference components in the circuit.
3. Interconnections: Lines or wires illustrate the electrical connections between components. The interconnections show how the components are linked and how signals flow through the circuit.
4. Power and Ground Symbols: Symbols representing power supply connections (VCC, +V, etc.) and ground (GND) are commonly used to indicate the power and reference points in the circuit.
5. Nets and Labels: Nets name specific interconnecting wires or nodes within the circuit. Labels are placed next to nets to indicate their purposes, such as input, output, or specific signal names.
6. Design Rules: Schematic designs adhere to specific design rules and conventions to ensure clarity, readability, and consistency. These rules help convey the circuit's function accurately.

The schematic diagram in Figure 12 contains the main components of the prototype, and how they have been connected to the microcontroller, details of the functionality of the sensors, such as the turbidity sensor and water level sensor. The diagram was drawn using KiCad 7.0, “an open-source software suite for creating electronic circuit schematics and printed circuit boards (PCBs)”(22).

In the Automation system, as shown in the schematic diagram, three LED lights are connected to different pins on the microcontroller through male-to-male jumper wires, representing the On, Off, Restart Electricity generator system. Also, a temperature and humidity sensor are connected to the microcontroller that will measure the temperature inside the electricity generator engine (installed inside the generator engine's walls). The water level sensor will measure the water of the coolant tank in the engine, which will be connected to the analogue pin of the microcontroller. The schematic diagram indicates how the data is collected through an electric circuit on the sensor. The turbidity sensor has been connected to the microcontroller that will measure the quality of the engine oil, and the schematic diagram indicates how the turbidity sensor can collect data. It has both an analogue and digital system, but as it will be used on Raspberry Pi Pico, the digital circuit will be used. Also, the sensor has a potentiometer for manual calibration.

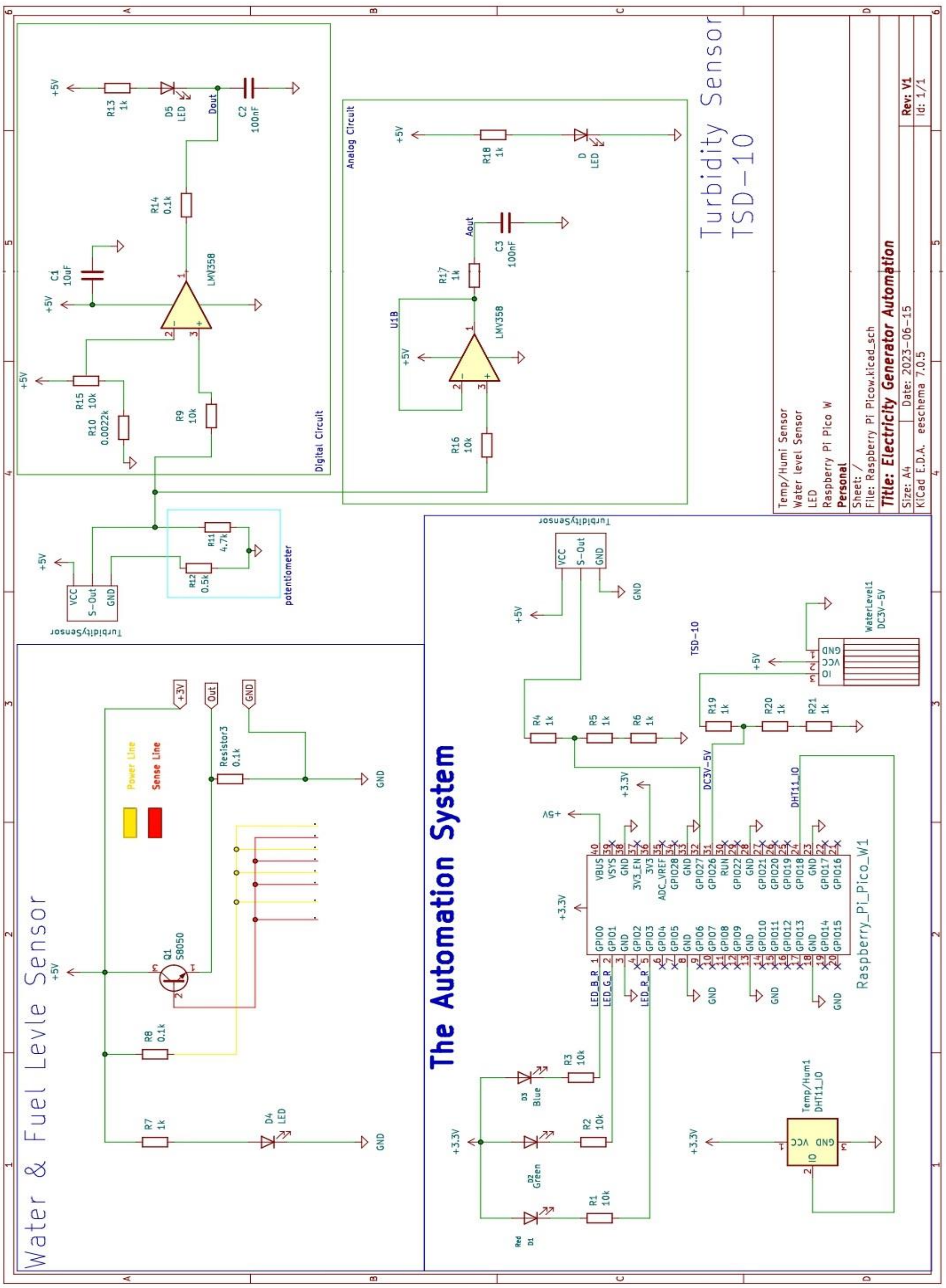


Figure 12 Automation system Schematic Diagram

4.2 Prototype Component Connections

The physical prototype of the hardware system has been built based on the schematic diagram design. Assembling the components, pairing the connections, and testing the initial functionality of the system, as in Figure 14. The left of the figure shows the sensor setups, and the right shows the connection between sensors and the microcontroller.

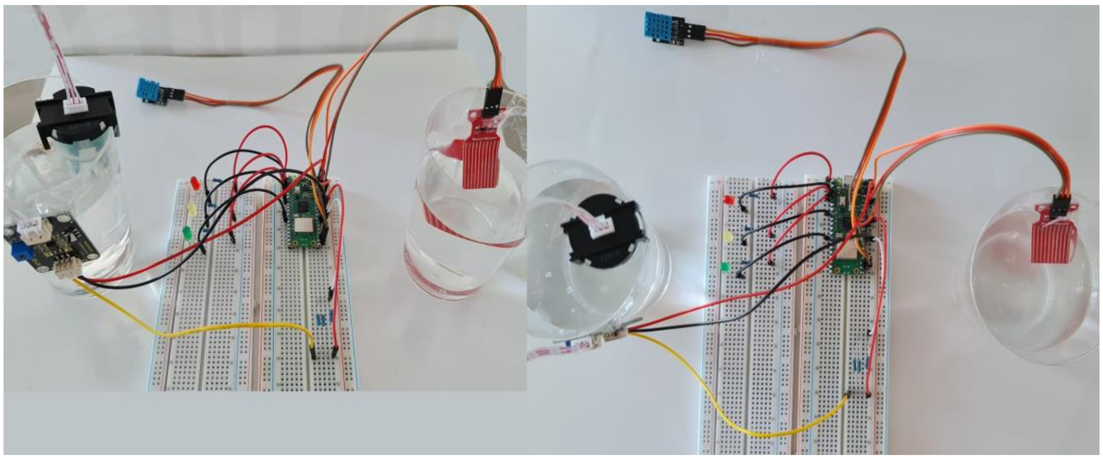


Figure 13 Componenets Connection

Figure 14 shows the LEDs' connection, which is connected through a male-to-male jumper. Each LED have 10k resistor to limit the electricity flow for the LED's.

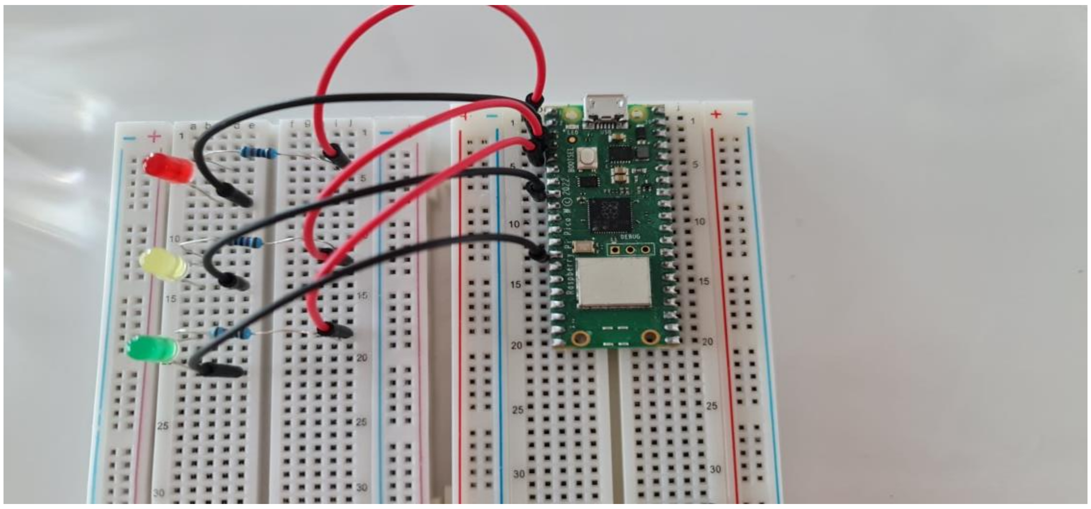


Figure 14 LED's Connection

Figure 15 shows the connection between the Temperature and Humidity sensor with the microcontroller, and the connection is made through male-to-female jumpers.

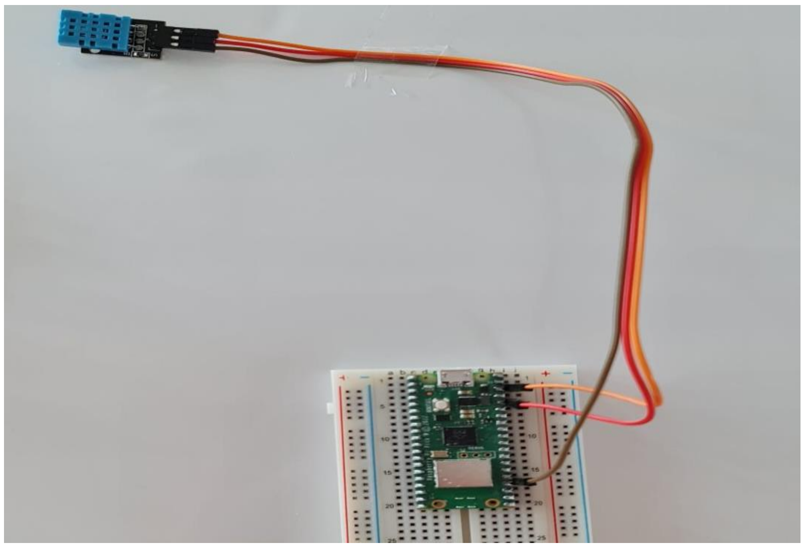


Figure 15 Temperature and Humidity Connection

Figure 16 shows the connection between the Water level sensor and the microcontroller, the connection is made through male-to-female jumpers.

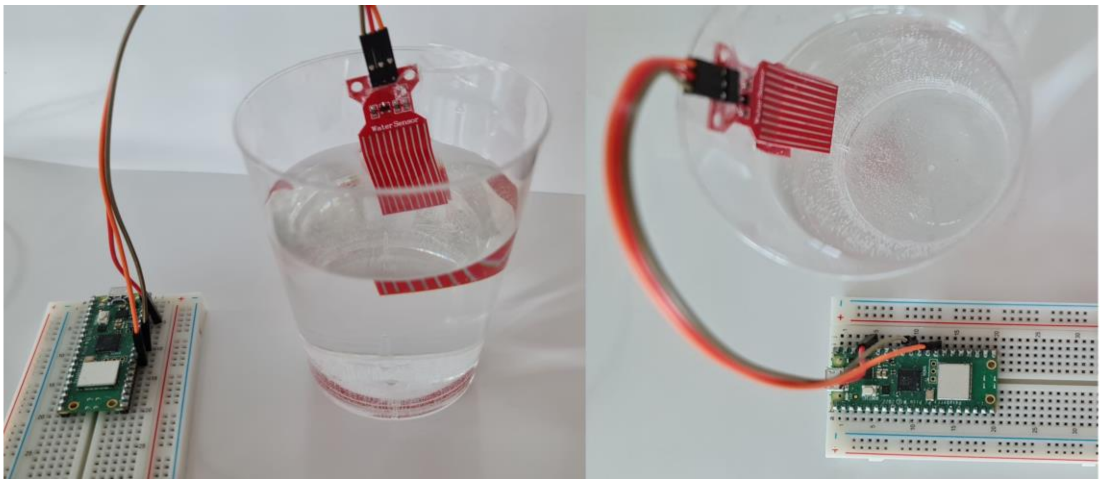


Figure 16 Water level Connection

The Turbidity sensor comes with jumpers with small clips that fit into the circuit connectors, male-to-male jumpers are used for extra length and easy access, and three 1K resistors to regulate the circuit, as in Figure 17.

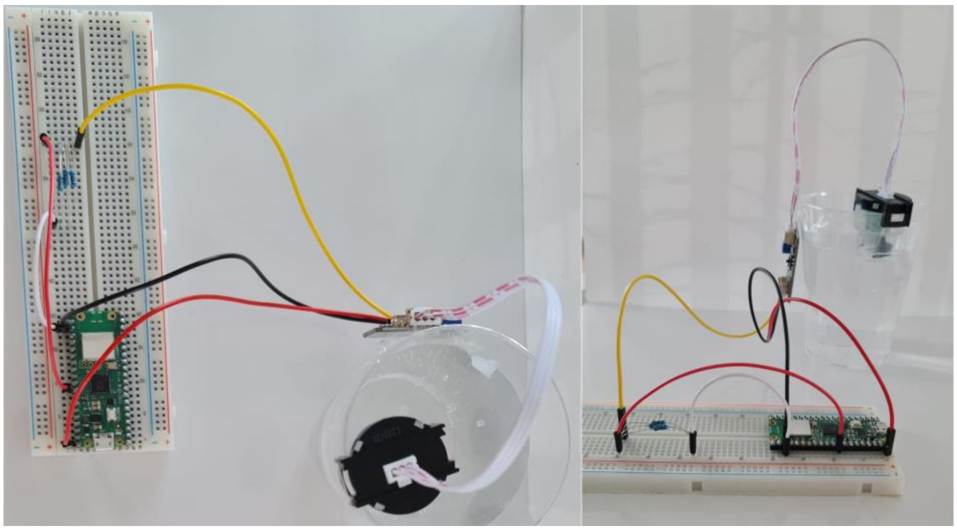


Figure 17 Turbidity Connection

4.3 Software Design

The software design entirely depends on the project needs that can be done with Raspberry Pi Pico. Pico is a microcontroller board that supports Micro Python, and can create various projects involving sensors, actuators, communication, and more projects. At first steps, setting up a Raspberry Pi Pico and designing the software involves several steps, as followings:

A. Hardware Setup: Obtain a Raspberry Pi Pico board, a micro-USB cable for programming or testing, and supply power.

B. Software Setup:

1. Install the Thonny IDE (Integrated Development Environment)(23) on the computer. Thonny is a beginner-friendly Python IDE and comes pre-installed with Raspbian (now known as Raspberry Pi OS).

   a. Select the Raspberry Pi Pico as the Interpreter: In Thonny, click on the "View" menu and choose "Interpreter." in the Interpreter window, a drop-down menu appears under "Interpreter" in the top-right corner. Click on it and select "MicroPython (Raspberry Pi Pico)".

b. Write and run MicroPython Code: Write or paste the MicroPython code into the Thonny editor. Click the "Run" button (a green arrow) in the top toolbar.
c. Monitor the Output: The code output will appear as print statements or error messages in the "Shell" tab at the bottom of the Thonny window.
d. Debugging: Thonny also supports debugging features like breakpoints, variable inspection, and Save Project.

2. Install the MicroPython Firmware: install the Micro Python firmware for Raspberry Pi Pico. as per the following steps:

- Downloading the latest MicroPython UF2 file for the Raspberry Pi Pico from the official MicroPython website: https://micropython.org/download/rp2-pico/ (24)
- Plug Raspberry Pi Pico W into the computer via USB while holding the BOOTSEL button, this will put it into bootloader mode.
- When the device appears as a drive, drag, and drop the downloaded UF2 file onto this drive. The Pico will reset itself, and MicroPython it will be installed.
- Also, the Micropythong firmware can be downloaded directly from Thonny(a free and open-source integrated development environment for Python).

3. Write and Upload code:

- By opening the Thonny IDE on the computer, select "Raspberry Pi Pico" as the interpreter by going to Tools > Options > Interpreter and choosing the Pico from the dropdown.
- Write sample Python code in Thonny.
- Running the code on the Raspberry Pi Pico. Thonny will automatically upload the code to Pico and execute it.

4.3.1 Coding Pico

Coding with the Raspberry Pi Pico involves writing code in a programming language supported by Pico's Micro Python firmware. Micro Python is a subset of Python 3 specifically optimized for microcontrollers. After installing the Thony IDE and Micro Python Firmware the first test needs to be done with a single LED connected to the GPIO Pin {25} on the Raspberry Pi Pico, as in the following listing 1:

```
import machine
import utime

led = machine.Pin(25, machine.Pin.OUT)
while True:
   led.toggle()
   utime.sleep(1)
```

Listing 1 First Pico test

This code imports the necessary libraries (machine for hardware access and time for utime-related functions). It sets up Pin 25 as an output pin and then toggles the state of the LED (ON and OFF) every second in an infinite loop using the {utime.sleep()} function.

As explained previously there are three LEDs and three sensors connected to the microcontroller, every component needs to be coded separately, and then putting all the codes together to have a fully functional prototype, the next step is calibration and testing for each sensor.

1. **LED's Coding**: the LED coding is to show the status of the EG, which represents the on, off, and restart systems. The idea is that when the EG is off, the status will show (Standby), and the red LED will turn on. When EG runs, the user interface will receive a notification that EG is (Running). When EG is stopped due to overload, overheating, under-

speed, etc., the user interface will receive a notification as (Restart), and the yellow LED will turn on.

```
# Initialize the LED pins (modify the GPIO pin numbers as needed)
RED_PIN = 3
GREEN_PIN = 1
YELLOW_PIN = 0
red_led = machine.Pin(RED_PIN, machine.Pin.OUT)
green_led = machine.Pin(GREEN_PIN, machine.Pin.OUT)
yellow_led = machine.Pin(YELLOW_PIN, machine.Pin.OUT)
# Function to control the LEDs based on the generator status
def control_leds(status):
    if status == "stand-by":
        red_led.on()
        green_led.off()
        yellow_led.off()
    elif status == "running":
        red_led.off()
        green_led.on()
        yellow_led.off()
    elif status == "restart":
        red_led.off()
        green_led.off()
        yellow_led.on()
    else:
        print("Invalid generator status.")
```

Listing 2 LED's setup

- **Initialize LED Pins**: GPIO pins are initialized for red, green, and yellow LEDs connected to the microcontroller (Raspberry Pi Pico W). The PINs are defined as constants RED_PIN, GREEN_PIN, and YELLOW_PIN. The machine. Pin class is to set up the pins as output pins (machine.Pin.OUT), which means these pins will send signals to the LEDs.

- **Control LEDs with control_leds Function:** The control_leds function is defined to control the LEDs based on the received generator status. It takes one argument(status), which should be one of three strings: "standby", "running", or "restart". Depending on the status received, it turns on or off the appropriate LEDs.

a) If the status is "standby", it turns on the red LED and turns off the green and yellow LEDs.
b) If the status is "running", it turns off the red and yellow LEDs while turning on the green LED.
c) If the status is "restart", it turns off the red and green LEDs and turns on the yellow LED.

d) If the status is error, it prints "Invalid generator status".

2. **Temperature and Humidity Sensor Coding:** The sensor will collect the temperature from surroundings and will deliver back to the microcontroller to show the value in the user interface. The sensor cannot receive any data from the microcontroller. The code reads temperature and humidity data from the DHT11 sensor, prints it to the console, and continues to do so in an infinite loop. If there are any errors while reading the sensor, it prints an error message.

```
# Initialize DHT11 sensor on GPIO pin 18
dht_sensor = DHT11(Pin(18))
while True:
    try:
        # Read temperature and humidity
        dht_sensor.measure()
        temperature = dht_sensor.temperature()
        humidity = dht_sensor.humidity()
```

```
# Printing temperature and humidity
 print("Temperature: {}°C".format(temperature))
print("Humidity: {}%".format(humidity))
# Wait for 5 seconds before the next reading
utime.sleep(5)
# Printing error-e in case of any error and code will be blocked in (exception)
    except Exception as e:
        print("Error:", e)
```

Listing 3 Temperature and Humidity sensor setup

3. **Water Level Sensor Coding:** the sensor will be installed in the water tank of the EG so that the sensor can have constant water level data. This data is as vital as the temperature data, as the water tank keeps the engine coolant running.

   The coding of the sensor is designed to collect data from the sensor and be received by the microcontroller and the user interface app.

```
# Define the GPIO pin of the Water Level sensor
FLOAT_SWITCH_PIN = 26
# Initialize the GPIO pin
float_switch = Pin(FLOAT_SWITCH_PIN, Pin.IN)
# Define the threshold values for water levels
LOW_THRESHOLD = 0.5
HIGH_THRESHOLD = 2
MEDIUM_THRESHOLD = 1

def read_water_level():
    water_level = float_switch.value()
    if water_level < LOW_THRESHOLD:
        return "Water level is low"
    elif water_level < MEDIUM_THRESHOLD:
        return "Water level is medium"
    elif water_level < HIGH_THRESHOLD:
        return "Water level is high"
    else:
        return "Water level is very high"
```

```
# Main loop
try:
    while True:
        # Read and print the water level
        level = read_water_level()
        print(level)
        # Delay before reading again
        utime.sleep(5)
# Printing interruption code in case of any error and
code will be blocked in (exception)
except KeyboardInterrupt:
    print("Program interrupted by user")
```

Listing 4 Water Level sensor setup

4. **Turbidity Sensor Coding:** The most crucial part is that the sensor needs manual calibration to avoid the low-level density and viscosity of the engine oil. Then coding the sensor is needed to receive the data and send it to the user interface. The level of quality will be read as over 100% for high quality and under 70% for low quality.

   The code continuously reads analogue data from an analogue sensor, calculates the voltage, and determines the percentage of the sensor's position within a specified range. It then prints these values to the console with a 1-second interval between readings. It helps monitor and analyse data from analogue sensors.

```
from machine import ADC, Pin
import utime

in_pin = ADC(26)
conversion_factor = 3.3 / 65535
low, high = 750, 34900

while True:
    raw = in_pin.read_u16()
    volts = raw * conversion_factor
    percentage = (int(((raw - low) * 100) / (high - low)))
    print('Raw: {} '.format(raw), 'Voltage {:.1f}V'.format(volts),'Percentage: {}%'.format(percentage))
    utime.sleep(1)
```

Listing 5 Turbidity sensor coding

5. Wi-Fi Connection Coding: The coding essentially imports necessary modules that is more compatible with Pico, such as network and socket, then sets up the Wi-Fi credentials (SSID and password) for the network connection, where it prints the IP address obtained after successfully connecting to the Wi-Fi network and prepares a server to handle incoming requests on a specified IP address '0.0.0.0' and port 80. Moreover, it prints information about the server, specifically the address that is listening from.

```
import network
import socket
# Wi-Fi credentials
ssid = 'DNA-Mokkula-2G-8Q6596'
password = '51307104019'
# Set up Wi-Fi connection
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)
while not wlan.isconnected():
    pass
print('Wi-Fi connected. IP:', wlan.ifconfig()[0])
```

```
# Set up the server
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
s = socket.socket()
s.bind(addr)
s.listen(1)
print('Listening on', addr)
```

Listing 6Wi-Fi Connection Coding

4.3.2 User Interface

A user interface (UI) is the interactive part of a software application or system that allows users to interact with and control the application. It serves as a bridge between the user and the underlying software, enabling users to communicate their intentions and receive feedback from the system. The user interface encompasses various elements, including:

1. Graphical User Interface (GUI): Users can interact with the windows, icons, buttons, menus, and other graphical components, by using a mouse, touchpad, or touchscreen for the user interface.
2. Text-Based User Interface (TUI): Sometimes referred to a command-line interface, TUI provides an interactive way for users to interact with the application using text through a terminal or command prompt entering commands.
3. Web User Interface (Web UI): This type of interface is used for web-based applications and is displayed within a web browser. It involves web pages, forms, buttons, and other web elements that users can interact with.
4. Mobile User Interface (Mobile UI): This is specific to mobile applications and designed to work on smaller screens with touch-based input.
5. Voice User Interface (VUI): A VUI, which is frequently featured in voice assistants and bright gadgets enable users to interact with the application through voice commands and responses.
6. Gesture-Based User Interface: This interface type enables users to interact with the application through gestures such as swiping, tapping, and pinching, commonly used in touch screen devices.

A user interface's main objective is to provide simple, effective, and user-friendly interaction between the user and the software. A well-designed user interface provides clear, straightforward navigation, meaningful feedback, and a pleasant user experience.

In the context of the project, the user interface can refer to the HTML(Hypertext Markup Language) Web User interface, where users will be able to view data from the microcontroller (Raspberry Pi Pico) and show the status of the generator (e.g., on, off, restart).

The Raspberry Pi Pico has several challenges one is the communication between the Pico(MicroPython) code and the application (React Native) codes, and the memory capacity of Pico. As a result, the user interface will allow users to interact with the system and remotely manage the generator using Web user interface (HTML) and not a mobile application (Mobile UI).

UML diagram:

The UML diagram(Unified Modelling Language) shown in Figure 18 the design of the HTML user interface.

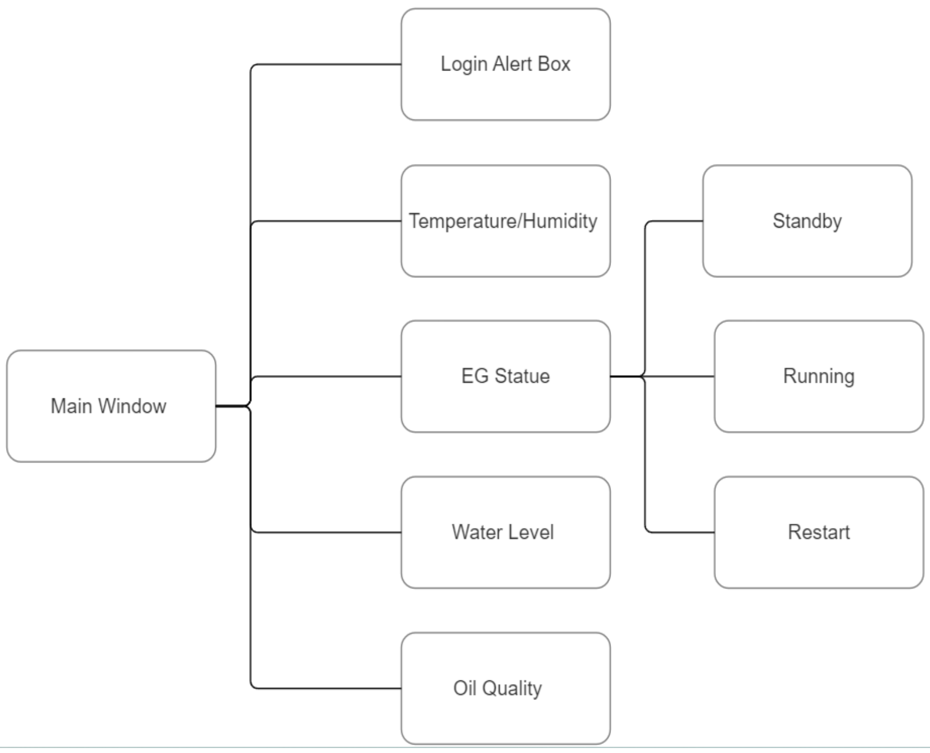
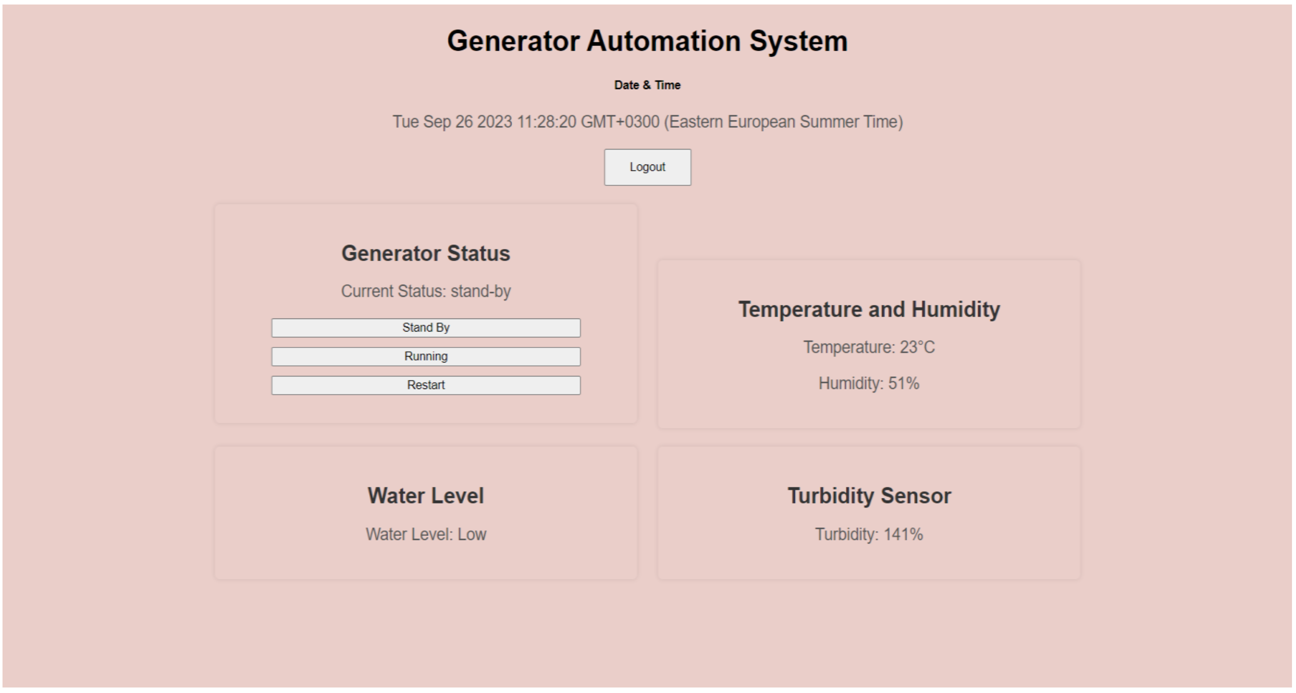


Figure 18 UML User Interface Diagrma

As per the UML diagram the user interface main window contain:

1. **Main Page:** the main page contains all the EG main controls and sensor data so the user can get updated status, temperature, humidity, water level, and Oil quality. Also, the main page can provide the actual date and time per the country/city zone.
2. **Login Alert Box:** in the main page a button is set to open an alert box where the user can write the credentials, also the alert box is verifying the username and password.
3. **EG Statues:** the EG statue contain updated statue such as (standby, running, and restart) as well as buttons that can change the status from one to another.
4. **Sensor's Control:** the sensors data will be updated automatically every 5 seconds so that the user can have the most updated data.


Generator Automation System

Date & Time

Tue Sep 26 2023 11:28:20 GMT+0300 (Eastern European Summer Time)

Generator Status

Current Status: stand-by

Temperature and Humidity

Temperature: 23°C

Humidity: 51%

Water Level

Water Level: Low

Turbidity Sensor

Turbidity: 141%

Figure 18 User Interface Main page

Content of the Main window coding and design:

1. Page Styling: the page styling is designed considering font and text styling, background colour, and paragraph styling on the main page body. Separating the contents by boxes, specify the borders, width, text alignment, box shadow, merging, display style, also giving styles to buttons.

```
<style>
body {
        font-family: Arial, sans-serif;
        margin: 0;
        padding: 0;
        background-color: #EACEC9;
        text-align: center;     }
.box {
        border-radius: 5px;
        padding: 20px;
        box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
        margin: 10px;
        text-align: center;
        display: inline-block;
        width: 30%;     }
```

```
button {
        display: block;
        width: 80%;
        margin: 10px auto;      }
    h2 {
        color: #333;
       font-size: 24px;
        margin-bottom: 10px;      }
    p {
        font-size: 18px;
        color: #555;       }
    #loginButton {
        width: 100px;
        height: 40px;       }
</style>
```

Listing 7 Style of the Main page

2. Functions:
   a) Login Authentication Function: The function manages a simple login/logout functionality using browser local storage and prompts for username and password. If the user ID is correct, it logs in, and if not, it prompts for credentials. As well as it allows the user to log out.

```
function toggleLogin() {
        const loggedIn = localStorage.getItem('loggedIn');
        if (loggedIn) {
          if (confirm('Are you sure you want to logout?')) {
            localStorage.removeItem('loggedIn');
            document.getElementById('loginButton').innerText = 'Login';
          }
        } else {
          const username = prompt('Username:');
          const password = prompt('Password:');
          if (username && password) {
            localStorage.setItem('loggedIn', 'true');
            document.getElementById('loginButton').innerText = 'Logout';
            alert('Login successful!');
          } else {
            alert('Login failed. Please try again.');      }          }       }
```

Listing 8 Login Authentication Function

b) EG Statue Function: The function is designed to update the displayed status on the web page and make an asynchronous HTTP GET request to a server to update the generator's status. It logs messages to the console for debugging and handles success or error scenarios when communicating with the server. The server's response is examined to determine whether the update was successful or if an error occurred.

```
function updateGeneratorStatus(status) {
          console.log("updateGeneratorStatus called with status:", status);
          document.getElementById("status").textContent = "Current Status: " + status;
          var xhr = new XMLHttpRequest();
          xhr.open("GET", "/generator_status?status=" + status, true);
          xhr.onload = function() {
            console.log("Response status:", xhr.status); // Log the response status
            if (xhr.status === 200) {
              console.log("Generator status updated successfully.");
            } else {
              console.error("Error updating generator status. Response text:", xhr.responseText);
            }
          };
          xhr.onerror = function() {
            onsole.error("Network error occurred.");
          };
          xhr.send();
          }
```

Listing 9 Statue Function

c) Temperature and Humidity Function: The code establishes a mechanism for fetching temperature and humidity data from the sensor at regular intervals and updating the corresponding HTML elements on a web page with the latest data. The initial call ensures that data is displayed when the page loads. The (setInterval) function ensures that data updates continue every 5 seconds, providing a real-time or periodically updated display.

```
        function updateTemperatureAndHumidity() {
          var xhr = new XMLHttpRequest();
          xhr.open("GET", "/temperature_and_humidity", true);
          xhr.onload = function() {
            var data = xhr.responseText.split(",");
            var temperatureElement = document.getElementById("temperature-value");
            var humidityElement = document.getElementById("humidity-value");
            temperatureElement.innerText = data[0];
            humidityElement.innerText = data[1];
          };
          xhr.send();
        }
// Call the updateTemperatureAndHumidity function initially
        updateTemperatureAndHumidity();
// Call the updateTemperatureAndHumidity function every 5 seconds
        setInterval(updateTemperatureAndHumidity, 5000);
```

Listing 10 Temperature and Humidity function

d) Water Level Function: Creates a new Request variable, configures the variable to send an asynchronous GET request to the "/water level" . Moreover, it sets up an event handler for the variable load event. It retrieves an HTML element with the ID "water-level-status." and updates the text content of this HTML element with the response text received from the sensor.
   (updateWaterLevel();) ensures that the water level data is fetched and displayed on the web page as soon as it loads. (setInterval) calling the updateWaterLevel() function every 5 seconds (5000 milliseconds), which can be changed per user request and necessity, this creates a real-time or periodic data update mechanism on the webpage.

```
function updateWaterLevel() {
   var xhr = new XMLHttpRequest();
   xhr.open("GET", "/waterlevel", true);
   xhr.onload = function() {
      var waterLevelElement = document.getElementById("water-level-status");
      waterLevelElement.innerText = "Water Level: " + xhr.responseText; };
   xhr.send();
  }
updateWaterLevel();
setInterval(updateWaterLevel, 5000);
```

Listing 11 Water Level function

e) Turbidity Function: The function code establishes a mechanism for regularly fetching Turbidity data from the sensor and updating the corresponding HTML element on the web page with the latest Oil quality status. The initial call ensures that data is displayed when the page loads and the setInterval function ensures that data updates continue every 5 seconds, providing a real-time or periodically updated display.

```
function updateTurbiditySensor() {
          var xhr = new XMLHttpRequest();
          xhr.open("GET", "/turbidity_sensor", true);
          xhr.onload = function() {
             var turbidityElement = document.getElementById("turbidity-value");
             turbidityElement.innerText = "Turbidity: " + xhr.responseText + "%";
          };
          xhr.send();
      }
     // Call the updateTurbiditySensor function initially
      updateTurbiditySensor();
    // Call the updateTurbiditySensor function every 5 seconds
      setInterval(updateTurbiditySensor, 5000);
```

Listing 12 Oil Quality function

3. Main Page Body: Header with the title "Generator Automation System". And secondary header that displays the current date and time with JavaScript code to fetch the current date and time (const d = new date

();) and update the content of the element with the ID "demo" and the current date and time. The main page contents are grouped with (<div class="box">).

- The first box section for displaying the generator status initially displays "Loading..." and will be updated with JavaScript to show the current generator status. The user can interact with the generator status using <button> components. They update the generator status by calling JavaScript functions (updateGeneratorStatus) with various status values when clicked.
- The second box section for displaying temperature and humidity information where initially it displays "Loading..." will be updated with JavaScript to show the current temperature value.
- The third box section displaying water level information will be updated with JavaScript.
- Four box sections for displaying turbidity sensor information.

```
<body>
   <div id="header">
      <h1>Generator Automation System</h1>
   </div>
   <div id="header2">
      <h5>Date & Time</h5>
      <p id="demo"></p>
      <script>
         const d = new Date();
         document.getElementById("demo").innerHTML = d;
      </script>
   </div>
<button id="loginButton" onclick="toggleLogin()">Login</button>
   <div class="box">
      <div id="led-status">
         <h2>Generator Status</h2>
         <p><span id="status">Loading...</span></p>
            <button onclick="updateGeneratorStatus('stand-by')">Stand By
            </button>
            <button onclick="updateGeneratorStatus('running')">Running
            </button>
            <button onclick="updateGeneratorStatus('restart')">Restart
            </button>
</div>
</div>
<div class="box">
      <h2>Temperature and Humidity</h2>
      <p><span id="temperature-value">Loading...</span></p>
      <p><span id="humidity-value">Loading...</span></p>
   </div>
   <div class="box">
      <h2>Water Level</h2>
      <p><span id="water-level-status">Loading...</span></p>
   </div>
   <div class="box">
      <h2>Turbidity Sensor</h2>
      <p><span id="turbidity-value">Loading...</span></p>
   </div>
</body>
```

Listing 13 Main body HTML code

5 Testing

Testing is the systematic process of evaluating, verifying, and validating a product, system, or component to determine if it meets specified requirements and works as intended. Testing is critical in developing quality assurance for various products, including software applications, electronic devices, mechanical systems, sensors, etc.

The automation system prototype has been tested based on the following critical aspects of testing:

1. Evaluation: Testing involved examining the system to assess its functionality, performance, and adherence to predefined specifications and user requirements.
2. Verification: testing confirmed that the system has been designed and implemented correctly, ensuring that meets the intended purpose and design objectives.
3. Validation: Testing ensured the system meets the end-users needed and expectations.
4. Defect Identification: During testing, any issues, errors, or deviations from the desired behaviors are identified and documented as defects or bugs.
5. Improvement and Assurance: Offered insightful feedback for enhancing the performance, dependability, and quality of the system.
6. Documentation: In this chapter, testing activities are well-documented, including test plans, test cases, test results, and any defects found during the process.
7. Types of Testing: Various testing techniques carried out, including functional testing, performance testing, security testing, usability testing, and more, each serving a specific purpose in assessing various aspects of the system.

5.1 Hardware Manual Testing

Hardware manual testing involved, testing electronic hardware components, circuits, or systems to ensure proper functioning, performance, and reliability. Unlike automated testing, which relies on software scripts and tools, and manual testing involves physical hardware interaction and human intervention.

5.1.1 Raspberry Pi Pico Testing

Testing a Raspberry Pi Pico (RPi Pico W model) involved some coding and software testing as following steps:

1. Verification and Validation: During the preliminary test, the Raspberry Pi Pico W(RP 2040) meets the intended purpose and design objectives. The configurations for the Pico (pin assignments and power/data connector) are correct. It also meets the end user's needs and expectations as the project is a prototype, the connection to the network is done correctly, response time and processing speed are at an average level, and power consumption is as required. Raspberry Pi Pico W is compatible (and has been tested)with both analogue and digital sensors. Pico W is user-friendly and easy to work with as it uses both C/C++ and MicroPython languages.
2. Defect Identification: during the tests and performance of the project, the Pico gave some issues/bugs*, such as:
   - Length on the HTML code implementation: as in the project, HTML code has been used as a user interface. The demonstrated web page gave issues, such as during the implementation of (username, password) as login account for security reasons, the length of the code affected in the demonstration of the main page and some parts of the sections were disappearing, and this issue could not be fixed as the length of the main page code could not change because of the importance of the main page sections. In other words, Pico W can have limited code lines for HTML.

- Library limitation: Raspberry Pi Pico W have a limitation in libraries that affects the connectivity between Mobile applications and the microcontroller. The solution is using third-party mobile applications as user interfaces and not creating the required application.

5.1.2 Sensor Testing and Calibration

1. Sensor Testing: Sensor testing involves the evaluation of a sensor's performance under various conditions to determine if it meets specified requirements and standards. The goal is to ensure that the sensor provides accurate and consistent measurements.
   - Functional Testing: is to verify that the sensor operates as expected when exposed to its intended stimuli (e.g., temperature, pressure, light, motion).
   - Accuracy Testing: Measure the sensor's accuracy by comparing its readings to known reference values. This helps identify any calibration errors.
   - Precision Testing: Evaluate the sensor's ability to provide consistent and repeatable measurements. Repeated readings should closely match each other.
   - Sensitivity Testing: Determine the slightest change in input that the sensor can detect and accurately measure.
   - Linearity Testing: Assess how well the sensor's response follows a linear relationship with the input. Non-linear behavior may require correction.
   - Response Time Testing: Calculate how long the sensor reacts when the conditions change. For applications that need real-time data, this is crucial.
   - Stability and Drift Testing: Determine if the sensor's readings change over time when exposed to continuous operation or varying conditions.
   - Cross-Sensitivity Testing: Evaluate if the sensor responds to factors other than the intended stimulus (e.g., a temperature sensor reacting to humidity).

- Power Consumption Testing: Assess the energy consumption of the sensor, which is especially important for battery-powered devices.

Table 1 explains that the first test made on the sensors based on the testing conditions, before the calibration phase:

Table 1 First Sensor Testing before Calibration

N	Sensors Testing	Temperature & Humidity Sensor	Water Level Sensor	Turbidity Sensor	Note
1	Functional testing	Passed	Not passed	Not passed	
2	Accuracy testing	Passed	Not passed	Not passed	
3	Precision Testing	Passed	Not passed	Not passed	
4	Sensitivity testing	Passed	Not passed	Not passed	
5	Linearity testing	N/A			The sensors have no input, and they only have output (the sensors are only sender and not receivers)
6	Response Time	Passed	Passed	Passed	
7	Stability & Drift**	Passed	Not passed	Passed	
8	Cross Sensitivity	Passed	Not passed	Passed	
9	Power Consumption***	Low consumption of power 3.3V	Low to moderate consumption between 3.3V to 5.5V	Very High consumption of power 5.5V	

The table above shows testing carried out on different testing conditions using different data variables when stated not passed means it was not passed at first step later adjusted and aligned with different data's during calibration phase.

When testing Temperature and humidity sensor (passed) In above table means that during testing phase temperature was gained from surrounding and did not need calibration.

A. **Water level sensor:**

1. The water level sensor functioned correctly but did not provide the expected output and response, therefor the testing was not passed because the sensor had to be set up based on the values of the variables used for coding calibration.
2. In the accuracy and precision testing the water level sensor did not provide accurate , consistent, and repeatable measurements therefore the test counted as not pass.
3. The water level sensor did not accurately detect and respond to small changes or variation in water level measurement parameter, for that the sensitivity testing was counted as not passed.
4. Linearity testing was not applicable for the sensors.
5. All sensors responded within an acceptable and specific time frame.
6. The water level sensor (not passed) exhibited instability and significant changes in its output and behavior over time.
7. The sensor exhibited significant interference and response to factors other than the intended parameter.
8. Low to moderate consumption in water level sensor means that the sensor consumes the intended power and a bit higher than the power provided by the microcontroller(the microcontroller provides 3.3V but the water level sensor can consume between 3.3V to 5.5V).

B. **The turbidity sensor** :(not passed) for functionality, accuracy, precision and sensitivity testing in above table means when turbidity sensor was connected to power source, light flashed spontaneously, this resulted as not passed until it was adjusted manually. Power consumption testing is carried out on turbidity sensor and the sensor needed 5.5V power supply and this consider as high consumption.

2. Sensor Calibration

Calibration refers to adjusting or determining the accuracy of a measurement device or sensor by comparing its output to a known reference or standard. It involves establishing the relationship between the measured values and the actual values measured.

In this research both Water Level, and Turbidity sensor were calibrated to correct any inherent biases or imperfections in the sensor's output and provide reliable and consistent results and recommended to periodically recalibrate the sensor to maintain accuracy, significantly if environmental conditions or other factors change over time.

The water level sensor is calibrated by coding as in listing 4 water level sensor setup (p. 47) . In this process calibration setup made as following: when water level on the sensor reaches high level reads 2 and when it reaches middle it reads 1 and when very low reads 0.5 .

Turbidity sensor’s calibrated in two steps one by connecting and disconnecting the sensor. Secondly by calibrating through using different density liquids while manual calibration made on single connector board see (Figure 10) (p. 33). Then the Turbidity sensor submerged into the engine oil the sensor gave 140% clarity of the oil. To check on other non-clear liquids, another dark messy liquid was used to test the sensor. This result gave less than 30%to 70%.

5.2 Code Testing

Code testing in the context of prototype implementation with Micro Python and HTML coding involves systematically evaluating and verifying the correctness, functionality, and performance of the software code written in Micro Python (for the backend logic) and HTML (for the frontend user interface). Code testing aims to identify and fix any issues or bugs, ensure the code operates as expected, and validate that the prototype functions as intended.

Critical aspects of code testing in this case include:

1. Unit Testing: Test individual units or components of the code in isolation to ensure they perform as expected. For Micro Python, this involves

testing functions, methods, and classes, while in the case of HTML, entail verifying the correctness of individual HTML elements and their attributes.

2. Integration Testing: Test how different units of code work together to ensure seamless integration and proper communication between the Micro Python backend and HTML front.
3. Functional Testing: Verify that the prototype functions correctly and delivers the intended functionality. For example, this could involve checking if user inputs are processed correctly, data is stored or retrieved as expected, and the HTML user interface renders appropriately.
4. User Interface (UI) Testing: Ensure the HTML user interface is responsive across various devices and screen sizes, and user-friendly.
5. Compatibility Testing: Ensure the prototype works well with various browsers and platforms and that Micro Python code runs effectively on the target hardware or platform.
6. Performance Testing: Assess the prototype's speed, responsiveness, and resource usage to ensure it meets performance expectations.
7. Security Testing: Identify and address potential security vulnerabilities, such as input validation and data handling, to prevent security breaches.
8. Regression Testing: Regression analysis should be done when code changes are made to ensure the functionality is not adversely affected.
9. Usability Testing: Solicit user feedback to assess the prototype's usability and gather insights for improvements.

5.2.1 Step by Step Testing

A. Test Plan: The test plan sets the foundation for a structured and organized testing process, ensuring that the testing objectives are aligned with the project goals, risks are managed, and resources are optimally utilized to achieve successful testing outcomes, as shown in Table 2:

Table 2 Test Plan

Section	Content
Project Name and Information	EG Automation system Prototype
	The Generator Automation System aims to monitor and control various aspects of a generator system, ensuring its reliable operation. The system involves multiple components and functionalities
Objectives	• Provide a user-friendly interface for controlling the generator's status. • Ensure real-time monitoring of temperature and humidity levels. • Continuously monitor the water level and categorize it. • Integrate a turbidity sensor for assessing water quality. • Display all relevant data through a web-based dashboard.
Scope	• The project includes the development of software to control and monitor the generator, temperature, humidity, water level, and turbidity. • It involves the integration of various sensors and the creation of a web-based user interface. • The project's scope covers data presentation, user interaction, and sensor integration.
Schedule	Start 01/Aug/2023
	End 10/Sep/2023
Resources	Raspberry Pi Pico W, USB power cable, LED, Temperature and Humidity sensor, Water level sensor, and Turbidity sensor.
	MicroPython, HTML, CSS, Javascript
Test Environment	Hardware Setup
	Software Configuration
Test Deliverables	• A fully functional Generator Automation System with a user interface. • Producing test case, and test result reports.
Risks and Assumptions	• Technical challenges related to hardware and sensor integration. • Network or connectivity issues affecting data transmission. • Potential inaccuracies in sensor readings.
	• Assumption of accurate sensor data. • Availability of required hardware components. • The stability of the network infrastructure.
Dependencies	• Proper hardware components, including sensors and

	the generator system. • Stable network connectivity for data transmission. • Availability of development and testing environments.
Testing Approach	Section by section execution sequence

The above plan table provides an overview of the project which contains the project name and description. The plan objectives clearly state what the testing aims to achieve by specifying the goals and targets.

The test scope is to define the boundaries by inclusion (what will be tested) and exclusions (what will not be tested).

The resources were to identify the tools and equipment needed for the testing. Also, specifying the technical environments in which testing will occur, such as hardware, software, network configuration, and sensors.

In test deliverables, listing the expected outcomes and producing test cases and test result reports.

Risk management Identifies potential risks and assumptions to consider during testing, such as risks that might impact testing and assumptions made during planning.

The testing approach describes the overall testing methodology and criteria to deliver required results. All the tests have been done based on a certain schedule that provided a timeline for the testing phases, enabling efficient planning and execution of the testing activities.

B. Test Cases : As shown in Table 3, where the test objects been identified, a prediction, test steps, and test environment has been set for each objects.

Table 3 Test Cases

Test Case ID	Test Objective	Precondition	Test Steps	Expected Results	Test Environment	Postconditions
T01-1	Generator Status Control	Switch to "Stand By" Mode	Click the "Stand By" button on the user interface	The generator status should change to "Stand By," and the "Stand By" LED should turn on	Raspberry Pi (or equivalent device-LED) running the Generator Automation System simulator	The generator status is updated correctly, and the LEDs change their state accordingly.
T01-2	Generator Status Control	Switch to "Running" Mode	Click the "Running" button on the user interface	The generator status should change to "Running," and the "Running" LED should turn on	Raspberry Pi (or equivalent device-LED) running the Generator Automation System simulator	
T01-3	Generator Status Control	Switch to "Restart" Mode	Click the "Restart" button on the user interface	The generator status should change to "Restart" required, and the "Restart" LED should turn on	Raspberry Pi (or equivalent device-LED) running the Generator Automation System simulator	
T02-1	Temperature and Humidity Monitoring	Display Temperature and Humidity	(Automatic reading every 5 seconds)	The temperature and humidity values should be updated every 5 seconds, and accurate values should be displayed on the user interface	DHT11 temperature and humidity sensor	The temperature and humidity data are accurately displayed on the user interface
T03-1	Water Level Monitoring	Low Water Level	Simulate a low water level condition.	The system should detect the low water level and display "Low" on the user interface.	Water Level sensor	The water level condition (low, medium, high) is correctly detected and displayed on the user interface.
T03-2	Water Level Monitoring	Medium Water Level	Simulate a low water level condition.	The system should detect the medium water level and display "Medium" on the user interface.	Water Level sensor	
T03-3	Water Level Monitoring	High Water Level	Simulate a low water level condition.	The system should detect the high-water level and display "High" on the user interface.	Water Level sensor	

T04	Turbidity Sensor Integration	Display Turbidity Percentage	(Automatic reading every 5 seconds)	The turbidity percentage should update every 5 seconds, and accurate values should display on the user interface.	Turbidity sensor	The turbidity percentage is correctly integrated, and its values displayed on the user interface.
T05-1	User Authentication	Login with Correct Credentials	Enter the correct and incorrect username and password	The system should grant or deny access to the user, allowing them to control and monitor the system.	Web server software for hosting the user interface. Web browser (for accessing the user interface).	The system either grants or denies access based on the correctness of the entered credentials.
T05-2	User Authentication	Login with Correct Credentials	User login successful	The system should show login button as logout	Web server software for hosting the user interface. Web browser (for accessing the user interface).	

C. Test Results: the result of the testes executed after calibration of the sensors and implementing correct codes, and the teste result were as follow:

Test Case ID	**Test Execution Date**	**Test Execution Status**	**Actual Results**	**Notes**	**Defects or Issues**
T01-1	01-Aug	Pass	Passed	The test was successful and is working	
T01-2	01-Aug				
T01-3	01-Aug				
T02-1	10-Aug	Pass	Passed	The test was successful and is working	
T03-1	15-Aug	Pass	Passed	The test was successful and is working	
T03-2	15-Aug				
T03-3	15-Aug				
T04	21-Aug	Pass	Passed	The test was successful and is working	
T05-1	25-Aug	Pass	Passed	the test was successful	**Des.:** the button converts to logout, a basic login function used **Severity:** the level of impact is critical as its security. **Status:** solved.
T05-2	25-Aug				

Table 4 Test Result

In summary the test was successful in most parts of the prototype, all the sensors been calibrated, and results are sent back as required. The EG statue controls are well implemented as they are being represented by LED's. Temperature and humidity sensor are functioning properly (with only 2-degree difference, as this depends -on the reference temperature device and the area they been placed).

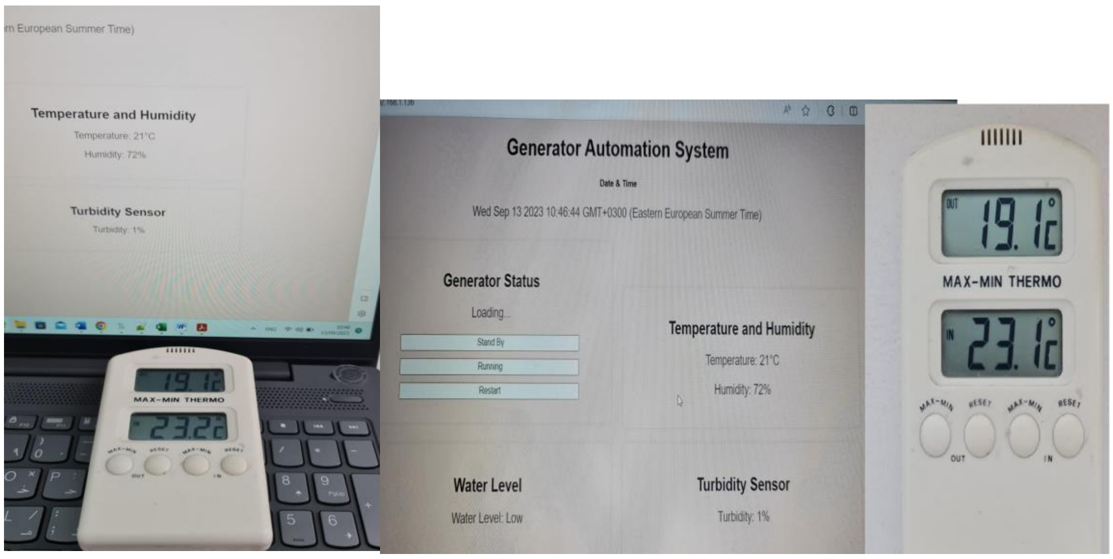


Figure 19 Temprature Refrencing

The water level sensor is an analogue sensor and the readings during tests are only (low and high), usually calibration is needed often.

During the testing phase, the user authentication was successful, when using basic login all operations are successfully loaded in the webpage and the login button on the main page changes from (Login) to (Logout) after successful login with specific username and password. But when using advanced login for user ID and authentication the web page loading was blocked , this is due to the HTML code length in Raspberry Pi Pico W and limited memory.

6 Summary and Conclusion

The motivation of this research was to upgrade and automate electrical power generators operations. Electrical generators are used locally for hospitals, buildings, education sector, businesses and homes in Iraq and neighbouring countries.

Iraq suffers continuous chronic electricity power shortage, this problem triggered by diverse factors such as climate change, population growth, and infrastructural inadequacies.

Government in Iraq failed to deal with power shortage in general, therefor the electricity power shortage is locally managed by individuals who owns electricity generators. These Electrical generators are significantly contributing to mitigating the electricity deficit in Iraq.

The background information helps contextualize the study and highlights the importance of the research topic. The journey through this thesis has encompassed an understanding of the brief historical background of electricity, and Iraq's electricity evolution.

This project finding has enhanced and upgraded the manually operated electricity power generators, to automated operation system and control of EG. These new findings will save operators time and effort.

This research describes and explains the design of the conceptual framework used in developing the prototype. It also includes the evaluation, verification, and implementation phases of the prototype.

The study also explains the hardware components selected for the prototype. Details of selected materials, circuits, sensors, and other relevant hardware aspects together with the system design, assembly, and configuration processes and described. Validation was carried out on the prototype, and it also covers code evaluation, step-by-step process, and software verification.

This project was a successful step to automate EG operation with the possibility to expand and upgrade to include different microcontrollers, more sensors, and mobile application user interface, though it is possible to add later more sensors such as engine oil level, vibration, load cell, strain gauges, speed sensors, proximity sensors, encoder sensor, and Hall Effect sensors. Also, by using different microcontroller, the advance user authentication and multiuser ID is possible.

In conclusion, the potential impact of this research extends beyond power generators in one city in Iraq, serving as a blueprint for similar regions grappling with electricity challenges.
This project enables more control over EG operation in term of time consuming and efforts. This project could be considered as an entrepreneur opportunity to set up a business to sell this application. For sure the first application will be installed at my home generator.

References

1. Ministry of planning. National Development Plan 2018-2022. 2018 p. 38.
2. Al-Maleki Y. Iraq Summer Power Shortages Grow Despite New CCGT. 2022 Jul 8;(65/27).
3. Iberdrola. A Brief History of electricity. 2022; Available from: https://www.iberdrolaa.com/sustainability/history-electricity
4. International Energy Agency. Iraq's energy sector: A roadmap to a brighter future [Internet]. OECD; 2019 [cited 2023 Sep 16]. Available from: https://www.oecd-ilibrary.org/energy/iraq-s-energy-sector_949e7e1e-en
5. Al-Hamadani S. Solar energy as a potential contributor to help bridge the gap between electricity supply and growing demand in Iraq: A review. Int J Adv Appl Sci. 2020 Dec 1;9(4):302.
6. Bansal R, editor. Handbook of Distributed Generation [Internet]. Cham: Springer International Publishing; 2017 [cited 2023 Sep 16]. Available from: http://link.springer.com/10.1007/978-3-319-51343-0
7. Muslim HN. Challenges and barriers in Iraq for solar PV generation: a review. Int J Energy Environ. 2019;10(3).
8. The ADE Power Generator Team. How does an Electricity Generator work? [Internet]. 2018 Spe. Available from: https://ade-power.com/blog/how-does-an-electric-generator-work
9. Sakr. Diesel Power Generating Set. Power Generation; Report No.: DS-SPG1000-C1-02-D.
10. Farzidayeri J, Bedekar V. Design of a V-Twin with Crank-Slider Mechanism Wind Energy Harvester Using Faraday's Law of Electromagnetic Induction for Powering Small Scale Electronic Devices. Energies. 2022 Aug 26;15(17):6215.
11. Grigsby LL. Electric Power Generation Transmission and Distribution. 2012. 768 p.
12. Krunal A M, Ankita A T, Pratik N B, Pradip K B, Bachwani ProfSA. Review on Different Microcontroller Boards Used in IoT. IJRASET [Internet]. 2022 Mar 1;(2321–9653). Available from: https://www.ijraset.com/research-paper/different-microcontroller-boards-used-in-iot
13. Raspberry Pi. Providers Raspberry Pi [Internet]. Available from: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
14. BeagleBoard. BeagleBone Black Boards Documentation [Internet]. Available from: https://docs.beagleboard.org/latest/boards/beaglebone/black/ch03.html
15. ODROID Wiki. Introduction of ODROID-XU4 [Internet]. Available from: https://wiki.odroid.com/odroid-xu4/odroid-xu4

16. Scherz P, Monk S. Practical Electronics for Inventors, Fourth Edition, 4th Edition. McGraw-Hill Education TAB; 2016.

17. Song K, Taghipour F, Mohseni M. Microorganisms inactivation by wavelength combinations of ultraviolet light-emitting diodes (UV-LEDs) [Internet]. 2019. Available from: https://pubmed.ncbi.nlm.nih.gov/30893742/

18. Happy heart32. Water level sensor DIY [Internet]. Available from: https://projecthub.arduino.cc/happyheart32/water-level-sensor-diy-ff331f

19. Arcaegecengiz. Using DHT11 [Internet]. 2023. Available from: https://projecthub.arduino.cc/arcaegecengiz/using-dht11-12f621

20. Team A. This remote sensor system determines changes in water quality [Internet]. 2023. Available from: https://blog.arduino.cc/2023/03/23/this-sensor-determines-if-water-is-good-for-drinking/?queryID=undefined

21. SACE A. Electrical installation handbook Protection, control and electrical devices. 6th ed. ABB SACE; 2010. 548 p.

22. Charras J, Tappero F, Evans J. KiCad 7.0 [Internet]. Available from: https://docs.kicad.org/7.0/en/kicad/kicad.html

23. Aivarannamaa. Thonny Instalation Guithub [Internet]. Available from: https://github.com/thonny/thonny/releases/tag/v4.1.1

24. Raspberry Pi. MicroPython Firmware [Internet]. 2023. Available from: https://micropython.org/download/RPI_PICO/

Table of Figures

Tables

Table of Source codes

Mobile Application

Mobile application is created and will be used as a user interface in the upgraded version. It was not used in this project because the Raspberry Pi Pico W did not connect to the application due to memory limitation.

The mobile application design and development were created with Expo Snack, which is an online code editor that enables developers to build and test React Native code in the browser. Without requiring the installation of any software, it can swiftly develop and test apps.

the designing of the application contains the following :

1. Raspberry Pi Pico W coding: the Pico was modified as per the mobile application coding and the requests from the react native. the code configures a microcontroller to serve as a primary HTTP server, handle specific API requests for sensor data, control LEDs based on generator status, and provide simulated or actual sensor data as responses. as following:

```
import ujson
import network
from machine import Pin, ADC
from dht import DHT11
import time
import random
import usocket as socket

# The Wi-Fi connection Set up, LEDs, DHT11, Water level, Turbidity sensors set up are the same as in the HTML Pico coding.

# The functions of the sensors are the same as in HTML Pico coding.

def http_server():
    addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(addr)
    s.listen(1)
    print('Listening on', addr)
    while True:
        cl, addr = s.accept()
        print('Client connected from', addr)
        request = cl.recv(1024)
        request = request.decode('utf-8')
        print('Received request:', request)
        if '/generator_status' in request:
            print('Received generator status request')
            path, query = request.split('?')
            status = query.split('status=')[1].split(' ')[0]
            print("Received status:", status)
            control_leds(status)
            generator_status = status
            response = 'HTTP/1.1 200 OK\r\nContent-Type:
text/plain\r\n\r\n' + generator_status
            cl.send(response.encode())
        elif '/temperature_and_humidity' in request:
            temperature, humidity = read_temperature_and_humidity()
            response = {'temperature': temperature, 'humidity': humidity}
            response_json = ujson.dumps(response)
            print("Temperature and Humidity Sent:", response)
            cl.send(response_json.encode())
        elif '/waterlevel' in request:
            water_level = read_water_level()
            response_json = ujson.dumps(water_level)
            cl.send(response_json.encode())
            print("Water Level Sent:", water_level)
```

```
elif '/turbidity_sensor' in request:
            turbidity_percentage = read_turbidity_sensor()
            response_json = ujson.dumps(turbidity_percentage)
            cl.send(response_json.encode())
            print("Turbidity Sent:", turbidity_percentage)

        else:
            response_data = {}  # Create an empty response
            response_json = ujson.dumps(response_data)
            response = 'HTTP/1.1 200 OK\r\nContent-Type:
application/json\r\n\r\n' + response_json
            cl.send(response.encode())
            cl.close()
            print('Connection closed')

try:
    http_server()
except Exception as e:
    print('Error:', e)
```

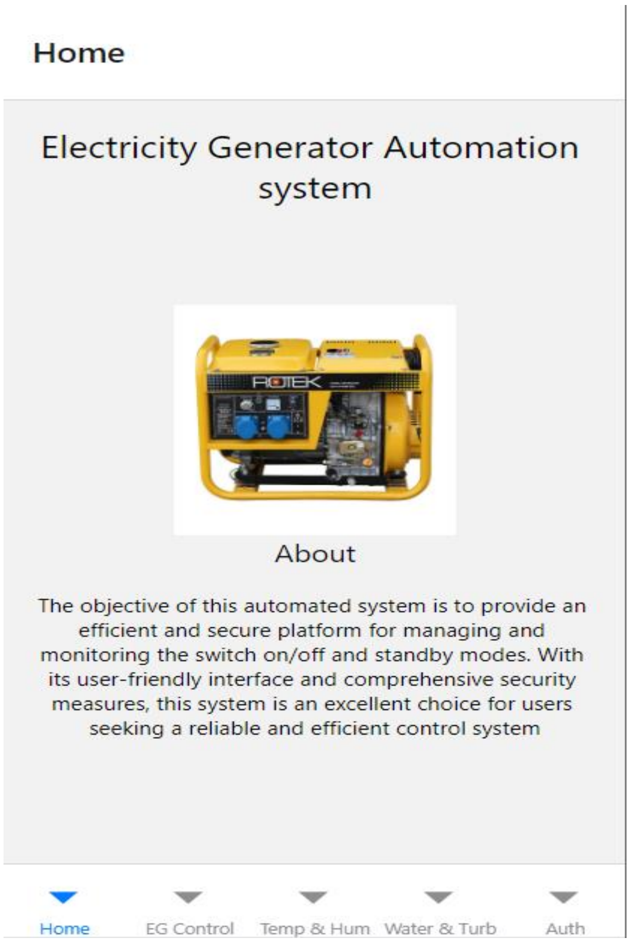
Listing 14 Pico HTTP request coding

2. The App.js file in a React Native application is the entry point and central configuration file. t typically initializes the app, sets up navigation, and defines the overall structure of the application.

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import AboutPage from './AboutPage';
import LEDControlPage from './LEDControlPanel';
import TempHumidityPage from './TempHumidityPage';
import WaterTurbidityPage from './WaterTurbidityPage';
import AuthPage from './AuthPage';
const Tab = createBottomTabNavigator();
const App = () => {
  return (
    <NavigationContainer>
      <Tab.Navigator>
        <Tab.Screen name="Home" component={AboutPage} />
        <Tab.Screen name="EG Control" component={LEDControlPage} />
        <Tab.Screen name="Temp & Hum" component={TempHumidityPage} />
        <Tab.Screen name="Water & E.Oil" component={WaterTurbidityPage} />
        <Tab.Screen name="Login" component={AuthPage} />
      </Tab.Navigator>
    </NavigationContainer>
  );
};
export default App;
```

Listing 15 App.js coding

3. Home Page: The home page or landing page is the main page containing a short description and aim of the app. AboutPage.js is where the illustration picture, page title, and short description has been created. As shown in Figure 21,



Electricity Generator Automation system

About

The objective of this automated system is to provide an efficient and secure platform for managing and monitoring the switch on/off and standby modes. With its user-friendly interface and comprehensive security measures, this system is an excellent choice for users seeking a reliable and efficient control system

Figure 20 Application Home Page

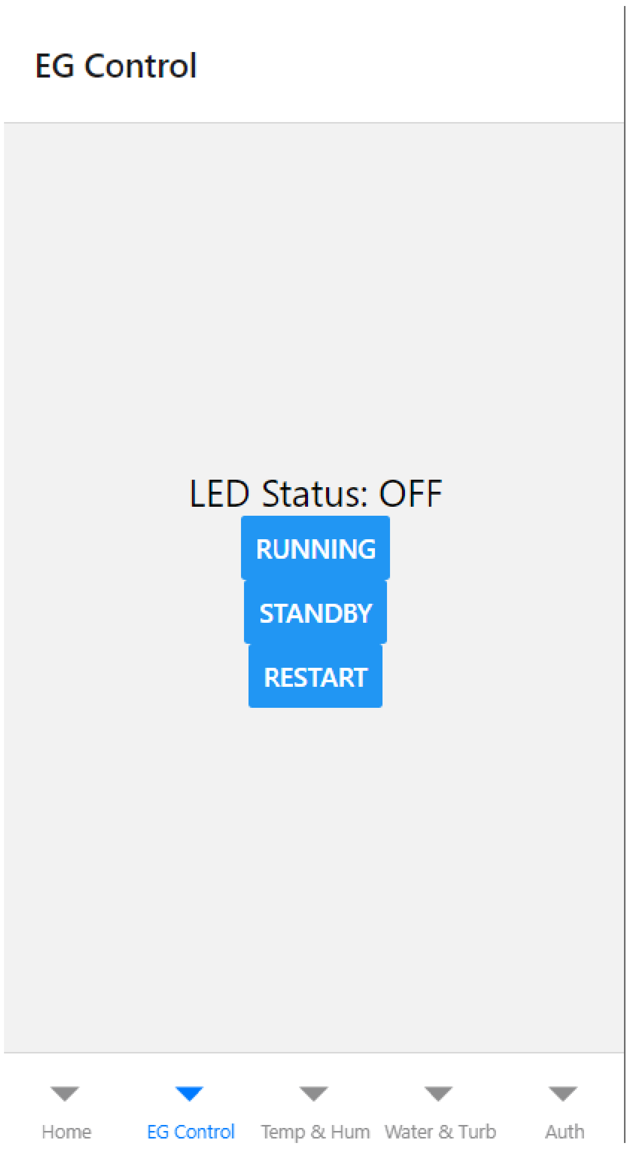
```
import React from 'react';
import { View, Text, SafeAreaView, Image } from 'react-native';
const AboutPage = () => {
  return (
    <SafeAreaView style={{ flex: 1 }}>
      <View style={{ flex: 1 , padding: 16}}>
      <Text style={{ fontSize: 20, textAlign: 'center', marginBottom: 16 }}>
            Electricity Generator Automation system
      </Text>
        <View style={{flex: 1, alignItems: 'center', justifyContent: 'center' }}>

          <View style={{ justifyContent: "center", alignItems: "center" }}>
            <Image
              source={require('./Pages/EG2.jpg')}
              style={{width: 150, height: 150}}
            />
            <Text style={{ fontSize: 16, textAlign: 'center', marginBottom: 16 }}>
              About
            </Text>
          </View>

          <Text style={{ fontSize: 12, textAlign: 'center', marginBottom: 16 }}>
          The objective of this automated system is to provide an efficient and secure
platform for managing and monitoring the switch on/off and standby modes. With its user-
friendly interface and comprehensive security measures, this system is an excellent choice
for users seeking a reliable and efficient control system
          </Text>
        </View>
      </View>
    </SafeAreaView>
  );
};
export default AboutPage;
```

Listing 16 AboutPage.js Main Home Page coding

4. EG Control Page: the tab contains the statue of the electricity generator where there are buttons to change from a statue to another as shown in the Figure 22:



LED Status: OFF

Figure 21 EG Control Page

```
import React, { useState, useEffect } from 'react';
import { View, Text, Button } from 'react-native';
import {EGStyles} from './Mystyles';
const LEDControlPage = () => {
  const [ledStatus, setLedStatus] = useState('Loading...');
  const fetchLedStatus = () => {
    console.log('Fetching LED status...');
    // Simulate response for demonstration purposes
    const randomStatus = Math.random() < 0.5 ? 'ON' : 'OFF';
    setLedStatus(randomStatus);
  };
  const handleLEDControl = async (status) => {
  try {
    const response = await fetch(`http://192.168.1.136:80/generator_status?status=${status}`);
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.text();
    console.log('LED Control Response:', data);
    // Fetch LED status after control
    fetchLedStatus();
  } catch (error) {
    console.error('Error updating LED status:', error);
    setLedStatus('Error');
  }
};
  useEffect(() => {
    // Fetch initial LED status when the component mounts
    fetchLedStatus();
  }, []);
  return (
    <View style={EGStyles.container}>
      <Text style={EGStyles.text}>LED Status: {ledStatus}</Text>
      <Button title="Running" onPress={() => handleLEDControl('running')} />
      <Button title="Standby" onPress={() => handleLEDControl('stand-by')} />
      <Button title="Restart" onPress={() => handleLEDControl('restart')} />
    </View>
  );
};
export default LEDControlPage;
```

Listing 17 EG Control Page Coding

5. Temperature and Humidity Page: The tab display temperature and humidity data of the inside shield of the EG and a button to refresh the page to get updated data when needed.

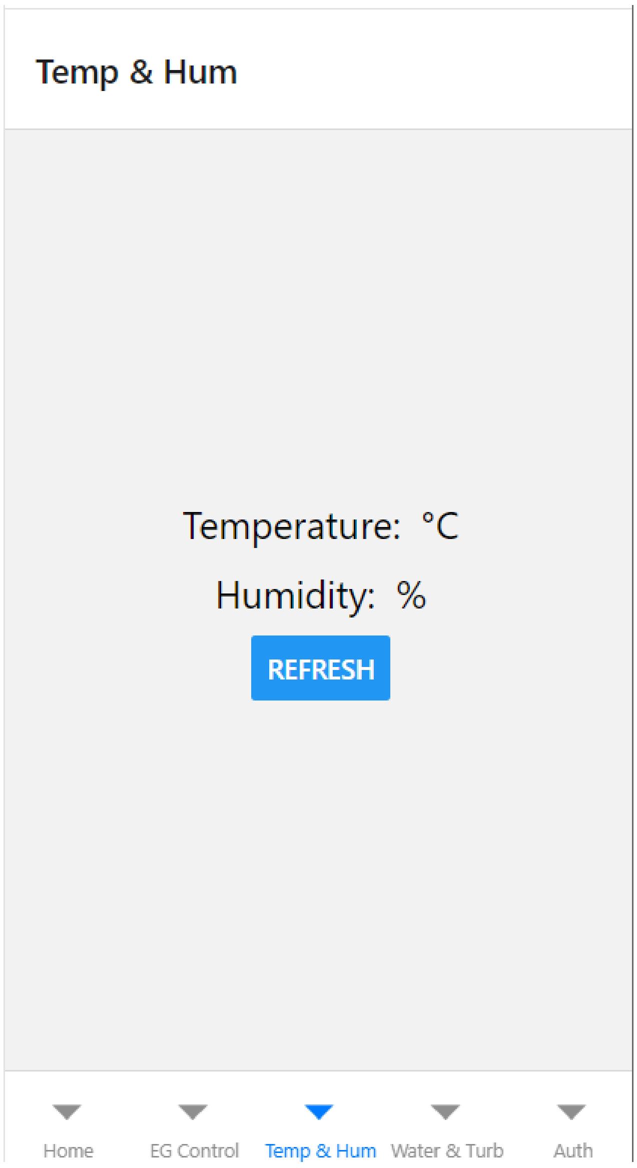


Figure 22 Temperature and Humidity Page

```
import React, { useState, useEffect } from 'react';
import { View, Text, Button } from 'react-native';
import {TempStyles} from './Mystyles';
import {fetchTemperatureAndHumidity} from './PicoAPI';
const TempHumidityPage = () => {
  const [temperature, setTemperature] = useState(null);
  const [humidity, setHumidity] = useState(null);
  const fetchData = async () => {
    try {
      const data = await fetchTemperatureAndHumidity();
      setTemperature(data.temperature);
      setHumidity(data.humidity);
    } catch (error) {
      console.error('Error fetching temperature and humidity:', error);
    }
  };
  useEffect(() => {
    fetchData();
  }, []);
  return (
    <View style={TempStyles.container}>
      <Text style={TempStyles.text}>Temperature: {temperature} °C</Text>
      <Text style={TempStyles.text}>Humidity: {humidity} %</Text>
      <Button title="Refresh" onPress={fetchData} />
    </View>
  );
};
export default TempHumidityPage;
```

Listing 18 Temperature and Humidity Page coding

6. Water level and Engin Oil Page: The tab shows real time water level and engine oil quality data, also contains button to refresh the page for more updated data. As shown in the following Figure 24:

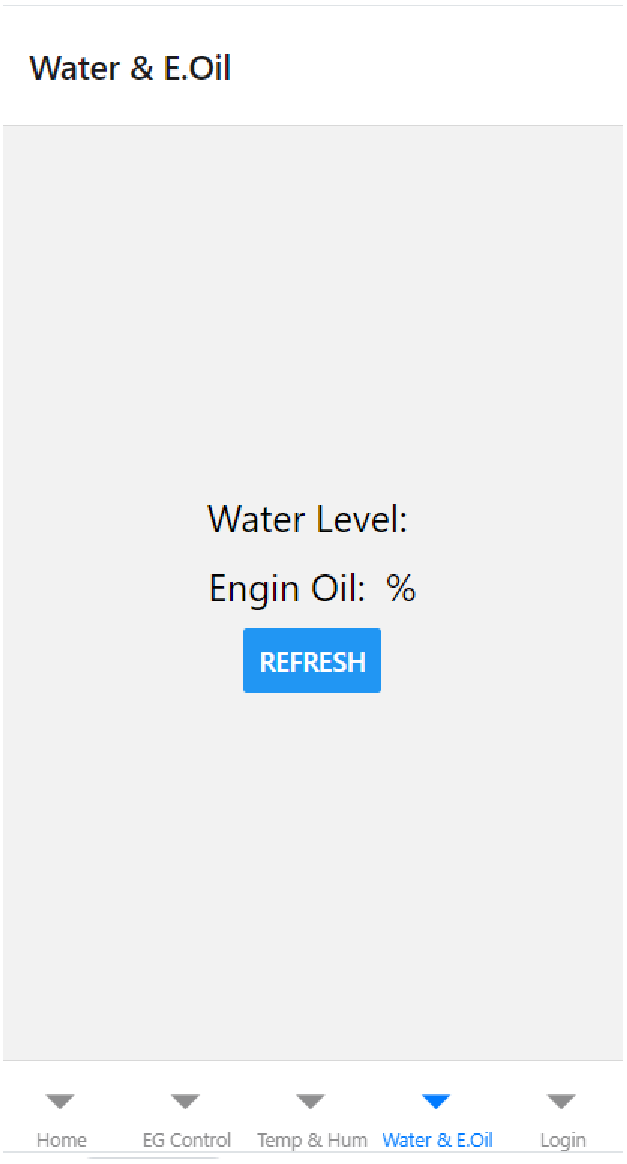


Figure 23 water and Engin Oil page

```
import React, { useState, useEffect } from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';
import {WaterStyles} from './Mystyles';
import { fetchWaterLevel, fetchTurbidity } from './PicoAPI';
const WaterTurbidityPage = () => {
  const [waterLevel, setWaterLevel] = useState(null);
  const [turbidity, setTurbidity] = useState(null);

  const fetchWaterLevelAndTurbidity = async () => {
    try {
      const waterLevelData = await fetchWaterLevel();
      setWaterLevel(waterLevelData);
      const turbidityData = await fetchTurbidity();
      setTurbidity(turbidityData);
    } catch (error) {
      console.error('Error fetching water level and turbidity:', error);
    }
  };
  useEffect(() => {
    // Fetch water level and turbidity data when the component mounts
    fetchWaterLevelAndTurbidity();
  }, []);
  return (
    <View style={WaterStyles.container}>
      <Text style={WaterStyles.text}>Water Level: {waterLevel}</Text>
      <Text style={WaterStyles.text}>Engin Oil: {turbidity} %</Text>
      <Button title="Refresh" onPress={fetchWaterLevelAndTurbidity} />
    </View>
  );
};
export default WaterTurbidityPage;
```

Listing 19 Water level and Engin Oil Page coding

7. The HTTP request function file (PicoAPI.js): the following code defines a set of functions to make asynchronous HTTP requests to a Raspberry Pi Pico's HTTP server for specific sensor data.

```javascript
const picoBaseUrl = 'http://192.168.1.136:80';
export const fetchTemperatureAndHumidity = async () => {
  try {
    const response = await fetch(`${picoBaseUrl}/temperature_and_humidity`);
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json(); // Assuming the response is in JSON format
    return data;
  } catch (error) {
    console.error('Error fetching temperature and humidity:', error);
    throw error;
  }
};
export const fetchWaterLevel = async () => {
  try {
    const response = await fetch(`${picoBaseUrl}/waterlevel`);
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json(); // Assuming the response is in JSON format
    return data;
  } catch (error) {
    console.error('Error fetching water level:', error);
    throw error;
  }
};
export const fetchTurbidity = async () => {
  try {
    const response = await fetch(`${picoBaseUrl}/turbidity_sensor`);
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json(); // Assuming the response is in JSON format
    return data;
  } catch (error) {
    console.error('Error fetching turbidity:', error);
    throw error;
  }
};
```

Listing 20 PicoAPI HTTP request coding

8. User Authentication page: A simple login and registration for security was created.

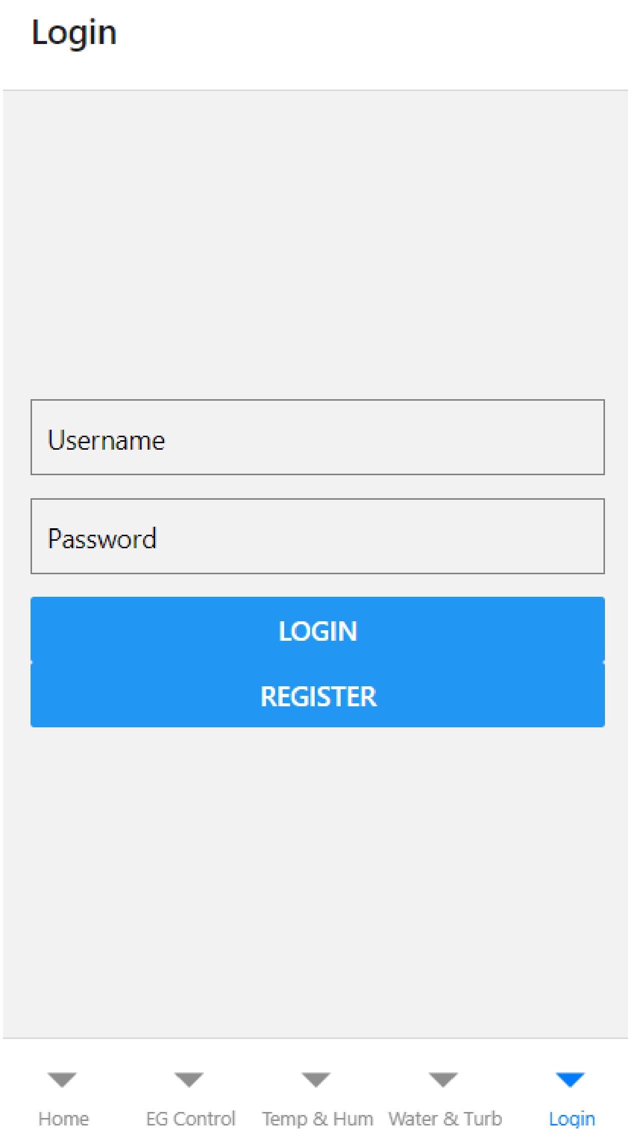


Figure 24 User Authentication Page

```
import React, { useState } from 'react';
import { View, TextInput, Button, StyleSheet, Alert } from 'react-native';
import { createStackNavigator } from '@react-navigation/stack';
const Stack = createStackNavigator();
const AuthPage = ({ navigation }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const handleLogin = () => {
    // Handle login logic based on username and password
    // For now, we'll just show an alert
    Alert.alert('Login', `Username: ${username}\nPassword: ${password}`);
    navigation.navigate('About');  // Navigate to About after successful login
  };
  const handleRegister = () => {
    // Handle registration logic based on username and password
    // For now, we'll just show an alert
    Alert.alert('Register', `Username: ${username}\nPassword: ${password}`);
    navigation.navigate('About');  // Navigate to About after successful registration
  };
  return (
    <View style={styles.container}>
      <TextInput
        style={styles.input}
        placeholder="Username"
        onChangeText={(text) => setUsername(text)}
      />
      <TextInput
        style={styles.input}
        placeholder="Password"
        secureTextEntry
        onChangeText={(text) => setPassword(text)}
      />
      <Button title="Login" onPress={handleLogin} />
      <Button title="Register" onPress={handleRegister} />
    </View>
  );
};
export default AuthPage;
```

Listing 21 Authentication Page coding