Project Report

Pool Automation

Author: CROCI Louis

Supervisor: Dr. Noel Murphy

14 January 2026

DCU
Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

Acknowledgements

I would like to thank Dr. Noel Murphy for his help, his invaluable advice, his patience, and the interest he showed in my project. I would also like to thank Mr Liam Lawlor for his help and supervision during my laboratory experiments. I also thank the DCU faculty for allowing me to carry out my project.

Abstract

This project focuses on the design and development of an automated and connected system for managing and monitoring essential pool parameters. The main objective is to reduce manual intervention while improving the reliability, safety, and efficiency of maintenance. The proposed solution is based on a hardware architecture using ESP32 microcontrollers, combined with a system of sensors for measuring pH, redox potential, temperature, and chemical levels.

The system integrates dosing pumps, electronic relays, and a local user interface via a touch screen, while ensuring remote communication using the MQTT protocol and integration with Home Assistant. Dedicated UART communication has been set up between the various modules to ensure a clear separation of functions. The testing and calibration phases validated the accuracy of the sensors and the consistency of the measurements, while highlighting the importance of rigorous risk management related to the simultaneous use of chemicals, water, and electricity.

The results obtained demonstrate the feasibility and relevance of an autonomous, scalable, and interoperable system for the intelligent management of a swimming pool, while providing a solid basis for future improvements focused on safety, robustness, and industrialization of the device.

Summary

I- Introduction

a - Project Origin

While maintaining my parents' pool, I thought it would be interesting to be able to monitor the metrics and the status of the pool regulations remotely. This would allow me to monitor the pool when I'm away, or simply when I leave my parents' house, enabling me to monitor it remotely.

Furthermore, being passionate about home automation and observing that pool maintenance requires considerable time due to the manual management of technical parameters such as pH, chlorine, temperature, and lighting, I decided to pursue this project. These tasks are not only repetitive and time-consuming, but they also carry risks associated with inadequate maintenance, which can degrade water quality and increase operating costs (chemicals, electricity, water).

This project aims to:

- Automate pool management to reduce manual intervention.
- Optimize costs (chemicals, energy, water).
- Integrate remote monitoring and control

b - Proposed Solution

I therefore decided to create a system capable of controlling and adjusting pH and chlorine levels, controlling the pool lighting and pump, and monitoring chemical levels in the chlorine and pH tanks. Having a strong interest in the HomeAssistant ecosystem and other home automation systems, I figured the solution should be able to communicate via MQTT to be compatible with as many home automation software programs as possible. The system allows:

- Real-time monitoring and adjustment of:
    - pH and Redox potential (chlorine levels).
    - Filtration time (adapted to usage and weather conditions).
    - Water level (automatic refill).
    - Lighting and auxiliary pumps.
    - Chemical container levels (chlorine, pH+).

- Integration with Home Assistant via MQTT (a lightweight IoT protocol) to:
    - Centralize data (historical trends, alerts, metrics).
    - Enable smart automation (e.g., adaptive filtration based on weather or usage).
    - Allow remote control (pump activation, target adjustments, metric monitoring).

Pool maintenance requires meticulous attention, particularly regarding chlorine and pH levels. The hardware independently manages critical functions (pH, chlorine, refill, pump) even without a Home Assistant connection, using embedded logic on the ESP32.

II - Technical Specifications

a - Hardware Components

Here is a table summarizing the components used for the hardware :

Component	Role	Interface/Protocol
pH Probe	Measures water pH.	Analog
Redox Probe	Measures Redox potential (indirect chlorine level)	Analog
Level Sensors x 4	Monitors chemical containers (chlorine, pH+)	Digital
Temperature Sensor	Measures water temperature.	I2C
Water Level Sensor	Measures water Level	Analog
Dosing Pumps x2	Automatically dispenses chlorine and pH corrector.	Relay
ESP32	Central controller: reads sensors, controls actuators, MQTT/Wi-Fi comms.	Wi-Fi/UART
ESP32 Touchscreen	Local UI for menus, settings, and alerts.	UART
Switches x4	Circulation pump. Refill solenoid valve. Auxiliary switches (lighting, heat pump, etc.).	Analog

The system is built around a central ESP32 controller, which reads sensors and controls actuators via analog, digital, and serial (UART/I^2C) interfaces. pH and ORP sensors measure water quality, while level and temperature sensors continuously monitor the status of the various tanks and the water itself. Dosing pumps and valves are controlled via relays to ensure automatic regulation of water chemistry. A touchscreen connected to the ESP32 provides a local user interface for configuration, alerts, and manual control. The entire system is designed to operate autonomously while also being able to communicate remotely via MQTT/Wi-Fi, enabling simplified monitoring and maintenance.

b - Software

To complete this project, I need to develop three separate programs: one for the ESP32 display in our solution, one for the ESP32 master, and one for the HomeAssistant add-on, which will retrieve data sent to the MQTT server and my PacketSerial-UART-ESP32 library.

1 - ESP32 Display

Developing the ESP32 screen software proved difficult, mainly due to a lack of documentation. Nevertheless, I found some tutorials that allowed me to integrate the LGVL library. This library will allow me to easily create a simple, fluid, and efficient interface. For development, I chose the C++ language, particularly for its object-oriented nature, which allowed me to build a small engine enabling me to easily manage the pages and different interfaces. This engine system also allows for easy evolution of the system, as each page is independent of the others. Since the LVGL library is a C library, integrating it into my code was easy; I first created a wrapper for LGVL in order to be able to use it correctly. The second part of this firmware results in communication with the master ESP in order to retrieve the information to be displayed or to communicate with it in order to transmit/interact with it.

To achieve this, I chose the UART protocol to allow communication between the ESP32 monitor and the master ESP32. This protocol is very simple; it only requires two wires (RX and TX) to send and receive information. UART is natively supported by the ESP32 and particularly well-suited to real-world systems. Therefore, I used the PacketSerial-UART-ESP32 library (which I created and will describe below) with a binary communication protocol.

You can view the code here : GitHub ESP SCREEN

2 - ESP32 Master

The role of this software is to establish the Wi-Fi connection, interact with the sensors, store the data necessary for the project's operation, and communicate with an MQTT server to share information via a protocol detailed later. It also communicates with the ESP32 display to interact with users via the UART detailed above. I also chose C++ as my programming language, because its object-oriented nature allows me to create classes related to the functions assigned to them, thus facilitating development.

You can view the code here : GitHub ESP MASTER

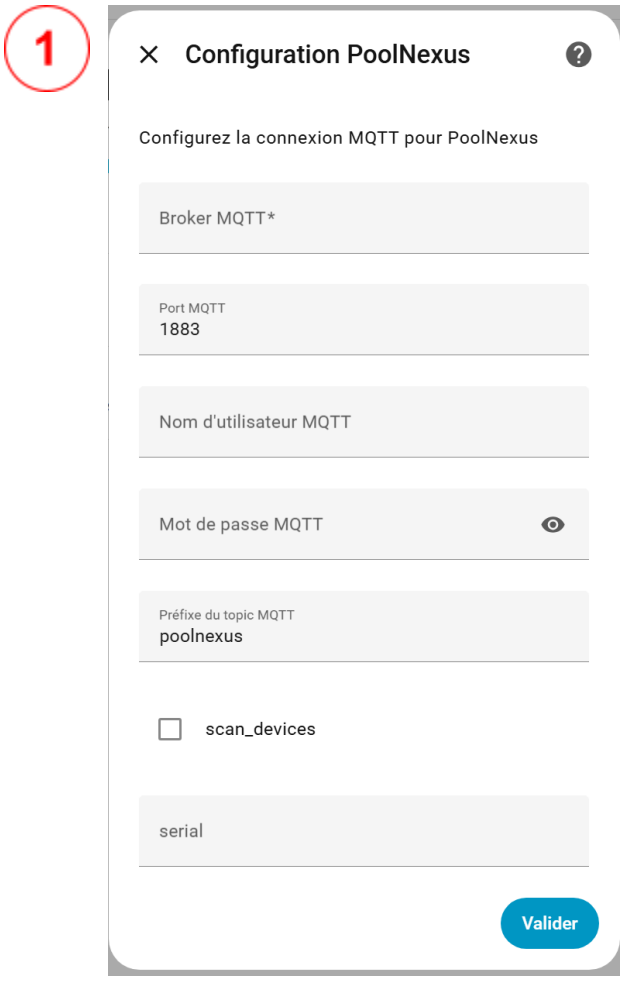
For code documentation, I wanted to use Doxygene, which, through a specific syntax upstream of a function, allows the creation of hostable HTML documentation via a script. Thanks to a CI/CD pipeline on GitHub, this documentation is automatically regenerated and deployed with each push/merge on the main branch of the repository.

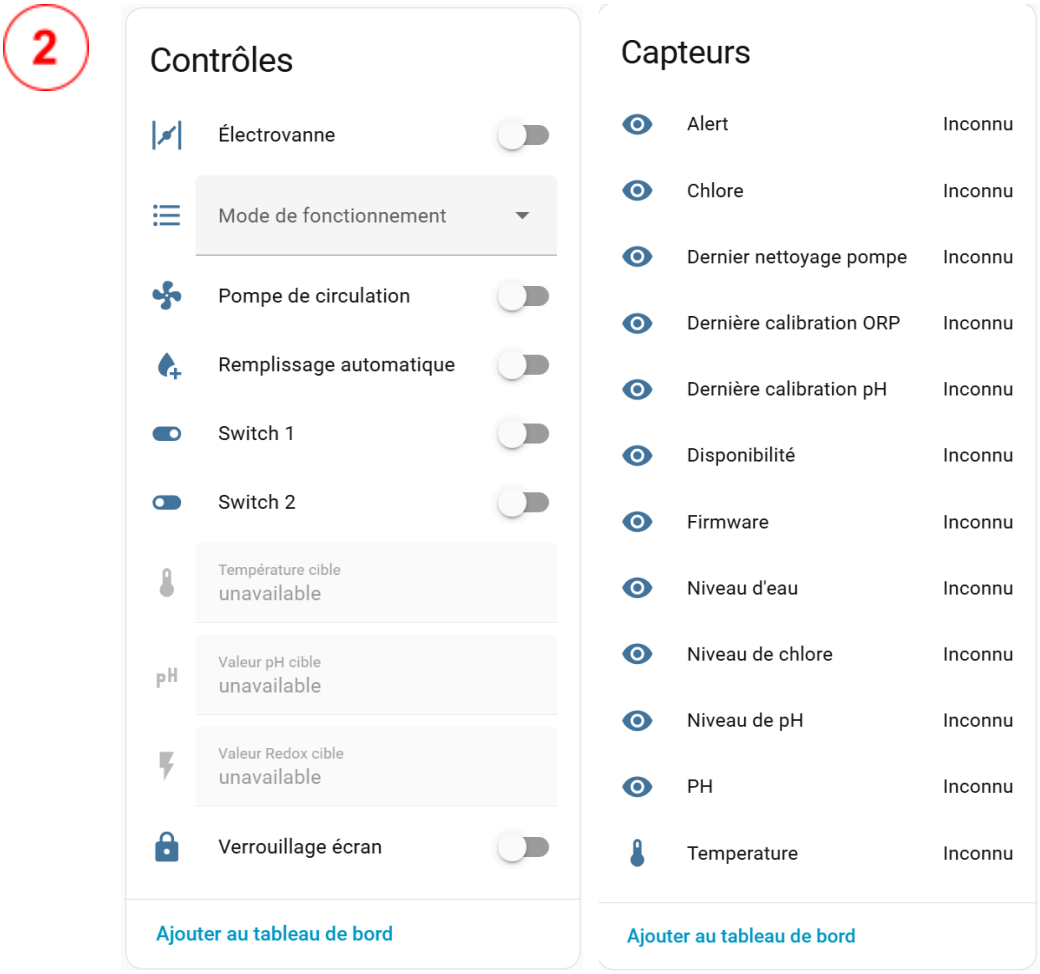
You can view the documentation here : Doxygene Documentation

3 - Home Assistant add-on

For the Home Assistant integration, I chose to use HACS to create my integration. It must therefore be written in Python and comply with HACS requirements, such as a manifest.json file containing: name, version, domain, documentation, requirements, dependencies and codeowners. And adhere to Home Assistant's naming conventions

The main role of this add-on will be to connect to the MQTT server of the homeassistant instance (image 1), to automatically create the sensors/controllers associated with the device (image 2) and to associate them with a value in an MQTT topic so that the values are updated live on homeassistant or on the ESP32 Master.





Furthermore, in order for my HACS integration logo to appear in Home Assistant and the HACS Store, I had to contribute to the GitHub ha-brands repository by adding my logos. This involved creating a fork of the repository GitHub Louis ha-brands FORK and then creating a Pull Request on the main repository GitHub ha-brands Pull Request.

4 - PacketSerial UART ESP32 library

The advantage of creating a library to manage UART communication between two ESP32s is the ability to control its operation from end to end. Furthermore, it avoids duplicating code in both software programs. It was also coded in C++. It works quite simply: to send information, you use the "sendPacket" function and provide the following information: the command type, the data type (int, float, string, bool), the data itself, and its length (especially important for strings). The library then handles the message. I also took on the task of publishing this library on PlatformIO Platform IO PacketSerial UART ESP32, which is a professional, open-source, cross-platform

ecosystem for embedded development. This facilitates my development with ESP32s and allows me to move beyond the overly basic Arduino IDE, given the complexity of the development. You can view the code here : GitHub PacketSerial UART ESP32

C - UART communication protocole

To enable communication between the two ESP32s, I needed to create a dedicated communication protocol, detailed in the appendix (*Figures 1: UART Protocole Packet type* and *Figure 2: UART Protocole*). This protocol defines the frame structure, message types, and the mechanisms for synchronizing and validating the exchanged data. Communication is carried out via a UART connection, chosen for its reliability and ease of implementation in an embedded system. This approach allows for a clear separation of roles between the primary ESP32, responsible for control and processing, and the secondary ESP32, dedicated to the user interface.

D - MQTT protocole

The MQTT protocol operates on a topic-based principle. A topic corresponds to a hierarchical communication channel within which subtopics can be defined, each carrying specific values.

The goal of this organization is to structure data coherently to facilitate its identification, use, and maintenance. This hierarchy also allows the protocol to evolve easily, adding new sensors or commands without modifying the existing architecture.

MQTT is particularly well-suited to connected systems, especially for the IoT, because it is lightweight, efficient, and optimized for real-time data exchange between devices and a central server.

Below is the MQTT protocol :

Read topics (sensors)

- `{prefix}/{serialNumber}/temperature` : Temperature in Celsius
- `{prefix}/{serialNumber}/ph` : Water pH
- `{prefix}/{serialNumber}/chlorine` : Chlorine level in mV
- `{prefix}/{serialNumber}/water_level` : Water level (on/off)
- `{prefix}/{serialNumber}/chlorine_level` : Chlorine level status (low / no liquid / ok)
- `{prefix}/{serialNumber}/ph_level` : pH level status (low / no liquid / ok)

Command topics (switches)

- `{prefix}/{serialNumber}/electrovalve/set` : Electrovalve control (ON/OFF)
- `{prefix}/{serialNumber}/auto_fill/set` : Automatic filling control (ON/OFF)
- `{prefix}/{serialNumber}/switch_x/set` : switch control (ON/OFF)

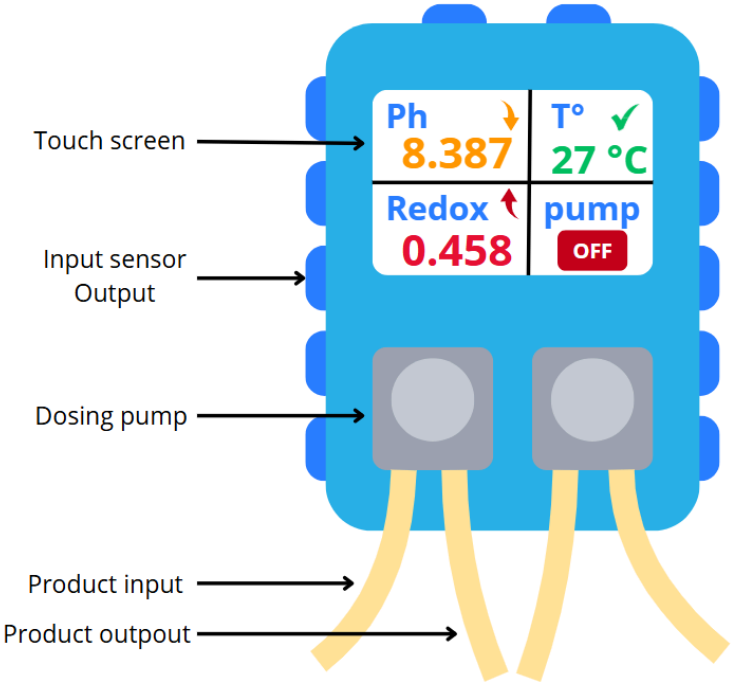
Configuration topics (text)

- `{prefix}/{serialNumber}/set_ph/set` : Set target pH value (format: XX.X)
- `{prefix}/{serialNumber}/set_redox/set` : Set target Redox value (format: X.XXX)
- `{prefix}/{serialNumber}/set_temperature/set` : Set target temperature (format: XX.X)

III - Design of the electronic

Currently, the electronics development is at the advanced concept stage and is not intended for use in a swimming pool, primarily due to the risk of electrocution. To avoid any risk, it will be important to use a galvanically isolated power supply to prevent any electrocution. *Figure 3: System Electronics Schematic* shows the component layout on a future circuit board and also allows for the connections to be made for testing in development mode.

Below you can see an image of the concept art and the final solution :



The final product incorporates two front-mounted peristaltic pumps for automatic adjustment of chlorine and pH levels. It also features two dedicated inputs for pH and Redox probes, as well as an SMA connector for the Wi-Fi antenna. An ESP32 with a touchscreen is positioned on the front panel for displaying information and user interaction. On the right side of the enclosure, watertight bulkhead fittings allow for the connection of various sensors, pumps, and external devices such as lighting.

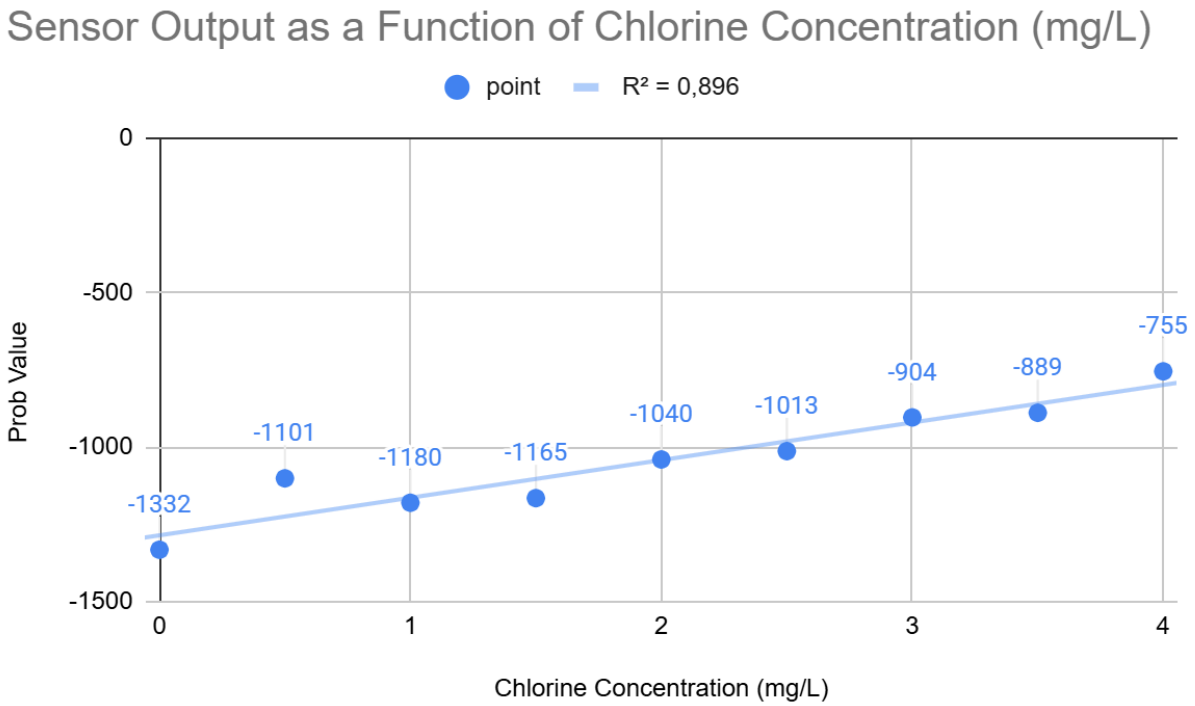
For installation in a technical room, the enclosure must be completely watertight. Particular attention must be paid to sensitive areas, especially around the screen and the peristaltic pumps. As the screen is not inherently watertight, a watertight protection or integration solution still needs to be designed and validated.

III - Implementation and testing of the sensor

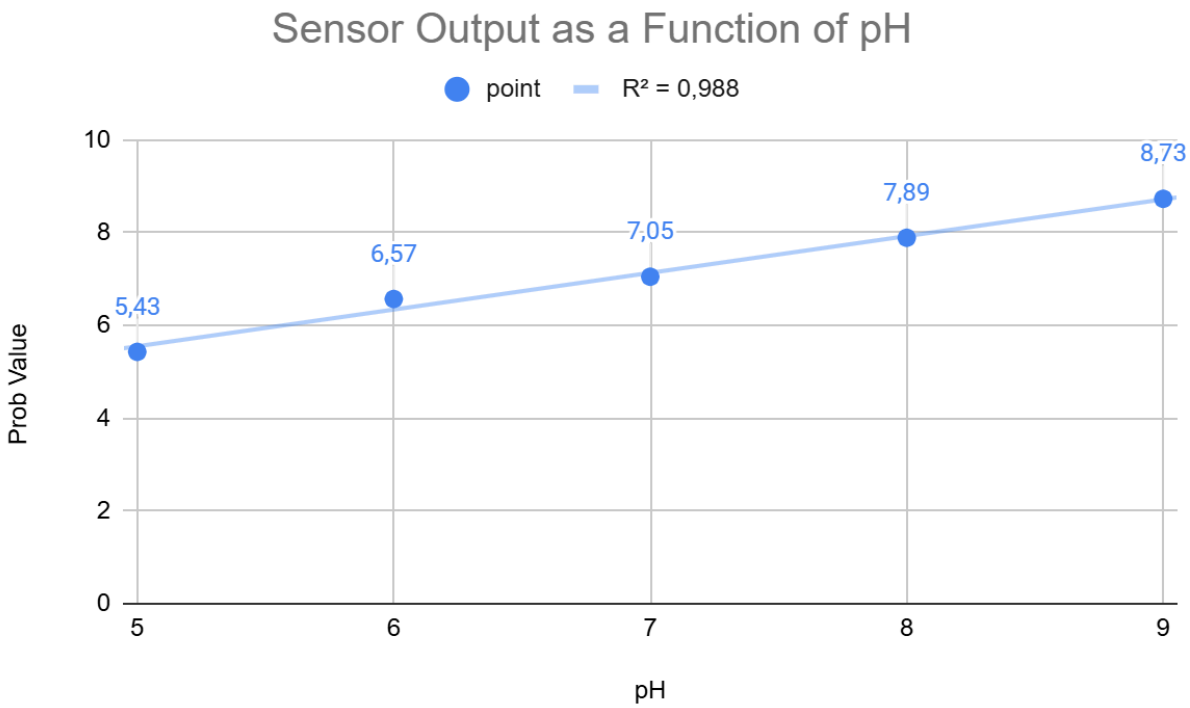
Several tests had to be carried out, including the calibration of the pH and chlorine probes. These tests present risks, as they involve the use of chemicals (acid, base, and chlorine), as well as the simultaneous presence of water and electricity. To mitigate these risks and in accordance with the university's safety requirements, a risk analysis was conducted (Figure 4: Risk Analysis). A test and handling protocol was also written and is available in the section Testing and Handling Protocols

During real-world testing with chlorine, a calculation error was detected in the chlorine concentration display, which distorted the measured values and made them inconsistent. Upon investigation, it was found that this error was systematic and consistent for each measurement, allowing it to be identified and corrected. The results obtained therefore remain usable, as the initial values are simply affected by a shift.

The calibration results for the chlorine probe are shown below. A coefficient of determination (R^2) of 0.89 is observed, indicating a good linear correlation between the measured values and the actual chlorine concentration. The chlorine probe can therefore be considered correctly calibrated.



Regarding the pH probe, the results show an R^2 of 0.98, which corresponds to an excellent correlation. These results provide strong confidence in the reliability and accuracy of the values provided by the pH probe.



Several tests still need to be carried out, including evaluating the system's reaction in the event of a chlorine or acid overdose. These tests will verify that the system correctly interrupts the injection and generates an appropriate error message, thus ensuring the safety and robustness of the device.

V- Conclusion

The aim of this project was to design an automated/connected system for monitoring and regulating swimming pool parameters, reducing manual intervention and the risks associated with inadequate maintenance. Through the development of a hardware solution based on ESP32 and a modular software architecture integrated into Home Assistant via MQTT, the initial objectives were broadly achieved.

The developed system enables reliable measurement of pH, redox potential (chlorine level), temperature, and chemical levels, as well as automated control of dosing pumps, filtration, and auxiliary equipment. The testing and calibration phases validated the accuracy of the sensors, with highly satisfactory correlation coefficients, while highlighting the importance of a rigorous methodology for detecting and correcting software errors.

This project also addressed specific embedded engineering issues, such as inter-microcontroller communication via UART, the creation of a dedicated communication protocol, the development of a reusable library, and integration into an existing home automation ecosystem that complies with strict standards (HACS, Home Assistant). It also highlighted issues related to electrical safety and waterproofing, which are essential for use in humid environments.

Improvements are still possible, including the finalization of a completely waterproof enclosure, the integration of galvanically isolated power supplies, and the validation of advanced safety scenarios in the event of overdose. We should also focus on the security of our software solution in order to prevent any malicious use of the product. At the same time, it is necessary to implement a watchdog program to monitor the main application and automatically restart the system in the event of a crash. Nevertheless, this project is a solid foundation for a reliable, scalable pool management solution that is suitable for real-world use, while also providing a comprehensive learning experience in electronics, embedded systems, and automation.

References

How MQTT Works -Beginners Guide (http://www.steves-internet-guide.com/mqtt-works/)

Beginners Guide To The MQTT Protocol (http://www.steves-internet-guide.com/mqtt/)

Platform IO Documentation (https://docs.platformio.org/en/latest/)

LGVL Documentation (https://docs.lvgl.io/master/)

Doxygene documentation (https://www.doxygen.nl/manual/index.html)

Galvanique isolation (https://fr.wikipedia.org/wiki/Isolation_galvanique)

DIY IoT Water pH Meter using pH Sensor & ESP32
(https://how2electronics.com/diy-iot-water-ph-meter-using-ph-sensor-esp32/#google_vignette)

Appendix

Packet Types	
NAme	**Value hex**
TYPE_REQUEST_VALUE	0x00
TYPE_REQUEST_CHANGE	0x01
TYPE_ACK	0x04

Figure 1: UART protocole Packet type

CMD					ESP SCREEN		ESP MAIN	
Command	Value hex	TYPE_REQUEST_VALUE	TYPE_REQUEST_CHANGE	TYPE_RESPONSE	reception	send	reception	send
constant								
CMD_PH	0x10	N/A		float	NO	OK	OK	NO
CMD_REDOX	0x11	N/A		float	NO	OK	OK	NO
CMD_TEMP	0x12	N/A		float	NO	OK	OK	NO
MQTT Configuration								
CMD_MQTT_SERVER	0x20	N/A		string	NO	OK	OK	NO
CMD_MQTT_PORT	0x21	N/A		int	NO	OK	OK	NO
CMD_MQTT_USER	0x22	N/A		string	NO	OK	OK	NO
CMD_MQTT_PASSWORD	0x23	N/A		string	NO	OK	OK	NO
CMD_MQTT_SERVER	0x20		string	TYPE_ACK	NO	OK	OK	NO
CMD_MQTT_PORT	0x21		int	TYPE_ACK	NO	OK	OK	NO
CMD_MQTT_USER	0x22		string	TYPE_ACK	NO	OK	OK	NO
CMD_MQTT_PASSWORD	0x23		string	TYPE_ACK	NO	OK	OK	NO
Controls								
CMD_PUMP	0x30	N/A		bool	NO	OK	OK	NO
CMD_REGULATION	0x31	N/A		bool	NO	OK	OK	NO
CMD_SWITCH_1	0x32	N/A		bool	NO	OK	OK	NO
CMD_SWITCH_2	0x33	N/A		bool	NO	OK	OK	NO
CMD_AUTO_FILL	0x34	N/A		bool	NO	OK	OK	NO
CMD_LiIGHT	0x35	N/A		bool	NO	OK	OK	NO
CMD_PUMP	0x30		bool	TYPE_ACK	NO	OK	OK	NO
CMD_REGULATION	0x31		bool	TYPE_ACK	NO	OK	OK	NO
CMD_SWITCH_1	0x32		bool	TYPE_ACK	NO	OK	OK	NO
CMD_SWITCH_2	0x33		bool	TYPE_ACK	NO	OK	OK	NO
CMD_AUTO_FILL	0x34		bool	TYPE_ACK	NO	OK	OK	NO
CMD_LiIGHT	0x35		bool	TYPE_ACK	NO	OK	OK	NO
Calibration								
CMD_LEVEL_PH	0x40	N/A		int (0 = LOW, 1 = MEDIUM, 2 = HIGH)	NO	OK	OK	NO
CMD_LEVEL_REDOX	0x41	N/A		int (0 = LOW, 1 = MEDIUM, 2 = HIGH)	NO	OK	OK	NO
CMD_STABILITY_PH	0x42	N/A		int	NO	OK	OK	NO
CMD_STABILITY_REDOX	0x43	N/A		int	NO	OK	OK	NO
CMD_CALIBRATE_PH	0x44		int (valeur de pH a clibré)	TYPE_ACK	NO	OK	OK	NO
CMD_CALIBRATE_REDOX	0x45		int (valeur de redox a clibré)	TYPE_ACK	NO	OK	OK	NO
CMD_VALIDATION_CAL_REDOX	0x46		N/A	TYPE_ACK	NO	OK	OK	NO
CMD_VALIDATION_CAL_PH	0x47		N/A	TYPE_ACK	NO	OK	OK	NO
System								
CMD_PING	0xF0	N/A		TYPE_ACK	OK	OK	OK	OK
CMD_GET_STATUS	0xF2	N/A		string (OK, NOK)	OK	OK	OK	OK
CMD_RESET	0xF1		N/A	TYPE_ACK	NO	OK	OK	NO
CMD_ALERT	0xF3	N/A		tablaux de int	NO	OK	OK	NO

Figure 2: UART protocole

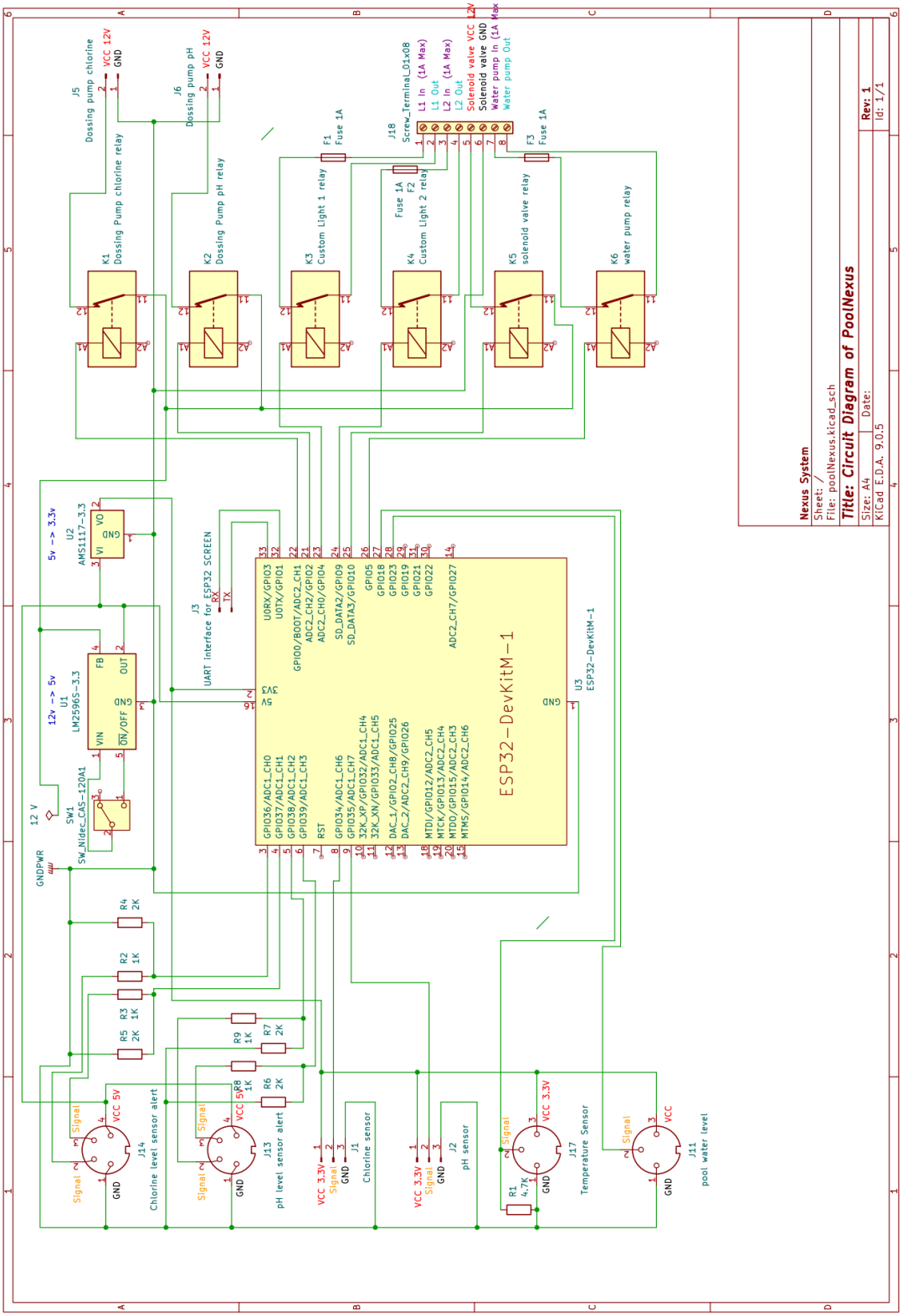


Figure 3: System Electronics Schematic

Hazard / Chemical	Risk Description	Exposure Situation	Preventive Measures	Hazard Class / Hazard Statement (CLP)	Risk Assessment		
					Impact (I) 1-5	Likelihood (L) 1-5	Total risk weighting (I * L)
Water leakage	Risk of slipping, falling, contact with electrical components	During system testing, rinsing, or maintenance	- Install the station on a flat, drained surface - Regularly check seals and pipes - Wear slip-resistant shoes - Immediately report and clean up any leaks	N/A	1	1	1
Electrical safety	Risk of electrocution or short circuit if water/electricity contact occurs	When connecting devices, cleaning, or in case of water leakage	- Use outlets with residual current device (RCD) - Keep hands and surfaces dry - Prohibit any electrical handling when water is present - Regularly maintain cables and connectors	N/A	1	1	1
NaOCl (Sodium hypochlorite 12–15%)	Corrosive and irritant: risk of chemical burns, eye splashes, inhalation of fumes	During preparation of solutions or chlorine testing	- Wear gloves, goggles, and lab coat - Use under a fume hood or in a ventilated area - Never mix with acid (Never inject at the same time as the acid, or inject at a distance of 10 cm from the acid injection point with a high mixing power.) - Store away from sunlight and heat - Dispose of in the container designated for halogenated or oxidizing waste.	**H290** Corrosive to metals **H314** Causes severe skin burns and eye damage **H318** Causes serious eye damage **H400** Very toxic to aquatic life **H411** Toxic to aquatic life with long lasting effects **EUH031** Contact with acids liberates toxic gas	4	2	8
HCl (Hydrochloric acid 20–33%)	Corrosive: risk of burns, irritating fumes, equipment corrosion	When handling to adjust pH	- Wear gloves, goggles, and lab coat - Handle in a ventilated area - Never pour into a highly concentrated chlorinated solution (Never inject at the same time as the chlorine, or inject at a distance of 10 cm from the acid injection point with a high mixing power.) - Store in a separate acid cabinet, away from bases and oxidizers - Throw it in the container designated for acide waste.	**H314** Causes severe skin burns and eye damage **H335** May cause respiratory irritation	4	2	8
NaOH (sodium hydroxide 20-33 %)	Corrosive: risk of burns, irritating fumes, equipment corrosion	When handling to adjust pH	- Wear gloves, goggles, and lab coat - Handle in a ventilated area - Never pour into a highly concentrated chlorinated solution (Never inject at the same time as the chlorine, or inject at a distance of 10 cm from the acid injection point with a high mixing power.) - Store in a separate acid cabinet, away from bases and oxidizers - Throw it in the container designated for basic waste.	**H314** Causes severe skin burns and eye damage **H335** May cause respiratory irritation	4	2	8
Soldering	Minor burns from contact with the hot soldering iron, inhalation of fumes	- Handling the hot soldering iron to assemble electronic components. - Inhaling flux fumes released during soldering.	- Use tweezers or a holder to secure components. - Work in a well-ventilated area to limit inhalation of fumes.	N/A	1	1	1
Cutting	Cuts, flying debris	- Handling sharp tools. - Cutting metal or plastic pieces. - Cleaning or trimming edges after cutting.	- Wear cut-resistant gloves. - Use safety glasses or a face shield to prevent debris. - Work on a stable, clear surface.	N/A	1	1	1
Screens	Eye strain, prolonged posture causing back or neck pain	- Prolonged use of a computer for programming or data analysis. - Reading diagrams or technical documents on screen. - Sitting for long periods at a poorly adjusted screen.	- Organize the workspace properly. - Maintain good posture. - Adjust the screen for comfortable viewing.	N/A	1	1	1

Figure 4: Risk analysis

Glossary

IoT Internet of Things

MQTT Message Queuing Telemetry Transport is a lightweight, publish-subscribe messaging protocol designed for efficient machine-to-machine (M2M) communication, ideal for the IoT.

HACS The Home Assistant Community Store is a custom integration that provides a UI to manage custom elements in Home Assistant.

LVGL Light and Versatile Graphics Library is a popular, free, and open-source C library for creating rich, graphical user interfaces (GUIs) on resource-constrained embedded systems like microcontrollers.

IDE Integrated Development Environment is a software application that bundles essential tools for programmers, like a code editor, debugger, and build automation, into a single interface.

UART Universal Asynchronous Receiver/Transmitter is a hardware protocol for simple, serial data exchange between two devices, using just two wires (transmit and receive) plus ground, without needing a shared clock

CI/CD Continuous Integration/Continuous Delivery (or Deployment), is a set of automated DevOps practices that streamline software development by frequently integrating code changes, automatically testing them, and reliably delivering them to users, creating a fast, efficient pipeline from code commit to production

I^2C is a two-wire serial communication protocol for short-distance communication between integrated circuits