Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Collecting and Compressing of measurement data from a Hydroponic Patch via a Wi-Fi connection

By:

Vikram Vasist

Supervised By:

Prof. Dr. Robert Heß

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Technology and Informatics
Department of Information- and Electrical Engineering

Vikram Vasist

Collecting and Compressing of measurement data from a Hydroponic Patch via a Wi-Fi connection

Bachelor Thesis based on the examination and study regulations for the
Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Robert Heß
Second examiner: Prof. Dr. Ulrike Herster

Day of delivery July 1st 2025

Vikram Vasist

Title of the Bachelor Thesis

Collecting and Compressing of measurement data from a Hydroponic Patch via a Wi-Fi connection

Keywords

Hydroponics, smart agriculture, IoT sensing, Raspberry Pi pico W, RESTful API, Flask, PHP, Plotly.js, Filters, smart-city farming, arid region farming

Abstract

The work detailed in this thesis builds upon earlier laboratory efforts overseen by Prof. Dr. Robert Heß and extends them into a functional hydroponic-monitoring prototype. The system, branded as **HydroSense** integrates low-cost sensing hardware, real-time signal processing, and web interface. HydroSense provides a scalable foundation for both urban vertical farms and off-grid greenhouse operation, while leaving ample scope for future upgrades.

Vikram Vasist

Thema des Bachelorarbeit

Erfassung und Komprimierung von Messdaten eines hydroponischen Beets über eine WLAN-Verbindung

Keywords

Hydroponik, intelligente Landwirtschaft, IoT-Sensorik, Raspberry Pi Pico W, RESTful API, Flask, PHP, Plotly.js, Filter, Smart-City-Landwirtschaft, Landwirtschaft in ariden Regionen

Abstract

Die in dieser Arbeit beschriebenen Entwicklungen basieren auf vorherigen Laborarbeiten unter Leitung von Prof. Dr. Robert Heß und erweitern diese zu einem funktionalen Prototypen für hydroponische Überwachung. Das entwickelte System mit der Bezeichnung **HydroSense** kombiniert kostengünstige Sensortechnik, Echtzeit-Signalverarbeitung und eine Weboberfläche. HydroSense bietet eine skalierbare Grundlage sowohl für urbane vertikale Farmen als auch für autarke Gewächshausanlagen und lässt zugleich Raum für zukünftige Erweiterungen.

Contents

Acknowledgements iii

1 Introduction 1
1.1 Background 1
1.2 Motivation 2
1.3 Objectives 2
1.4 Overview of Approach 2

2 Theoretical Background 3
2.1 Principles of Hydroponics 3
2.2 Sensing Technologies 5
2.3 Challenges of Sensor Calibration 9
2.3.1 Circuit-level Noise Mitigation Techniques 14
2.4 IoT and Embedded Systems 15
2.5 RESTful APIs and Web Data Presentation 16

3 Requirement Specification 19
3.1 General Objective 19
3.2 Mandatory vs. Optional Scope 19
3.2.1 Mandatory Requirements - Local Monitoring Stack 19
3.3 Optional Requirements - Remote Monitoring Stack 20

4 Previous Work 22

5 Design of the Solution 23
5.1 Overall System Architecture 23
5.2 Hardware Layer 23
5.2.1 Functional Overview 23
5.2.2 Input and Output Dynamics 24
5.2.3 Internal State and Behavior 24
5.2.4 Error Handling / Validation Points 24
5.2.5 Behavioral Flow and Activity Structure 25
5.2.6 Architectural Integration 26
5.3 Local Processing and UI layer 27
5.3.1 Functional Overview 27
5.3.2 Input and Output Dynamics 27
5.3.3 Internal State Management 27
5.3.4 Behavioral Flow Activity Structure 28
5.3.5 Validation and Fault Handling 29
5.3.6 Architectural Integration 29
5.4 Remote Processing and UI layer 30
5.4.1 Functional Overview 30
5.4.2 Input and Output Dynamics 30
5.4.3 Internal State Management 30
5.4.4 Behavioral Flow Activity Structure 31
5.4.5 Validation and Fault Handling 32
5.4.6 Architectural Integration 32

6 Implementation Details **34**
6.1 Hardware Layer . . . 34
6.1.1 Driver-Level Architecture . . . 34
6.1.2 Calibration Logic - Analog Sensors . . . 35
6.2 Local Application . . . 38
6.2.1 Sequence Diagram . . . 40
6.2.2 Filtering Techniques . . . 40
6.2.3 Flask Endpoints . . . 42
6.2.4 Front-end with Plotly . . . 43
6.3 Remote Application . . . 44
6.3.1 User Authentication . . . 46
6.3.2 Database Connection . . . 47
6.3.3 Data Visualization . . . 47

7 Test and Validation **50**
7.1 Multi-day Readings at various pH benchmarks . . . 50
7.2 Noise and Repeatability Metrics . . . 51
7.3 Web-Dashboard Usability and Date-Picker Logic . . . 52
7.4 Accuracy Against Laboratory Reference . . . 54

8 Summary and Further Work **55**
8.1 Limitations . . . 55
8.2 Further Work . . . 56

Glossary **57**

List of Tables **58**

References **59**

List of Figures **61**

Acknowledgements

I would like to express my sincere gratitude to my supervisor, **Prof. Dr. Robert Heß**, for his expert guidance, insightful feedback, and unwavering encouragement throughout the course of this research. His thoughtful advice and high standards have been instrumental in shaping the direction and quality of this thesis.

Also, I would like to acknowledge the software tools that made the presentation of my work possible:

- **Circuit Designer IDE**, for its intuitive environment and rich library support in circuit modeling;
- **Visual Paradigm**, for creating clear, professional activity diagrams that documented my workflows; and
- **Mermaid.live**, for producing elegant sequence diagrams that illustrate the dynamic behavior of my system.
- **LaTeX & Overleaf**, for writing this thesis with precise formatting and typographical quality + the Latex AI tool for text enhancement.
- **StackOverFlow**, for code examples and implementation.

1 Introduction

1.1 Background

The first evidence of humans cultivating plants from the geological epoch of the Holocene 11,700 years ago, after the end of the last Ice Age. Temperatures rose up north, which allowed early humans to venture out of Africa. The end of Ice Age also marked the mass dying of large mammals that previously fed the growing early human population. This put strain on the growing early human population. Humans as they migrated into Asia, came across a region of flood plains, abundant rivers and soil perfect for cultivation, this area is now known as the Fertile Crescent.

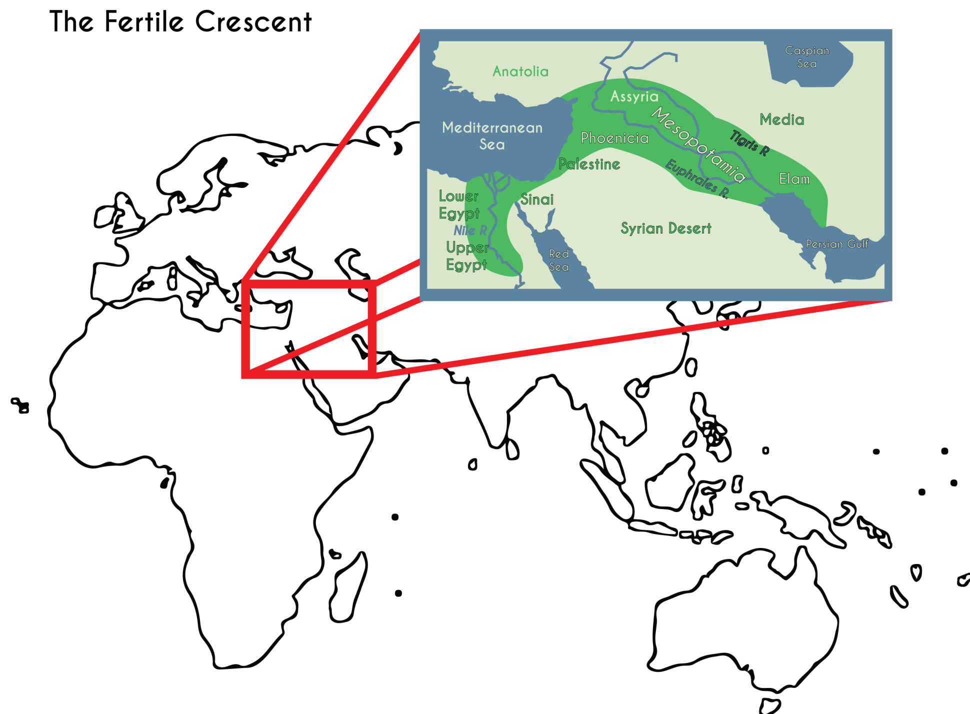


Figure 1: The Fertile Crescent region—an early site of agricultural development. Image source: [1].

However, in the 21st century, we are facing unprecedented environmental and demographic pressures. Reports from the United Nations project a global population of 10.8 billion by 2050. [2] To meet this demand, food production must increase significantly -at a time when fertile land is becoming increasingly scarce and degraded due to overuse, urbanization, and climate change. The intense exploitation of land for food production is making it increasingly unsuitable in the future. The land has long been considered a finite resource, and 25% of the total farmland is rated as highly degraded (Organization for Economic Cooperation and Development, 2012).

Soil is usually the most available growing medium for plants. Provides anchorage, nutrients, air, water, etc. for successful plant growth. Soil is also the natural habitat for the plants. It supplies support, nutrients and water. Where an adequate supply of productive soil is available, growing plants without soil is not practical. However, where good soil is not available, where maintenance of favorable soil conditions is too expensive, or where growth of high-value out-of-season crops is contemplated, growing plants without soil may be desirable. [3]

1.2 Motivation

Soil-less systems such as hydroponics offer a path forward: they decouple food production from fertile top-soil, use up to 90% less water, and can be developed vertically in dense urban areas or in climates where conventional farming is impossible. Coupling hydroponics with low-cost sensing, data analytics, and web dashboards turns a simple grow-bed into a smart-city-micro farm or a resilience tool for arid-county farmers.

1.3 Objectives

This thesis develops and evaluates an open-source hydroponic monitoring platform with two overarching objectives:

1. **Smart-city self-sufficiency** - enable neighborhood-scale or building-integrated food production that cuts transport emissions and eases stress on urban logistics and cold-storage infrastructure.
2. **Climate-resilience for arid regions** - give farmers in water-scarce, high-temperature environments the ability to cultivate crops that normally require mild, humid conditions.

Operational goals that support these societal objectives:

- **Accurate**, continuous sensing of hydroponic contstraints.
- **Robust data conditioning** for noise-free decision-making.
- **Dual-mode connectivity**:
  - *Local Implementation* - completely offline Wi-Fi dashboard for greenhouses or remote sites.
  - *Remote Implementation* - cloud-hosted server with user authentication for large-scale or multi-site deployments.
- **User-friendly web interface** suitable for growers with minimal technical expertise yet extensible for researchers.

1.4 Overview of Approach

To realize these objectives the system was implemented for 2 cases:

Table 1: Approach Overview

Layer	Technology stack	Purpose
Local / Edge	Raspberry Pi Pico (Wi-Fi) sensors → Raspberry Pi running Flask + PostgreSQL	Fully offline operation; instant dashboard; high-frequency filtering
Remote / Cloud	Sensor posts → PHP + MySQL LAMP server; user auth; long-term storage	Centralized data aggregation, multi-user access

2 Theoretical Background

2.1 Principles of Hydroponics

Hydroponics is the act of growing plants without the use of soil. Instead of relying on soil to hold nutrients and water, hydroponics systems deliver everything the plants need-mainly water, nutrients and oxygen-directly to the roots. In traditional soil gardening, water dissolves nutrients in the soil so that plant roots can absorb them. It also helps bring oxygen to the roots.

At its core, a hydroponic system needs three things:

1. **Nutrient-rich water**-This acts as the plant's food source.
2. **Oxygenation**-Roots still need oxygen, so the water has to be aerated.
3. **Water delivery system**-Something has to move the water to the plant roots.

Since nutrients can be pricey, most hydroponic systems are designed to recirculate the water. That means any water that drains off after feeding the plants is collected and reused, making the whole setup more efficient and eco-friendly. Also since plants aren't growing in soil, the water has to do all the heavy lifting-it, we chiefly care about three factors: pH, concentration of Macro-nutrients and oxygenation.

Oxygenation is vital in the water for healthy growth of plants. If the water has low dissolved oxygen, the roots of a plant can "drown". In container gardening, this manifests as over watering, and the symptoms are very similar to under watering. Since hydroponics systems lack soil to retain water, the major concern is under oxygenated water. Luckily, we don't need to worry about over oxygenating the water. For the majority of setups, a small air pump with an air stone is sufficient. Oxygen can also be introduced when the runoff returns to the water source if there is a waterfall (or the water free falls through air). Watering cycles do not seem to be well researched, but common wisdom states a 15-on-45 off cycles. This can be done all day, or the cycle can be reduced at night. However, certain hydroponics setups require constant water flow - Nutrient Film Technique (NFT) is an example of constant water flow. For more traditional systems, the cycling allows the water to drain completely from the root of the plant - allowing the roots to come into contact with air directly - before the next cycle begins. This can also help prevent mold buildup the roots.

Lightening is important for photosynthesis. For outdoor hydroponics systems, the available lighting is usually sufficient. However, indoor systems almost always lack enough sunlight for proper plant growth. Furthermore, indoor lights typically do not emit the proper wavelengths that plants use for photosynthesis. There are many commercially available lights that do produce the light plants need, and some of the newer lights use LEDs, which produce much less heat than older bulbs. The bulb should be placed directly placed over the plants., but care should be taken that the bulb is not too close so that any heat produced will not affect the plants. Common lighting cycles seem to match the sun cycle - about 12 on and 12 off. While it is very important that the plants receive enough light, it is also important that the reservoir gets no light, algae can grow in the reservoir when introduced to light. While algae is not directly harmful to the plants, the biggest impact is to the nutrients concentration. Since algae uses nutrients hust like the

plants do, if algae begins to grow in your reservoir, you will need to add nutrients often to feed your plants, this can become expensive, it is vital that as little light as possible is permitted to enter the reservoir. [4]

pH is important due to nutrient lockout. The various nutrients within a water source are more easily absorbed by a plant's root is based on the pH of the water source - too basic or too acidic, and the plant will not be able to absorb the nutrients from the water, even if they are present. The desired pH range will vary deepening on the plants in the hydroponics system, but typical values are between 5.5 and 6.5.

Vegetable	**pH**	**EC**	**Vegetable**	**pH**	**EC**
Artichoke	6.5–7.5	0.8–1.8	Marrow	6.0	1.8–2.4
Asparagus	6.0–6.8	1.4–1.8	Okra	6.5	2.0–2.4
Basil	5.5–6.5	1.0–1.6	Onions	6.0–6.7	1.4–1.8
Bean (Common)	6.0	2.0–4.0	Pak Choi	7.0	1.5–2.0
Beetroot	6.0–6.5	0.8–5.0	Parsnip	6.0	1.4–1.8
Bok Choi	6.0–7.0	1.5–2.5	Pea	6.0–7.0	0.8–1.8
Broad Bean	6.0–6.5	1.8–2.2	Pea (Sugar)	6.0–6.5	0.8–1.4
Broccoli	6.0–6.5	2.8–3.5	Pepino	6.0–6.5	2.0–5.0
Brussel Sprout	6.5–7.5	2.5–3.0	Peppers	5.8–6.3	2.0–3.0
Cabbage	6.5–7.0	2.5–3.0	Peppers (Bell)	6.0–6.5	2.0–3.0
Capsicum	6.0–6.5	1.8–2.2	Peppers (Hot)	5.0–6.5	3.0–3.5
Carrots	6.3	1.6–2.0	Potato	5.0–6.0	2.0–2.5
Cauliflower	6.0–7.0	0.5–2.0	Pumpkin	5.5–7.5	1.8–2.4
Celery	6.5	1.8–2.4	Radish	6.0–7.0	1.6–2.2
Cucumber	5.8–6.0	1.7–2.5	Spinach	6.0–7.0	1.8–2.3
Eggplant	5.5–6.5	2.5–3.5	Silverbeet	6.0–7.0	1.8–2.3
Endive	5.5	2.0–2.4	Sweet Corn	6.0	1.6–2.4
Fodder	6.0	1.8–2.0	Sweet Potato	5.5–6.0	2.0–2.5
Garlic	6.0	1.4–1.8	Taro	5.0–5.5	2.5–3.0
Kale	5.5–6.5	1.25–1.5	Tomato	5.5–6.0	2.0–5.0
Leek	6.5–7.0	1.4–1.8	Turnip	6.0–6.5	1.8–2.4
Lettuce	5.5–6.5	0.8–1.2	Zucchini	6.0	1.8–2.4

Table 2: Optimal pH and EC levels for various hydroponic vegetables.Table source: [5]

Macro-Nutrients refer to Nitrogen, Phosphorus and Potassium (fertilizers typically list these as relative concentrations: 10-10-10 means the nutrients are balanced) - plants use these three in greater concentrations than other nutrients like Iron and Magnesium, which are called Micro-nutrients. The particular concentrations your system will need depends heavily on the plants you plan on growing. Leafy vegetables like lettuce and cucumbers need a higher concentration of Nitrogen, whereas others (like tomatoes) require a more balanced mixture. In any case, hydroponics systems use special nutrient mixes designed fro hydroponics, and will cause buildups that can be difficult to get rid of. As the plants use up the nutrients in the water, more fertilizer will need to be added. The common way of detecting when it is time to add more is with an Electrical Conductivity (EC) Meter.

These meters measure how conductive a solution is. This is important for hydroponics because we typically use distilled water (or de-ionized water) and add nutrient solution. Since the nutrient solution is the only source of ions, EC is a reasonable measure of nutrient concentration. EC is not a direct measure of nutrient concentration. EC is not a direct measure of nutrient concentrations, however, and if other sources of ions make their way into the water source, then EC will not be a very accurate measure of nutrient concentration.

2.2 Sensing Technologies

The system developed in this thesis integrates the following sensors:
1. SEN0161 Gravity pH Sensor – The analog pH sensor is a electrochemical sensor which measures the acidity or the alkalinity of a liquid solution by producing a voltage that depends on the hydrogen ion concentration. It typically uses a standard pH probe (glass electrode) connected to a signal conditioning module which aplifies and conditions the weak millivolt signal produced by the probe. The measuring range of sucha a sensor is generally from pH 0 to 14, with a typical accuray ± 0.1 to ± 0.2 pH units depending on proper calibration and temperature compensation.

Figure 2: SEN0161 pH Sensor by DFRobot [6]

This type of pH sensor outputs a steady and continuous analog voltage signal, which must be read using an analog-to-digital converter (ADC) on the microcontroller. The voltage is then converted to a pH value using calibration curve, which is obtained by immersing the sensor in buffer solutions that has been standardized, and adjusting the formula accordingly. The simplicity of the analog interface makes the sensor easy to integrate with various micro-controllers, such as Arduino or Raspberry pi.

Proper care and calibration are required to maintain measurement accuracy, as pH probes can drift as the time passes due to contamination or temperature effects. These sensors are widely used in hydroponics, aquariums, environmental monitoring, and industrial process control, where uninterrupted monitoring of water is essential. [6]

2. DS18B20 Sensor – The DS18B20 is a digital thermometer that can measure temperature in Celsius. It offers different accuracy levels from 9 to 12 bits. This means that

it can provide very accurate measurements, with a higher bit count providing a more accurate temperature measurement. The measuring range of the DS18B20 is between -55°C and +125°C, and the accuracy is ±0.5°C in the range from -10°C to +85°C.

Figure 3: DS18B20 Temperature Sensor [7]

The DS18B20 uses a so-called 1-wire bus to communicate with a microprocessor. This means that only a single data line is needed to exchange information. The sensor can even draw its power from this data line, so no additional power source is required. This mode of operation is called Parasite Power.

Each DS18B20 sensor has a unique 64-bit serial code. This is like a unique identification number for each sensor. This means that several DS18B20 sensors can be connected to the same 1-Wire bus and still be individually recognized and read out. This makes it easy to implement networks with multiple sensors.

This sensor is particularly useful in many applications, such as in heating, ventilation and air conditioning (HVAC) systems, for temperature monitoring in buildings or machines and in process monitoring and control systems. Other applications include medical devices, industrial control systems and temperature monitoring in cooling systems. [7]

Raspberry Pi Pico		Sensor
GPIO2	↔	Signal
3.3V	↔	+V
GND	↔	GND

Table 3: Pin mapping between Raspberry Pi Pico and sensor

3. KS0429 TDS Meter V1.0 – Keyestudio TDS sensor kit is compatible with Arduino controllers, plug and play, easy to use. It can be applied to measure TDS value of the water, to reflect the cleanliness of the water.

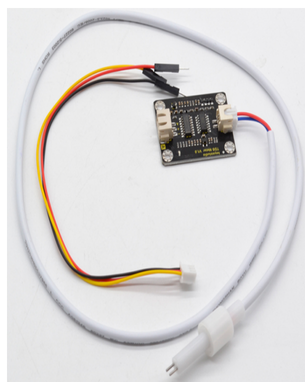


Figure 4: KS0429 Keyestudio TDS Meter V1.0

TDS (Total Dissolved Solids) indicates that how many milligrams of soluble solids dissolved in one liter of water. In general, the higher the TDS value, the more soluble solids dissolved in water, and the less clean the water is. Therefore, the TDS value can be used as one of the references for reflecting the cleanliness of water

Measuring the TDS value in the water is to measure the total amount of various organic or inorganic substances dissolved in water, in the unit of ppm or milligrams per liter.

Its Electrode can measure conductive materials, such as suspended solids, heavy metals and conductive ions in water. The module comes with four 3.2mm fixed holes, easy to mount on any other devices.

4. DS3231 RTC – General Description The DS3231 is a low-cost, extremely accurate I2C real- time clock (RTC) with an integrated temperature- compensated crystal oscillator (TCXO) and crystal. The device incorporates a battery input, and maintains accu- rate timekeeping when main power to the device is inter- rupted. The integration of the crystal resonator enhances the long-term accuracy of the device as well as reduces the piece-part count in a manufacturing line. The DS3231 is available in commercial and industrial temperature ranges, and is offered in a 16-pin, 300-mil SO package.

Figure 5: DS3231 RTC ClocK [8]

The RTC maintains seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator. Two programmable time-of- day alarms

and a programmable square-wave output are provided. Address and data are transferred serially through an I2C bidirectional bus.

A precision temperature-compensated voltage reference and comparator circuit monitors the status of VCC to detect power failures, to provide a reset output, and to automatically switch to the backup supply when necessary. Additionally, the RST pin is monitored as a pushbutton input for generating a μP reset. [8]

5. BME280 – The BME280 is as combined digital humidity, pressure and temperature sensor based on proven sensing principles. The sensor module is housed in an extremely compact metal-lid LGA package with a footprint of only 2.5×2.5 mm^2 with a height of 0.93 mm. Its small dimensions and its low power consumption allow the implementation in battery driven devices such as handsets, GPS modules or watches. The BME280 is register and performance compatible to the Bosch Sensortec BMP280 digital pressure sensor

The BME280 achieves high performance in all applications requiring humidity and pressure measurement. These emerging applications of home automation control, in-door navigation, fitness as well as GPS refinement require a high accuracy and a low TC_0 at the same time

The humidity sensor provides an extremely fast response time for fast context awareness applications and high overall accuracy over a wide temperature range.

The pressure sensor is an absolute barometric pressure sensor with extremely high accuracy and resolution and drastically lower noise than the Bosch Sensortec BMP180.

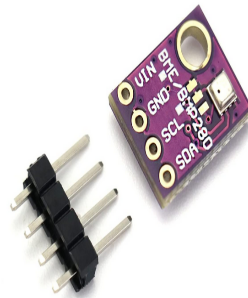


Figure 6: BME280 Temperature Sensor

The integrated temperature sensor has been optimized for lowest noise and highest resolution. Its output is used for temperature compensation of the pressure and humidity sensors and can also be used for estimation of the ambient temperature.

The sensor provides both SPI and I^2C interfaces and can be supplied using 1.71 to 3.6 V for the sensor supply VDD and 1.2 to 3.6 V for the interface supply VDDIO. Measurements can be triggered by the host or performed in regular intervals. When the sensor is disabled, current consumption drops to 0.1 μA.

BME280 can be operated in three power modes:

- **Sleep Mode**
- **Normal Mode**
- **Forced Mode**

In order to tailor data rate, noise, response time and current consumption to the needs of the user, a variety of oversampling modes, filter modes and data rates can be selected. [9]

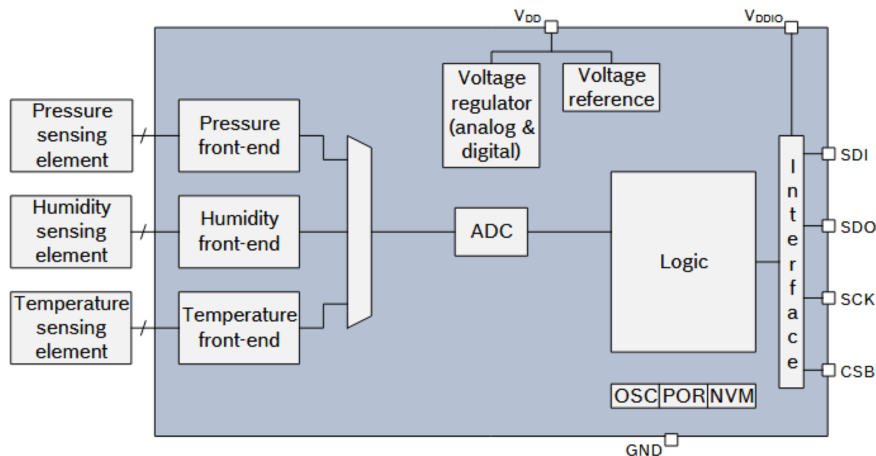


Figure 7: BME280 Block Diagram. Image source: [9].

2.3 Challenges of Sensor Calibration

Sensor calibration in hydroponic systems presents multifaceted technical and environmental challenges that require careful consideration:

1. Electrochemical Drift

pH sensors , particularly glass-electrode analog probes, exhibit gradual signal drift due to:

- **Electrode aging**: The reference electrolyte slowly leaks through the junction, changing potential over time.
- **Fouling effects**: Organic residues from nutrient solutions coat the glass membrane, reducing sensitivity. Studies show biofilm formation can cause up to 0.5 pH unit deviation within 72 hours.
- **Dehydration**: Probes left dry between measurements develop crystalline deposits that affect ion exchange.

Mitigation: Automated recalibration routines every 48-72 hours are recommended for continuous monitoring systems.

2. Non-linear response characteristics

The behavior of real-world pH electrodes significantly deviates from the ideal Nernst model, which predicts a perfectly logarithmic response to changes in hydrogen ion concentration. According to the Nernst equation, the voltage (E) generated by an ideal pH sensor changes by approximately 50.16 mV per pH unit at 25°C. This slope assumes ideal electrode performance, perfect ionic mobility, and no interfering factors. However, actual measurements with commercial pH probes frequently show deviations because of electrochemical, mechanical, and manufacturing constraints.

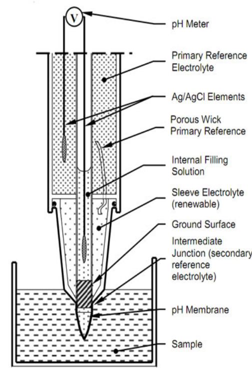


Figure 8: Standard pH electrode construction [10]

Nernst Equation (hydrogen-ion-sensitive electrode):

$$E = E_0 + \frac{RT}{nF} \ln\left([\mathrm{H}^+]\right)$$

- E : Measured electrode potential (volts)
- E_0 : Standard electrode potential (volts)
- R : Universal gas constant ($8.314\,\mathrm{J\,mol^{-1}\,K^{-1}}$)
- T : Absolute temperature (Kelvin)
- n : Number of electrons transferred in the redox reaction
- F : Faraday constant ($96485\,\mathrm{C\,mol^{-1}}$)
- $[\mathrm{H}^+]$: Hydrogen ion concentration (mol/L)

- **Deviation from Ideal Slope**: One of the most common discrepancies observed is the **non-ideal slope** of the pH response curve. While the theoretical sensitivity is 59.16 mV/pH at 25° C, most commercial pH sensors achieve only 90-95% of this value. This reduction in sensitivity results from several contributing factors:
  - *Membrane quality and thickness*: Variations in glass composition, thickness, and homogeneity affect ion exchange dynamics and electrochemical gradient.
  - *Temperature effects*: Since the Nernst equation is temperature-dependent, even minor fluctuations can lead to inaccurate readings unless compensated for.

- *Aging and fouling*: Prolonged use causes membrane degradation, junction clogging, and electrolyte dilution, reducing sensor responsiveness over time.

  This nonlinear particularity affects applications requiring high-precision measurements, such as biomedical diagnostics, bio process control, and environmental monitoring. Regular recalibration and slope verification are necessary to maintain measurement fidelity.

- **Asymmetric potential**: In an ideal system, the voltage output of a pH electrode in a pH 7.00 buffer should be zero (or a defined reference point). However,a asymmetric potential - a voltage offset. often occurs due to imperfections in electrode construction and aging. This zero-point error manifests itself as a change in the expected baseline potential, which can significantly skew measurements if not corrected.

  Causes of asymmetric potential include:

  - *Manufacturing imperfections* in the internal reference electrode, often due to inconsistent coatings or incomplete silver chloride deposition.
  - *Contamination* of the internal or external reference solution by proteins, lipids or salts, which alters the ionic balance.
  - *Electrode aging*, where the hydrated gel layer on the glass membrane becomes uneven or partially dehydrated, disrupting ion exchange.

- **Electrochemical Dynamics (The Role of junction and Ionic Flow)**: Figures 8(a) and 8(b) shows the difference between idealized and practical electrode behavior. In (a), the junction allows controlled diffusion of ions like Cl^- from the internal filling solution to the test sample, maintaining electrochemical contact. In practice, as shown in (b), this process is more chaotic. Potassium (K^-) and chloride (Cl^-) ions migrate through the junction in both directions, and this uncontrolled ion exchange can:

  - Lead to *sample contamination*, especially in long-term monitoring.
  - Alter the *junction potential*, a small voltage that develops at the liquid junction, which becomes significant in high-resistance or low-ionic-strength solutions.
  - Cause *signal instability* or slow response times in low-buffer-capacity media.

- **Multi-Point Calibration and ISO Compliance**: Unlike theoretical models, real sensors do not produce perfectly linear output across the entire pH range. Relying on single-point calibration (typically at pH 7.00) introduces significant inaccuracies, particularly near the acidic and basic ends of the spectrum. Inaccuracies of up to ±0.3 pH units at the extremes (e.g., pH 4 or pH 10) are not uncommon with this method.

  To address this, multi-point calibration becomes essential. This project employed a 3-point calibration protocol using NIST-traceable buffer solutions at:

  - pH 4.01 (acidic)
  - pH 7.01 (neutral)
  - pH 10.01 (alkaline)

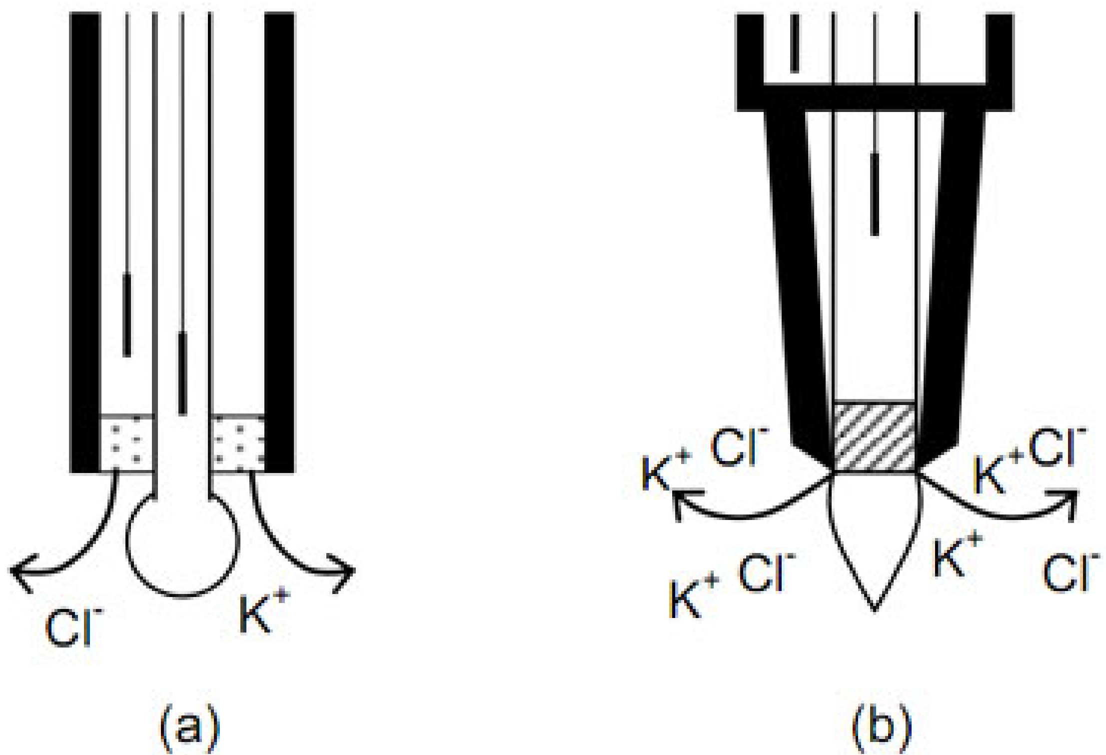


Figure 9: Common electrode designs, have a porous ceramic restriction, see (a). Whereas it allows free movement of ions past the restriction, see (b) . Image source: [10].

Following ISO 17025 protocols, this method ensures:

- *Traceability*: Calibration references are linked to national metrology standards.
- *Repeatability*: Consistent performance across devices and environments.
- *Accuracy*: Reduced non-linearity across the full measurement range.

The calibration process involves measuring the probe output at each reference point, constructing a best-fit regression or interpolation curve, and applying corrections across the range. In high-accuracy environments, temperature compensation and electrode diagnostics are integrated to dynamically adjust for drift and slope variation over time.

3. Signal Integrity Challenges

The analog voltage signals generated by the pH electrodes are typically in the range of$\pm$**414 mV** on the pH scale of 0-14. These small signals are highly susceptible to various forms of electrical and environmental interference. Ensuring signal fidelity is therefore critical, particularly in applications such as precision agriculture, biochemical analysis, and environmental monitoring, where misinterpretation of pH data can lead to flawed decisions or experimental errors.

Several common sources of noise and signal degradation are in analog pH measurement systems:

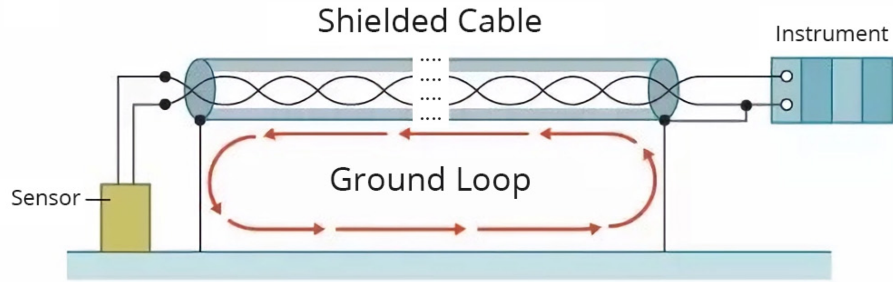


Figure 10: Ground looping: [11].

- **Ground loops** Ground loops occur when multiple devices share a common ground potential, but are also grounded at different physical points. This difference creates unintended current paths, introducing noise typically in the range of a few millivolts to more than 10mV - equivalent to 0.2 pH units of error. This is particularly problematic when sensors are distributed across large-scale systems (e.g. greenhouse hydroponics or aquaculture tanks. [11]

  Mitigation Strategies:

  - Use of *isolated ground references* or *differential amplifiers* to break the loop.
  - *Star grounding topology*, where all ground connections are at a single point.
  - Incorporation of *optional or galvanic isolation* when interacting with microcontrollers or data acquisition systems.

- **Capacitive coupling**: Capacitive coupling is an major concern in electrically noisy environments, especially when PWM signals (pulse width modulation) from devices such as solenoid valves, water pumps, or LED lighting systems induce high-frequency noise on adjacent signal lines. These disturbances manifest themselves as spikes or oscillations in the pH signal, which may be misinterpreted as pH fluctuations.

  Mitigation Strategies:

  - Employ Twisted-Pair Cabling with tightly coupled conductors to reduce loop area and cancel out common-mode interference.
  - Apply Faraday shielding (e.g. foil or braided connected to chassis ground) around signal cables.
  - Physically separate low-level analog signal paths from high-power or high-frequency switching elements.
  - Differential measurement circuitry.

- **Solution Resistance Effects (Voltage Divider)**: In high-electrolyte-concentration (high-EC) nutrient solutions, the resistance of the measurement loop becomes significant. This can result in a **voltage divider** effect between the electrode's internal impedance and solution resistance, reducing the measured signal amplitude and introducing nonlinearities.

  This effect becomes even more pronounced if the reference junction becomes partially blocked, increasing the series impedance and slowing the sensor response time.

  Mitigation Strategies:

- Choose low-impedance glass electrodes suited for high-EC environments.
- Ensure frequent cleaning and maintenance of reference junction.
- Use high-input-impedance ($> 10^{12}\,\Omega$) instrumentation amplifiers to prevent loading the sensor output.

2.3.1 Circuit-level Noise Mitigation Techniques

Given the vulnerabilities outlined above, multiple **hardware-level design strategies** are employed to preserve the integrity of the pH signal before it reaches the analog-to-digital conversion (ADC) stage:

- **Twisted-Pair Cabling with Faraday Shielding**
  - Twisted-pair conductors minimize electromagnetic interference by reducing the effective loop area.
  - Surrounding the cable with grounded conductive (Faraday Cage) absorbs external EMI before it can couple into the signal lines.
  - Shielding must be properly terminated to a **single ground point** to avoid forming new ground loops.
- **4th-Order Bessel Low-Pss filter (10 Hz Cutoff)**

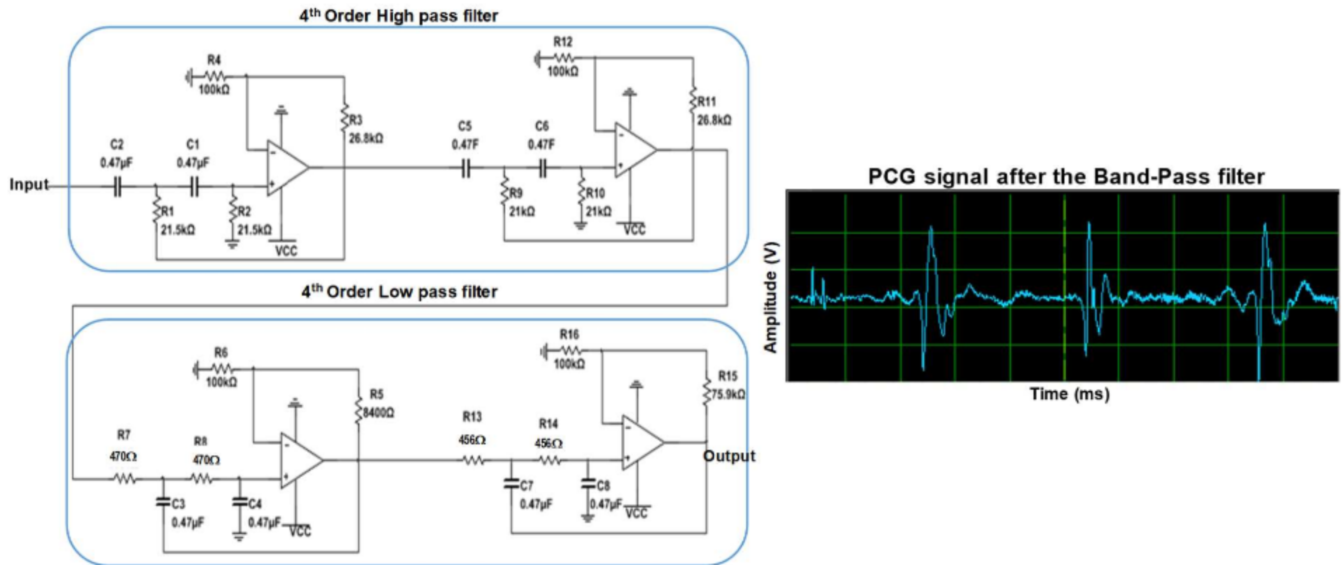


Figure 11: Schematic of fourth-order Bessel band pass filter.: [12]

  - High-frequency noise components, especially those from power electronics or digital switching circuits, are attenuated using analog filtering.
  - A **4th-order Bessel filter** is selected for its **linear phase response**, which preserves waveform integrity and avoid phase distortion, crucial in time-sensitive applications.
  - The 10 Hz cutoff) frequency is chosen to match the expected dynamics of pH changes while eliminating irrelevant high-frequency interference.

- **Differential Measurement Circuitry**
  - Instead of measuring the electrode signal relative to noisy or fluctuating system ground, differential amplifiers compare two input lines (signal and reference).
  - This suppresses **common-mode noise** (identical voltage present on both inputs), improving signal-to-noise ratio (SNR).
  - Differential techniques are especially important when electrode cables exceed **1 meter** in length, where cable capacitance and inductive coupling become problematic.

2.4 IoT and Embedded Systems

The core of this project lies in the deployment of an **Internet of Things (IoT)** framework to monitor the crucial environmental parameters for hydroponic cultivation. The system was designed to be modular, cost effective, and internet-enabled, allowing real-time sensing and remote data logging.

Embedded Hardware Platform

The Raspberry Pi Pico was selected for its affordability and was provided by prof. Heß. Pico is affordable, has lower power consumption and GPIO flexible. It served as the central unit interfacing with:

- A **DS18B20 digital temperature sensor** (1-Wire) for water temperature.
- A **SEN0161 PH Sensor V1.1**: for nutrient concentration.
- A **KS0429 TDS Meter V1.0** is a sensor for measuring the total dissolved solids present in the water.
- A **BME280**: for air temperature, humidity and pressure.

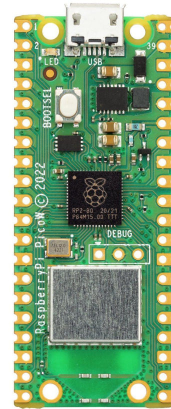


Figure 12: Raspberry Pico-W [13]

Sensor data was collected using Micro-Python, allowing Pythonic interaction with microcontroller-level operations and peripheral communication.

Software Architecture Evolution

The project originally employed a **local Flask-based web server** running on a Debian machine:

- The Raspberry Pi Pico sent data via **HTTP Post** requests to the Flask server over a local Wi-Fi network.
- The server stores incoming sensor data into a local **Postgres Database**.
- A **HTML/CSS** dashboard was implemented to visualize readings over time.

This initial version allowed for rapid prototyping and validation of the data flow, end-to-end connectivity, and sensor integrity.

Transition to PHP-Based Web Infrastructure

As the project matured, it transitioned to a production-ready PHP-based web hosting setup:

- A remote server was provisioned under the sub-domain *vasist.rrhess.de* .
- Sensor data was transmitted over a secured HTTP endpoint to an insert.php script, which parsed the JSON payload and stored the data in a **MySQL database** .
- A structured schema was created with field of the values that were measured using the Sensors mentioned above.
- Data visualization was later enabled through a front-end using **PHP and HTML**, with potential for future integration of charting tools.

  The evolution from local prototyping to cloud-connected deployment cycle -from edge computation and offline resilience to centralized remote management and analytics.

2.5 RESTful APIs and Web Data Presentation

As sensor-based system evolved from a standalone system, embedded designs to **network-connected smart infrastructures**, the need for robust, scalable communication protocols becomes paramount. In this context, **RESTful APIs (Representational State Transfer)** have become one of the cornerstone in IoT system architecture. These APIs enable seamless and standardized data exchange between physical devices and web-based interfaces, allowing sensor data to be retrieved, stored, visualized and managed in real-time across multiple platforms.[14]

RESTful APIs are especially well-suited for IoT deployment due to their **stateless, lightweight architecture**, which uses standard HTTP method to carry out basic CRUD (Create, Read, Update, Delete) operations. Each RESTful route corresponds to a well-defined resource and HTTP verb, making them intuitive for both developers and client-side applications to interact with. In this project, the following operations were used extensively:

- **GET** - Retrieve the latest sensor data (e.g., most recent pH or temperature values) for real-tome dashboards or analytics tools.
- **POST** - submit new sensor readings from a microcontroller to a centralized server.
- **PUT** - Modify configuration parameters such as calibration coefficients or threshold limits.
- **DELETE** - remove corrupted data entries or deprecated configuration settings.

This model of stateless communication improves **interoperability and scalability**, particularly in systems where microcontrollers like the Raspberry Pi Pico must regularly

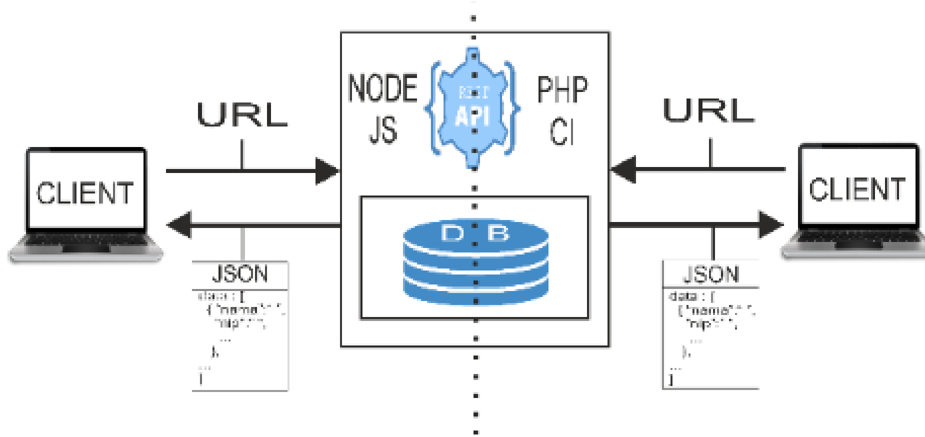


Figure 13: BME280 Block Diagram. Image source: [9].

push updates without maintaining persistent connections. Because each API call contains all the necessary context, devices can operate independently and efficiently in bandwidth-constrained or intermittently connected environments.

Integration in this thesis

In the early stages of this thesis, data was collected using **Micro-Python** on a Raspberry Pi Pico W and stored locally in a flat file format (.CSV). This setup offered limited extensibility and lacked remote accessibility. To address this, a **Flask-based REST API** was developed. Flask, a lightweight Python web framework, enabled quick prototyping and testing of API endpoints. The backend service could:

- Accept **JSON-formatted payloads** via POST requests from the microcontroller.
- Insert and manage records in a **PostgreSQL database**, chosen for its performance and relational capabilities,
- Handle GET requests from the web clients to serve both **current and historical data** on demand.

As the project matured and the deployment goals expanded, the back-end infrastructure was changed to a more production-ready **PHP-based server environment**. This new stack utilized **MySQL** for storage and extended API functionality via PHP scripts. Key enhancements included:

- Dedicated endpoints for temperature, pH, and calibration history, providing fine-grained access control.
- A **WebSocket-based real-time data layer** and **cron job scheduling** for automatic data updates and visualization refreshes.
- Integration with mobile and desktop browsers through responsive front-end design.

Web Visualization Techniques

To convert raw numerical sensor data into **user-friendly visual insights**, several front-end visualization tools were implemented. These included:

- **Dynamic HTML tables** for real-time display of sensor metrics,
- **Interactive line graphs** for pH and temperature trends, implemented using libraries such as **Plotly.js** or **Chart.js**, which support zooming, panning, and tooltips for precise data exploration,
- **Color-coded indicators**, , such as red for out-of-range pH values and green for optimal conditions, to offer at-a-glance system status to users.

The visualization interface was designed using responsive design principles, ensuring compatibility across devices including tablets, smartphones and desktop displays. By providing clear, intuitive visual feedback, the interface allowed end-users - such as farmers or lab technicians - to make timely and informed decisions regarding their hydroponic systems.

Modular Design and Scalability

By adopting a **modular software architecture**, the project clearly separated the data collection, communication, storage and presentation layers. This abstraction allowed for independent development and debugging pf each module, significantly improving maintainability and scalability. For example, if sensor hardware or calibration model changes, the data acquisition code can be updated without impacting the API layer or front-end display.

Furthermore, using RESTful APIs ensures future compatibility with other data-consuming applications. This opens up potential integrations with:

- **Cloud-based platforms** (e.g. AWS IoT, Azure IoT Hub,
- **Mobile apps** for remote system monitoring,
- **Data analytics tools** for pattern recognition or predictive maintenance.

The incorporation of RESTful APIs bridged the gap between embedded hardware and modern web technologies, transforming raw environmental readings into actionable, accessible and visually meaningful insights.

3 Requirement Specification

3.1 General Objective

The primary goal of this work is to develop a working prototype of a smart monitoring system for a small-scale hydroponic setup. The system is designed to **automatically measure environmental parameters** - namely pH and temperature of the nutrient solution - and display the results in a user-friendly web interface. The aim is to deliver a complete solution that integrates **sensor data acquisition, real-time data processing**, and **web-based visualization**, providing an accessible tool for both hobbyists and agricultural practitioners.

This system is intended for users with minimal technical expertise, such as farmers, educators, or botany enthusiasts. Therefore, the interface and the underlying infrastructure must prioritize simplicity and reliability, while still allowing extensibility for more advanced users who may wish to tweak, scale or adapt the system for different crops or hydroponic designs.

3.2 Mandatory vs. Optional Scope

At the beginning of this thesis, Prof. Heß and I defined a core set of **mandatory** system requirements - the hardware, firmware, database, and front-end features that any acceptable prototype must implement. As the work progressed, we implemented **optional** system requirements - remote dashboard, MQTT support, cloud-based storage, etc. - which enhanced the solution but were not essential to the primary purpose. These **optional** requirements were nevertheless completed, with diligence.

Scope	**Description**	**Status**
Local Monitoring Application	Pico W → Flask → PostgreSQL → Plotly dashboard (runs entirely inside the greenhouse network)	**Mandatory**
Remote Web Portal (public sub-domain)	PHP / MySQL site reachable over the Internet, with TLS, login, MQTT ingest, etc.	**Optional**

Table 4: Mandatory vs. Optional Scope

NOTE: the above mentioned differentiation, between Mandatory and Option requirements are just remnants of early discussions between me and supervisor. While developing HydroSense, no such distinction was taken into account, both are implemented as standalone systems in their own right. They were simply referred to as Local and Remote layers.

3.2.1 Mandatory Requirements - Local Monitoring Stack

ID	Category	Requirement	Success Criterion
HW-1	Micro-controller	Raspberry Pi Pico W (RP2040 + Wi-Fi)	Stable boot, sensor polling $\geq$ 1 Hz
HW-2	Power	5 V USB or Li-ion pack	$\geq$ 8 h autonomous runtime; splash-proof enclosure
HW-3	Network	Configure a local Wi-Fi network over which, Raspberry Pi Pico can send data	Stable connection for the devices connected + JSON payload capability
SW-1	Firmware	MicroPython code: calibration, sensor reading, WLAN AP join	Reports JSON payload over UDP / HTTP
SW-2	Local backend	Flask 2.x on Raspberry Pi OS or Linux PC	Serves `index.html`, `/data` (JSON); uptime $\geq$ 95 %
SW-3	Local DB	PostgreSQL 15 (CSV possible for quick demo)	Inserts $\leq$ 1 s after reception; 30-day retention
SW-4	Front-end	HTML + CSS + JS (Plotly, Flatpickr)	Live charts $<$ 1 s redraw; responsive to 5cm–24cm screens
FN-1	Data logging	Persist timestamp, pH, EC, water-temp, air-temp, humidity every 2 min	No gaps $>$ 5 min while power is stable
FN-2	Filtering	Apply 101-tap low-pass FIR & scalar Kalman smoothing in Python	Filtered signal shown alongside raw
FN-5	Usability	Auth-guarded dashboard; selections for date & time window	Steps $\leq$ 3 clicks from login to plot
FN-6	Robustness	Works with no Internet/NTP; uses RTC-backup offset at boot	Time drift $\leq \pm$ 2 min / 24 h

Table 5: Mandatory Requirements — Local Application

3.3 Optional Requirements - Remote Monitoring Stack

Identifier	Requirement	Description / Success Criterion
OPT-1	Public dashboard	Mirror all sensor data to a cloud-hosted site reachable, in our case at `http://vasist.rrhess.de`.
OPT-2	Secure ingress	Terminate TLS with Let's Encrypt; enforce HSTS and rate-limited log-in.
OPT-3	Broker integration	Use MQTT (e.g., Eclipse Mosquitto) to publish live readings; client apps subscribe with QoS 1.
OPT-4	Band-width guard	Sync in 10-min batches if upstream ¡ 128 kbit s^{-1}; retry with exponential back-off.
OPT-5	Offline fallback	If WAN down 30 min, queue data locally and auto-flush when connectivity returns; no loss allowed.

Table 6: Optional Enhancement – Remote Web Portal

4 Previous Work

The present project builds on a series of earlier prototypes developed in the laboratory of Prof. Dr. Robert Heß. A fully cultivated tomato hydroponic bed had already been installed in Room 13.80 at HAW Hamburg campus at Berliner Tor 7. That code-base was a mixed collection: the original sensing and control routines were written in C++, by Mr. Thjorven Rubach (HAW Hamburg). In parallel, Prof. Heß maintained a remote server, for his personal website, where he could provide me with a sub-domain that accepted HTTP data posts but supported only PHP on the backend, so an initial web dashboard and MySQL data store were also created in PHP.

"The local application was developed in Python, the Raspberry Pi pico code in Micro-Python, the web application, whose server and remote database was provided by Prof. Heß, developed in PHP, as the sever only had PHP compiler, and at last the remote database was on MySQL server.

5 Design of the Solution

The design of this hydroponic monitoring and management system follows a staged architectural model. The system was developed in two progressive layers: a **self-contained local prototype** that created a local area within the BrosTrend™AC1200 Wi-Fi adapter range and is termed HydroSense, followed by a **server-connected remote system** for cloud-based data visualization and access. This layered approach was crucial not only to facilitate development and debugging but also to ensure future scalability and modular expansion.

5.1 Overall System Architecture

The complete system integrates three major layers:

1. **Hardware Layer**-Comprising analog and digital sensors connected to a Raspberry Pi .
2. **Local Processing and UI layer**-A Flask-based Python web app running locally on a PC or Pi, handling real-time logging, filtering, visualization, and calibration UI.
3. **Remote Server and Database Layer**-A web-based PHP/MySQL application hosted on a remote server, allowing centralized data storage and browser-based access.

Data flow from the sensor to the microcontroller, which transmits it via a local Wi-Fi network. to the local web application or over HTTP to the PHP server. Each layer is loosely coupled enabling independent development, testing and deployment.

5.2 Hardware Layer

5.2.1 Functional Overview

The microcontroller layer represents the first stage in the system's hierarchical architecture. It is primarily responsible for interfacing with the physical environment through various sensors, acquiring raw data such as temperature, humidity or motion signals. Once data is collected, it undergoes basic pre-processing such as scaling, threshold comparison, and formatting, before being passed on to the upper layers of the system.

This layer also governs rudimentary logic, including local thresholding (e.g. triggering an alert if a temperature exceeds a defined limit), and acts as the first point of validation before data is forwarded via serial or wireless communication. In its current configuration, the microcontroller sends output through a USB serial interface to the host machine.

This layer is responsible for:

- **Sensor data acquisition** (e.g. temperature, humidity, motion, etc.)
- **Pre-processing** (scaling, formatting, thresholding)
- **Initial decision logic** (e.g. sending alerts)
- **Communication** via serial or network interfaces (e.g. USB, UART or WiFi modules like ESP)

5.2.2 Input and Output Dynamics

In the microcontroller's operational cycle, input data originates from multi-sensor peripherals interfaced through analog-to-digital converters (ADC) and digital communication protocols such as I^2C. Inputs specifically include analog sensor signals for pH and TDS, digital readings from the DS18B20 temperature sensor, and atmospheric measurements via I^2C from the BME280 sensor. Following collection, these data points are processed into structured JSON payloads. The output of this structured data is transmitted via HTTP POST requests directed toward a remote PHP API endpoint, and optionally logged locally over UART serial communication for debugging purposes. This clear delineation of input-output flows aligns closely with the methodology demonstrated, emphasizing structured transformation of raw data into validated outputs.

Source/Destination	**Type**	**Details**
Sensors	Input	Analog/Digital
Thonny (Serial Output)	Output	JSON/CSV-encoded strings
Flask API endpoint	Output	WiFi module sends via HTTP

Table 7: Input/Output Design Differentiation

5.2.3 Internal State and Behavior

The microcontroller operates under a volatile state management model, meaning that no persistent storage is implemented across cycles or device rests. Each operational iteration independently captures and processes sensor data, temporarily maintaining sensor states and data validity flags during runtime only. After each successful transmission or identifies failure, the internal state is reset ensuring subsequent cycles start from a known, reliable state. This transient state management approach simplifies error handling and recovery, echoing the intermediate-state philosophy. The microcontroller can therefore be restarted or re-initialized without jeopardizing system integrity, provided the downstream layers implement their own fault tolerance.

The Microcontroller operates in a **looped polling mode**:

- Reads sensor data at set intervals (e.g., every 3 seconds)
- Maintains volatile state (current readings)
- No persistent storage (current readings)
- Implements conditional logic (e.g., "send if threshold exceeded")

5.2.4 Error Handling / Validation Points

The robustness of the microcontroller's operational cycle is reinforced through explicit validation and fault-handling mechanisms at each data processing stage. Sensor readings undergo validity checks against defined operational threshold (e.g. acceptable range checks for temperature, pH and TDS readings). Detected sensor anomalies or communication failures trigger immediate logging of explicit error messages. Similarly, the transmission stage incorporates HTTP response validation, logging successful (HTTP 200) or unsuccessful transmissions accordingly. This systematic fault-handling strategy not

only enhances system reliability, but also directly emphasis on intermediate validation checkpoints, providing clear guidance on operational integrity at each processing stage.

- **Sensor read failure** : Logs "error" in serial output.
- **Threshold breach** : Sends trigger immediately
- **Corrupt data** : Skip cycle, do not send.

5.2.5 Behavioral Flow and Activity Structure

The micocontrollers structure can be visualized as a looped activity diagram comprising initialization, data acquisition, validation, transmission, and delay stages. This tightly bound loop ensures deterministic timing and facilitates easy debugging , particularly in constrained environments where timing jitter or missed cycles could compromise data quality.

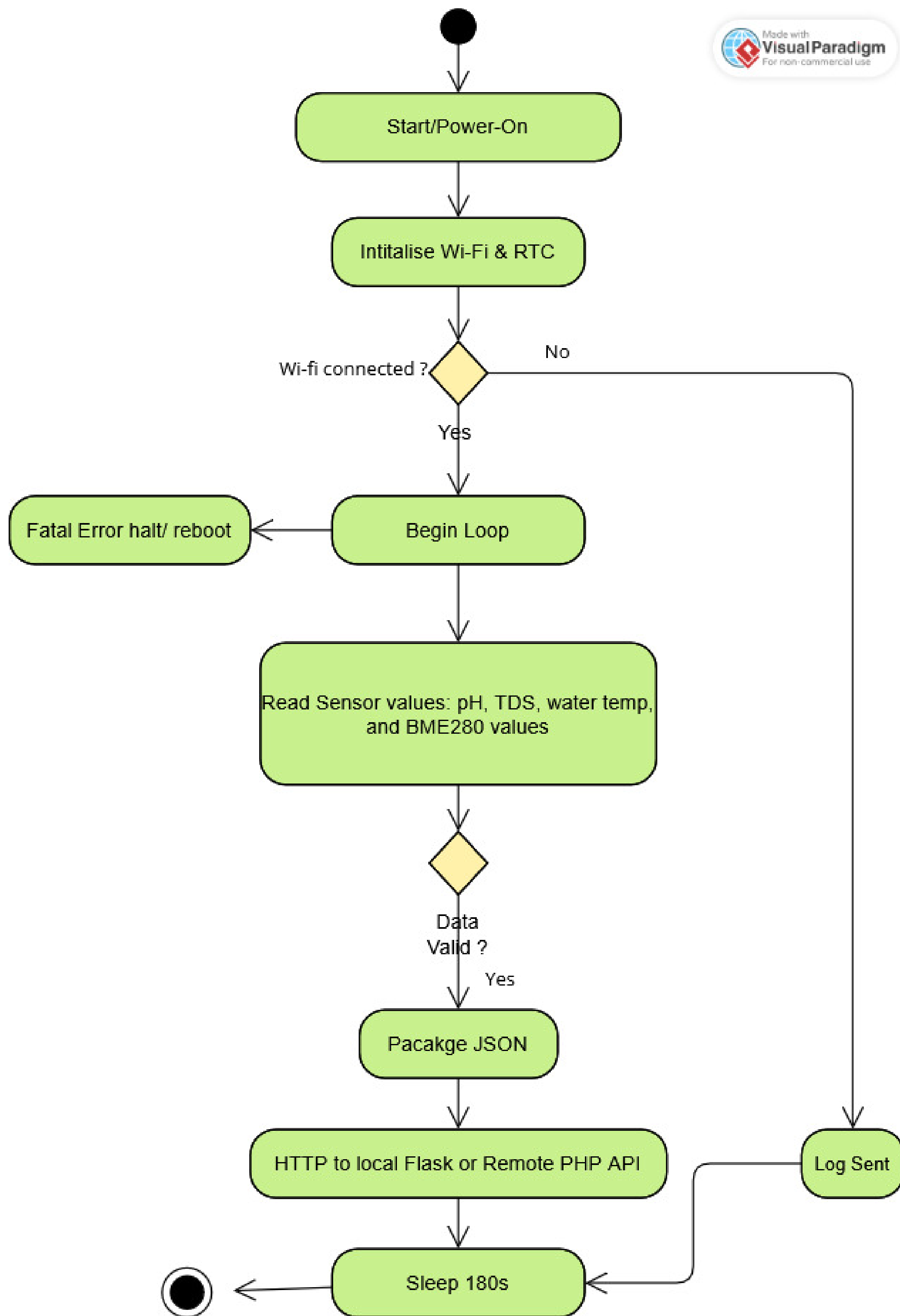


Figure 14: Activity Diagram for the Microcontroller layer.

5.2.6 Architectural Integration

This layer is the foundation of the full-stack system. It interfaces directly with the hardware (sensors) and indirectly with the software ecosystem via USB or network. In terms

of architectural layering, the microcontroller provides the raw input layer, feeding data into the application logic layer (Flask backend) and ultimately into the analysis and presentation layer (PHP or web front end). Its simplicity is intentional, keeping processing lightweight and energy-efficient while offloading complex logic to downstream services.

The modularity of this layer ensures that it can be swapped or upgraded independently. For instance, the Thonny environment currently provides the programming interface and serial monitor, but the microcontroller firmware is platform-agnostic and can be redeployed via Arduino IDE or PlatformIO if needed.'

The structure is analogous to the MVP pattern described in the thesis: the microcontroller acts as the **Model**, outputting data; the terminal or serial interface is the **View**; and the logic controlling sensor polling and data formatting forms the **Presenter**.

5.3 Local Processing and UI layer

5.3.1 Functional Overview

The local Flask application forms the **second** tier in the hydro-monitoring stack, delivering the first level of computational transformation on data originating from the microcontroller. The Flask service converts unstructured JSON sensor payloads into relationally stored, queryable records and generates user-friendly visualizations. Its responsibilities encompass **(i)** authenticated ingestion of HTTP POST events, **(ii)** syntactic and semantic validation of each payload, **(iii)** durable persistence ina relational store, and **(iv)** immediate presentation through REST endpoints, WebSocket broadcasts, server-rendered dashboards.

5.3.2 Input and Output Dynamics

Inputs. Each POST request to */ingest* carries a JSON object with fields **Timestamp (ISO-8601)**, **pH value**, **EC-value**, **Water temperature**, **Air Temperature** and **Humidity**. These packets are forwarded by the microcontroller once per acquisition cycle.

Outputs. After validating the service:

- **Persistent** records accessible through REST endpoints.
- **Rendered** HTML pages that provide tabular and graphical summaries for technicians working on-site.
- The PyQT front-end does not post new data; instead it listens passively and issues read-only GETs when the user presses the "Show Data" button.

5.3.3 Internal State Management

Unlike the microcontroller, the Flask layer maintains a persistent, durable state. The primary database stores each sensor event, which is calibrated coefficients and metadata. A secondary cache layer - implemented with Flask-Caching backed by a simple in-memory store-retains the most recent measurement per sensor to accelerate dashboard rendering and reduce query overhead at high ingest rates. Rollback semantics are governed by SQLAlchemy sessions: failed transactions automatically revert without corrupting previously committed rows, preserving ACID guarantees.

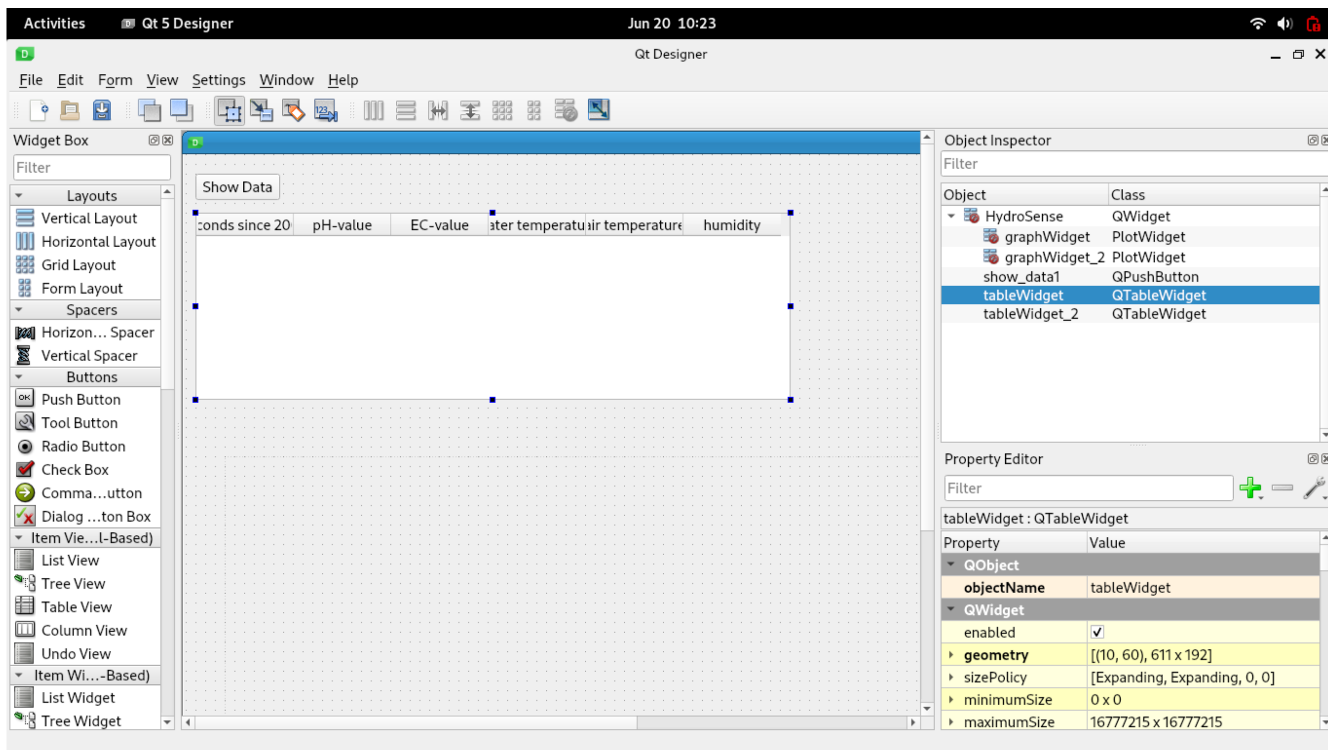


Figure 15: For the initial GUI development QT designer was used, which was then converted to CSS code as styles.css.

5.3.4 Behavioral Flow Activity Structure

On start-up the application loads configuration parameters (database URI, secret key, cache TTL) from environment variables, initializes the SQLAlchemy context, the principal execution loop is request-driven:

Operational logic is event-driven:

- **Start-up**. The app factory loads configuration, opens the Database connection, attaches blueprints, and schedules background tasks.
- **Ingestion**. The /Ingest: blueprint decodes JSON, begins a DB transaction.
- **Persistence**. Valid entities are inserted via ORM; failure provoke a 400 response.
- **Broadcast**. Upon commit the application signals dashboards.
- **Response**. The client receives HTTP201 with the canonical record.
- **Dashboard Rendering**. Get /dashboard queries cached aggregates (or recomputes on cache miss)

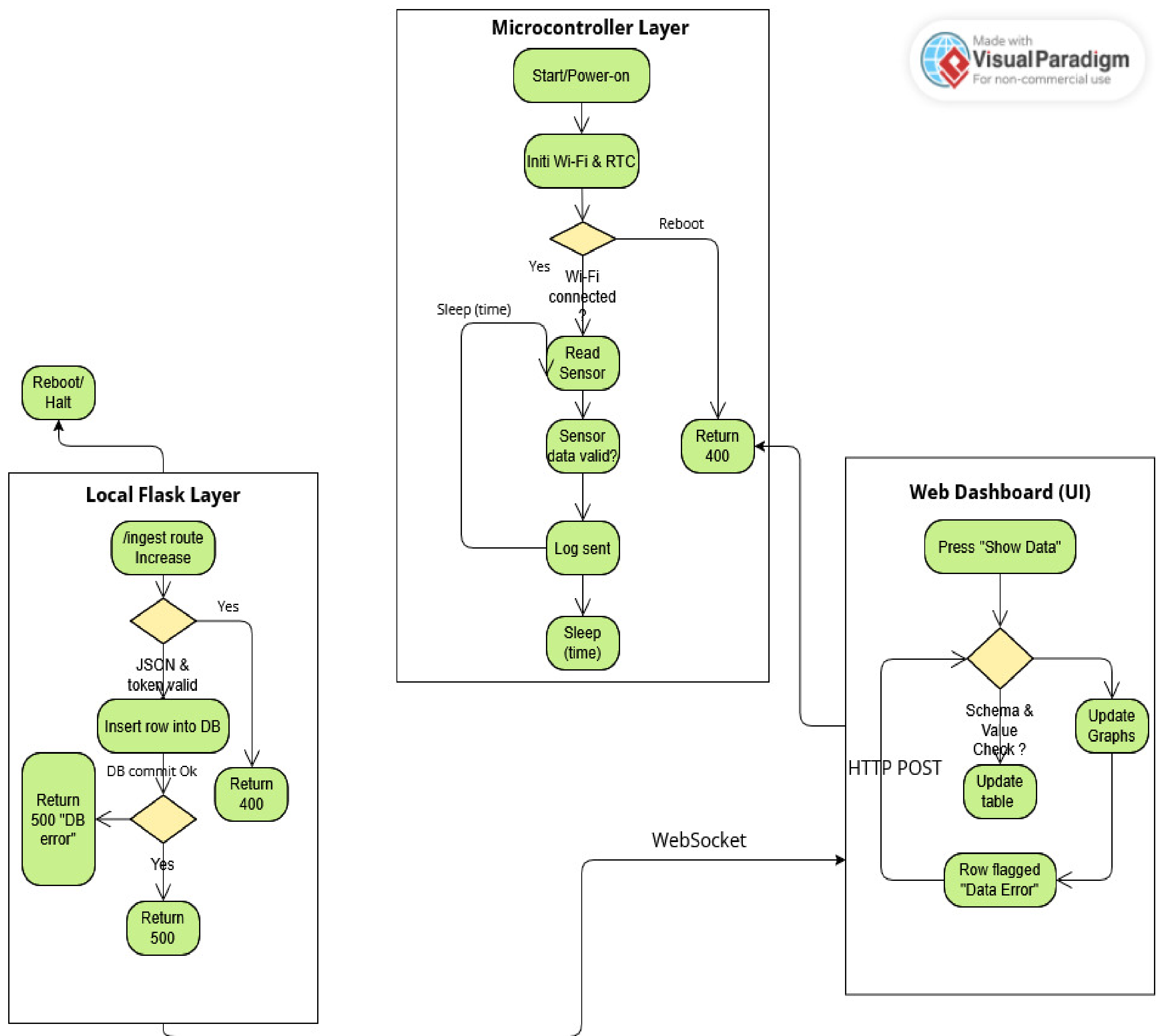


Figure 16: Activity Diagram for interaction between the local layer and hardware layer.

5.3.5 Validation and Fault Handling

Two concentric validation rings safeguard reliability. **Ring 1 (input validation)** reflects malformed or physically implausible readings (e.g., pH < 0 or >14). **Ring 2 (persistence validation)** captures database exceptions - unique constraint violations, disk I/O errors, or schema mismatches - logging them through Flask-Logging and returning HTTP 500 with correlation identifiers. A circuit-breaker wrapper throttles ingest if consecutive failures exceed a configurable threshold, preventing runaway error storms.

5.3.6 Architectural Integration

Since the QT designer GUI and the Acting as an *edge-side middle-ware*, the Flask layer decouples high-frequency sensor traffic from the intermittently connected cloud tier. Back-end jobs (APS-scheduler) forward locally verified records to the remote

5.4 Remote Processing and UI layer

5.4.1 Functional Overview

The Remote processing and UI layer provides an interactive user-facing web interface hosted remotely as a sub-domain (vasist.rrhess.de). It offers comprehensive functionality for accessing, processing, and visualizing sensor data collected by the hardware layer of the raspberry pi pico and the connected sensors. Primary functions include data retrieval from a remote MySQL database, real-time FIR filtering, secure user authentication, dynamic data visualization using Plotly.js, and intuitive user interactions enhanced by custom CSS for a coherent graphical user interface.

5.4.2 Input and Output Dynamics

Input:

- Sensor data sourced remotely from a MySQL database, containing parameters such as pH, Electrical Conductivity (EC), water and air temperature, and humidity.
- User-generated inputs for authentication (usernames and passwords), data selection via data pickers (Flatpickr integration), and dynamic user interactions via buttons and drop-down menus.

Output

- Interactive graphs rendered by Plotly.js, displaying FIR-filtered sensor data clearly and responsively.
- Structured and user-friendly data tables for detailed inspection of sensor readings.
- JSON-formatted data responses delivered through remote PHP scripts to facilitate fronted AJAX requests.
- Secure user authentication results, providing session handling with clear login and logout functionalities.

5.4.3 Internal State Management

- **User Sessions**: Maintained through PHP session variables such as $SESSION$, ensuring secure and persistent user authentication states.
- **Data Cache Management**: In-memory caching of sensor data in JavaScript($allData$) minimizes unnecessary remote data fetches significantly improving UI responsiveness and efficiency.
- **UI State Tracking**: Current user selection (date, filters, visualization preferences) are tracked within the front-end state to maintain consistency and streamline the user experience across interactions and page refreshes.

5.4.4 Behavioral Flow Activity Structure

1. **Initialize**: Upon initial page load, the front-end issues AJAX requests to remote endpoints (data.php) to retrieve initial sensor data, initializing user interface elements such as date pickers and drop-down selectors.

2. **Interactive Event Handling**: User actions such as logging in, selecting dates, and refreshing data are captured through event listeners, dynamically updating the internal state and immediately reflecting changes and in the visualization layers.

3. **Automatic Data Updates**: JavaScript-driven polling (*setInterval()*) periodically refreshes the sensor data, ensuring continuous real-time updates within the visualization layers.

4. **Data Processing**: FIR filtering is performed remotely through PHP scripts, converting raw data into structured JSON suitable for front-end consumption, ensuring accuracy and clarity in data presentation.

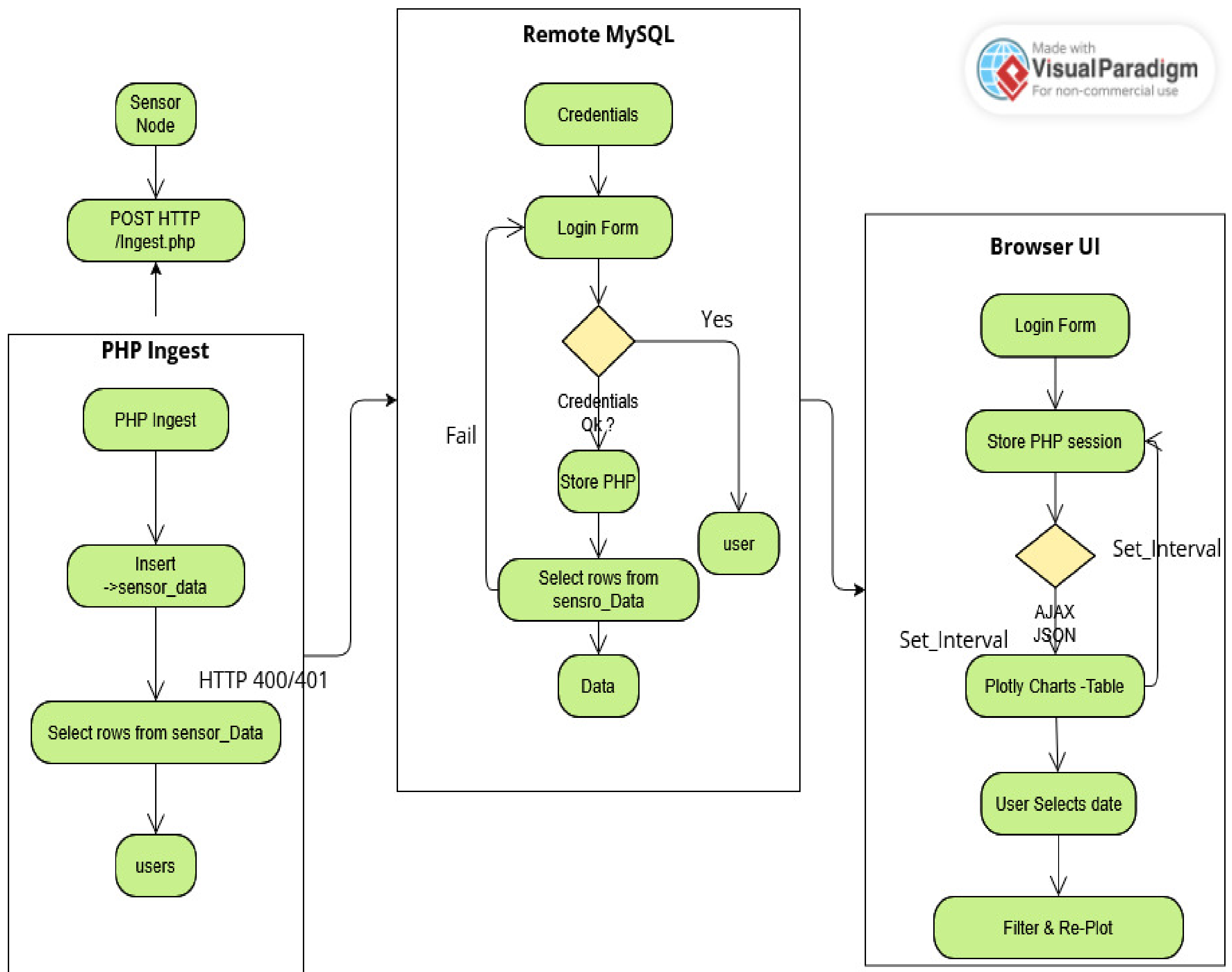


Figure 17: Activity Diagram for interaction between the Remote layer and hardware layer.

5.4.5 Validation and Fault Handling

- **Data Validation**: Backend PHP scripts thoroughly verify input and output data consistency, ensuring JSON responses are accurate and reliable.
- **Exception Management**: Comprehensive PHP try-catch structures handle potential faults, such as connectivity issues or data inconsistencies, providing clear JSON-based error messaging and appropriate HTTP error responses.
- **Authentication Security**: Strong user credential management employing PHP's secure hashing functions, along with robust session verification processes, ensures secure and controlled access to the system.

5.4.6 Architectural Integration

This Remote Processing and UI layer seamlessly integrates into the broader Hydroponics solution through well-defined interfaces:

- **Remote Data Storage**: Interaction with a centralized remote MySQL database, ensuring and efficient data retrieval.
- **Processing Logic Integration**: PHP scripts hosted remotely provide real-time data processing and user authentication.
- **Front-end User Interface**:The web-based UI is enhanced through interactive Plotly.js visualization and user-friendly CSS styling *style.css*, providing a cohesive and engaging user experience.

This approach ensures a modular, scalable, and user-centric design, effectively managing remote data visualization, real-time responsiveness and secure user interactions within the Hydroponics ecosystem.

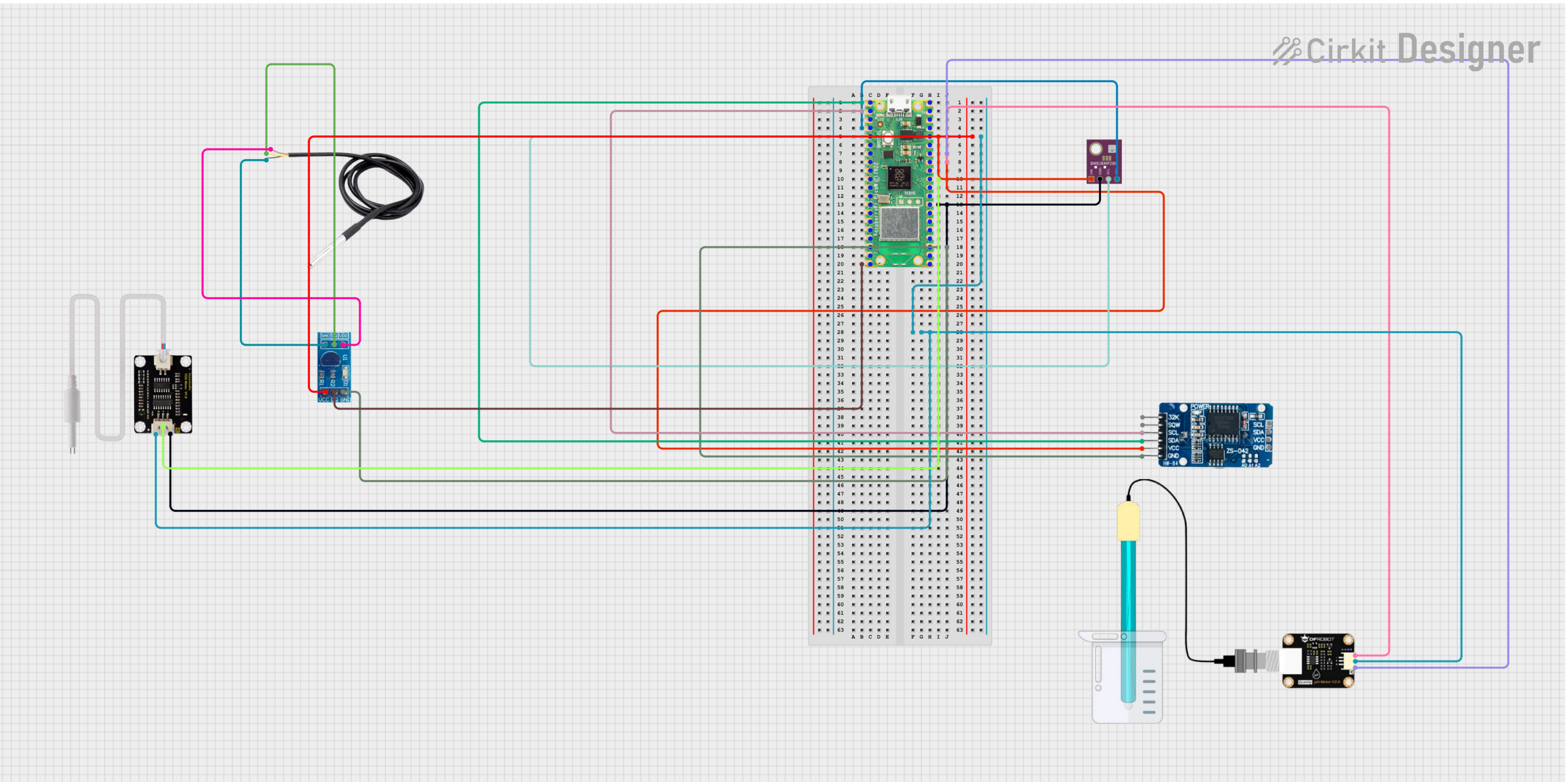


Figure 18: Pin Mapping of Sensor and Microcontroller. Done with the help of the Raspberry Pi Picos official pinout daigram. [15]

6 Implementation Details

6.1 Hardware Layer

At the foundation of the the system lies a Raspberry Pi Pico running Micro-Python 1.22. The microcontroller is chosen for its dual-core Cortex-M0+, low power draw and ample GPIO header, which together accommodate a heterogeneous mix of digital and analogue peripherals. Tow independent I^2C busses are configured: The first addresses a BME280 environmental module at 0x76, while the second services a DS3231 real-time clock at 0x68. One-wire support on GPIO 15 connects a DS18B20 probe for precise water-temperature measurement. Analogue pH and TDS probes interface through ADC channels 1 and 0 (GP27 and GP26 respectively), with a bidirectional level shifter translating the 5 V probe outputs to the the Pico's 3.3 V domain.

6.1.1 Driver-Level Architecture

Each sensor is wrapped by a concise, single responsibility driver - bme_reader.py, water_temp_reader.py, ph_reader.py, tds_reader.py, and rtc_clock.py. The drivers normalize raw transducer outputs into SI-friendly units by applying empirically derived calibration factors: the pH routine executes a ten-sample moving average before transforming voltage into pH via a linear slope intercept model, whereas the TDS routine fits a third-order polynomial to convert conductivity millivolts into parts per million. A supervising script, main.py, coordinates all acquisitions in a two-minute interval. After aggregating the readings into a JSON object, it attaches an epoch-second timestamp sourced from the DS3231, signs the payload with an HMAC, and transmits the result to the edge API using an HTTPs POST. If network connectivity is unavailable, the code is checked 3 times with an interval until it dies.

A supervisor script, main.py, runs the data collection show on a 120-second cadence. After synchronous sampling, the routine stamps the reading with seconds since 2000-01-01 00:00:00 UTC, suing the DS3231 as the authoritative clock, which is then converted to regular time by the python/php script.

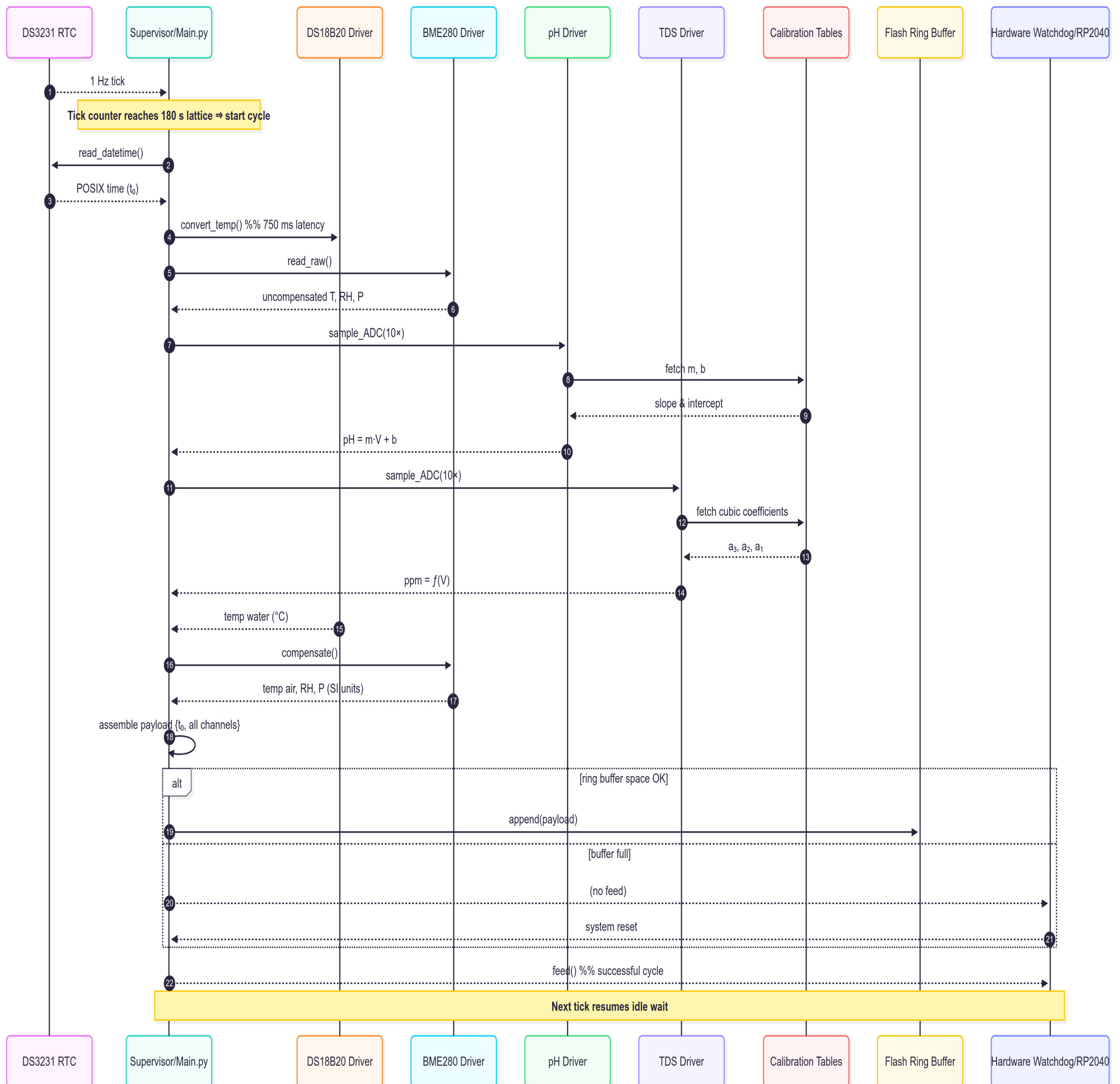


Figure 19: Sequence Diagram for the Hardware Layer (made by mermaid charts)

6.1.2 Calibration Logic - Analog Sensors

1. **DS18B20 Water-Temperature Driver (water_temp_reader.py)**: The DS18B20 us a Digital temperature sensor that communicates via the 1-Wire protocol, which is handled in code using the *one-wire* and *DS18x20 modules* in Micro-python:

```python
import machine, onewire, ds18x20, time

# Connect DS18B20 data pin to GP15

data_pin = machine.Pin(15)
ds = ds18x20.DS18X20(onewire.OneWire(data_pin)) #using libraries
roms = ds.scan()

```

Figure 20: Code snippet from the file: *water_temp_reader.py*

The sensor is scanned, and its unique ROM address is stored, which allows addressing multiple DS18B20 sensors on the same 1-Wire bus. The modules one-wire and ds18x20 handle the low-level communication with the sensor; however, it is important to note that the DS18B20 does not require manual calibration, as it comes pre-calibrated from the factory with built-in digital compensation

2. **TDS (Total-Dissolved-Solids) Driver (tds_reader.py)**: The driver first converts a 16-bit DC word to volts and then evaluates a cubic function that linearizes the chemistry of the probe.

```python
def read_tds_voltage():
    raws_Data= adc.read_u16()
    voltage= raws_Data * 3.3 / 65535
    return voltage

def read_tds_value():
    """Looping to change voltage to TDS"""
    voltage = read_tds_voltage()
    tds = (133.42* voltage**3 - 255.86* voltage**2 + 857.39* voltage)* 0.5
    return round(tds, 2)

```

Figure 21: Code snippet from the file: *tds_reader.py*

Those three coefficients originated from a nine-point 25°C KCL series (0 -¿ 2000 ppm) fitted by ordinary least squares ($R^2 \approx 0.999$). Every quarter the probe is checked in a 342 ppm standard: if the measured value deviates by ±2 ppm the full nine-point calibration is repeated and the polynomial constants are rewritten.

3. **pH Driver - (ph_reader.py)**: The sensor measures voltage which corresponds to the hydrogen ion concentration in the solution. This voltage is typically in the range of 0 to 3V depending on the pH of the liquid. The signal was read using the ADC pin (GPIO27) of the pico.

   A three-point calibration process was performed using standard buffer solutions at pH 4.0, pH 7.0, and pH 10.0. For each buffer:

   - Multiple voltage readings were taken with the sensor immersed and the solution stable.
   - The average voltage was computed per buffer point.
   - This resulted in three (voltage, pH) pairs

Table 8: Sample Calibration Data

pH Buffer	Average Voltage (V)
4.00	~0.56 V
7.00	~1.02 V
10.00	~1.43 V

Using the recorded calibration points, **linear regression** was applied to derive the slope and intercept of the line:

$$pH = m \cdot Voltage + b$$

Based on your final calibrated set, the equation used in your code was:

$$pH = 4.755 \cdot V + 2.043$$

Where:

- V is the ADC voltage (in volts),
- $m = 4.755$ is the slope of the fitted line,
- $b = 2.043$ is the y-intercept.

After the values for Slope and Intercept are derived, the function *read_ph_voltage()*, then calculates and provides with the Voltage after the average of raw values are fetched from the sample.

```
8
9   def read_ph_voltage( samples=10, delay=0.01):
10      """ Read Values"""
11      total = 0
12      for _ in range(samples):
13          total += adc.read_u16()
14          time.sleep(delay)
15      avg_raw = total / samples
16      voltage = avg_raw * 3.3 / 65535  # Convert to voltage
17      return voltage
18
19  def read_ph_value():
```

Figure 22: Code snippet from the file: *ph_reader.py*

4. **Environmental Sensor (BME280) - (bme_reader.py)**: The BME280 environmental sensor was utilized in conjunction with an open-source Micro-python driver obtained from the official GitHub repository by Robert HH. [16]. This driver implements Bosch's compensation algorithms and extracts the required factory calibration coefficients from the sensor's onboard **EEPROM**. As such, all sensor readings - including air temperature and humidity — are digitally compensated and do not require additional manual calibration. This approach ensures stable, reliable, and repeatable environmental measurements across multiple sampling intervals.

```
1    from machine import Pin, I2C
2    import bme280_float as bme280   #From the Robert HH.
3
4    # Setup I2C for BME280 (bus 1, GP2 = SDA, GP3 = SCL)
5    i2c = I2C(1, scl=Pin(3), sda=Pin(2), freq=400000)
6    bme = bme280.BME280(i2c=i2c, address=0x76)
7
```

Figure 23: Code snippet from the file: *bme_reader.py*

5. **Real-Time Clock (*rtc_clock.py*)**: The RTC system is done by integrating the DS3231 Real-Time Clock (RTC) module. This module is connected via the I2C interface using GPIO pins 0 (SDA) and 1 (SCL) on the Raspberry Pi Pico, and provides highly precise time tracking down to the second. The DS3231 contains an onboard temperature-compensated crystal oscillator and backup battery support, a custom function, *seconds_since_2000()*, computes the total number of second elapsed since the epoch (January 1, 2000). This timestamp is essential for uniquely tagging each batch of sensor data before transmission to the remote server, allowing for consistent chronological ordering and comparison.

```
48  def seconds_since_2000()  :
49      try:
50          rtc_time =  read_rtc_time_struct()
51          t_current =  time.mktime(rtc_time)
52          t_base_2000  = time.mktime((2000, 1, 1, 0, 0, 0, 0, 0))
53          return int(t_current  - t_base_2000)
54      except Exception as e:
55          print("Failed to get RTC time:", e)
56          return 0  # Fallback value
57
58  # Haupt Testing Block
59  if __name__ == "__main__":
60      print("Time Right Now", read_real_time())
61      print("Seconds that have passed since year 2000", seconds_since_2000())
62
```

Figure 24: Code snippet from the file: *rtc_reader.py*

The module also includes a function set_rtc_time() to manually initialize the time on the DS3231 module, ensuring flexibility in case the device is powered on in an environment without automatic time synchronization like NTP.

6.2 Local Application

The developed local web-based visualization system provides an interactive dashboard for monitoring sensor data collected by a Raspberry Pi Pico, and transmitted to a PostgreSQL database. The application stack consists of PHP for data ingestion, a PostgreSQL database, for storage, and a Python Flask server that processes and displays these data in real time through a web interface.

Sensor data (e.g., pH, EC, temperature, humidity) is sent from the Raspberry Pi Pico directly to the Flask application's /insert endpoint via HTTP GET requests. The Flask application receives and stores the data using SQL commands.

Local Flask Application The Flask application (*app.py*) handles:

- Serving the HTML dashboard

- Providing a /data endpoint to supply sensor data as JSON
- Querying the database
- Applying filtering techniques (FIR and Kalman)

Structure of *app.py*:

```
1  from flask import Flask, render_template, jsonify
2  import psycopg2
3  import numpy as np
4  from scipy import signal
5  import datetime
6
7  app = Flask(__name__)
```

Figure 25: Code snippet from the file: *app.py (1)*

Data Loading Function:

```
def load_data():
    hostname = 'localhost'
    database = 'Hydrolog'
    username = 'postgres'
    password = '12345'
    port_id = 5432

    try:
        connection = psycopg2.connect(
            host=hostname,
            database=database,
            user=username,
            password=password,
            port=port_id
        )
        cursor = connection.cursor()
        cursor.execute('SELECT * FROM "Hydrolog1"."hydro_sensor2" LIMIT 1000;')
        rows = cursor.fetchall()
```

Figure 26: Code snippet from the file: *app.py (2)*

Time Conversion Utility:

```
def seconds_to_datetime_str(seconds):
    base_time = datetime.datetime(2000, 1, 1)
    time = base_time + datetime.timedelta(seconds=seconds)
    return time.strftime('%d.%m %H:%M')
```

Figure 27: Code snippet from the file: *app.py (3)*

6.2.1 Sequence Diagram

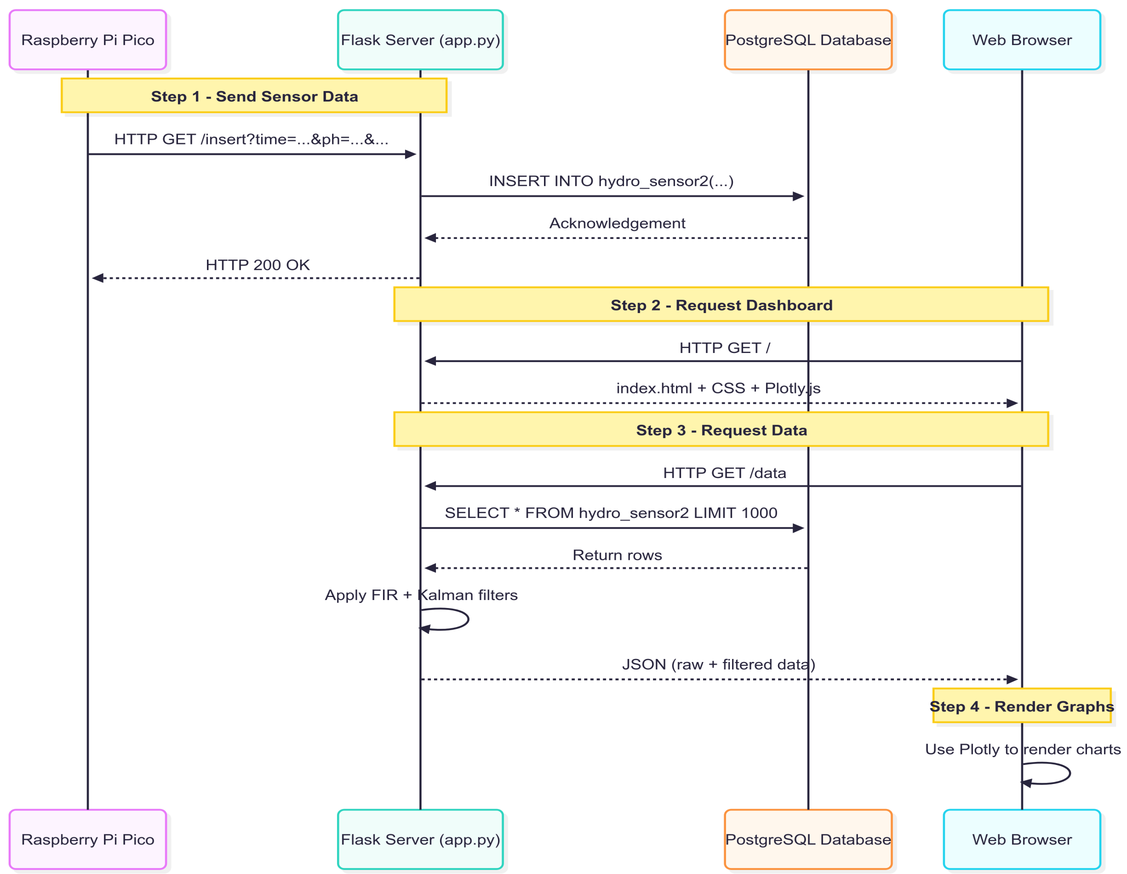


Figure 28: Sequence Diagram for Local Application, made by Mermaid Chart

6.2.2 Filtering Techniques

To enhance data reliability, two filters are applied to the pH value readings.

1. **Finite Impulse Response (FIR) Filter** An FIR filter smooths the data using a linear combination of previous input samples, it is a 101-tap filter (i.e. order = 100). It is low-pass, meaning it allows low-frequency signals (slow changes in pH) to pass while attenuating high-frequency components (fast spikes or noise), designed using windowed sinc method via *scipy.signal.firwin()*. This window function defaults to Hamming, which reduces side lobes (ringing) provides a smooth response.

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

Where:

- $h[k]$: Filter coefficients
- $x[n-k]$: Input data
- N: Number of filter taps

In the Implementation:

```
42  def apply_fir_filter(y_values, time_values):
43      time_intervals = np.diff(time_values)
44      average_time_interval = np.mean(time_intervals)
45      sampling_rate = 1 / average_time_interval
46      nyquist_rate = sampling_rate / 2
47      cutoff_fraction = 1 / 20
48      cutoff_frequency = cutoff_fraction * nyquist_rate
49      num_taps = 101
50      fir_coeff = signal.firwin(num_taps, cutoff_frequency, fs=sampling_rate)
51      y_smooth = signal.lfilter(fir_coeff, 1.0, y_values)
52      return y_smooth
```

Figure 29: FIR-filter implementation from *app.py*

This approach dynamically adjusts the sampling rate based on the time intervals from the dataset.

Why is it needed ?:

Real-world sensors often give fluctuating, noisy readings. A FIR filter:

- Smooths out erratic jumps.
- Gives consistent trends.
- Retains phase (important for time series).

2. **Kalman Filter** The Kalman filter provides a recursive solution for estimating the true state of a process, when the measurements contain noise. The one developed here is 1D discrete-time scalar Kalman filter for one variable (pH values). It's an optimal estimator that accounts for uncertainty in both the sensor measurements and the model prediction.

   Equations:

   - **Prediction**: $x_{\text{pred}} = x_{\text{est}}[k-1]$ $P_{\text{pred}} = P[k-1] + Q$

   - **Update**:

   $$K = \frac{P_{\text{pred}}}{P_{\text{pred}} + R} \quad x_{\text{est}}[k] = x_{\text{pred}} + K(y[k] - x_{\text{pred}}) \quad P[k] = (1-K) \cdot P_{\text{pred}}$$

   Where:

   - Q (Process noise): Expected change in the system
   - R (Measurement noise): Expected noise in the sensor data
   - K (Kalman Gain): Adaptive weight, higher as the measurement becomes more reliable.

In the Implementation:

```
54  def apply_kalman_filter(values):
55      n = len(values)
56      x_est = np.zeros(n)
57      P = np.zeros(n)
58      Q = 1e-5  # Process variance
59      R = 0.1 ** 2  # Measurement variance
60
61      x_est[0] = values[0]  # Initial estimate
62      P[0] = 1.0  # Initial estimate error
63
64      for k in range(1, n):
65          # Prediction step
66          x_pred = x_est[k-1]
67          P_pred = P[k-1] + Q
68
69          # Update step
70          K = P_pred / (P_pred + R)
71          x_est[k] = x_pred + K * (values[k] - x_pred)
72          P[k] = (1 - K) * P_pred
73
74      return x_est
75
```

Figure 30: Kalman-filter implementation from *app.py*

Why is it needed ?

- Unlike the FIR filter, Kalman filtering reacts faster to dynamic changes.
- Its good when sensor noise is non-uniform or data is missing/intermittent.
- Ideal when you want to balance responsiveness and smoothness.

Table 9: Comparison of FIR Filter and Kalman Filter

Feature	FIR Filter	Kalman Filter
Type	Low-pass, 100th order	1D scalar recursive
Response	Smooth, fixed delay	Adaptive, less delay
Memory	Requires full window (101 samples)	Only last estimate + noise
Use-case	Stable smoothing	Fast changes, adaptive fusion
Phase preservation	Yes	Not exact

6.2.3 Flask Endpoints

Flask routes handle both front-end rendering and back-end data delivery. The application consists of two primary endpoints:

(a) **Homepage Endpoint (/)**: This route handles the front-end of the application. When users navigate to the root URL of the web server, Flask responds by rendering and serving the *index.html* page. This HTML page includes JavaScript code that interacts with the backend to fetch and visualize sensor data.

```
77  @app.route('/')
78  def index():
79      return render_template('index.html')
80
```

Figure 31: Calling index.html implementation from *app.py*

(b) **Data Endpoints (/data)**: This is a RESTful API endpoint that delivers the sensor data in JSON format. It is responsible for:

- Loading raw data from the PostgreSQL database.
- Applying both FIR and Kalman filters to the pH readings.
- Structuring and returning the data in a format suitable for front-end consumption.

This endpoint is typically triggered by JavaScript in the front-end when the user interacts with the interface (e.g., clicking a button to fetch or refresh data).

Together, these endpoints ensure a clean separation of concerns, allowing the backend to focus on logic and computation while the front-end focuses on presentation and user interaction.

```
83   def data():
84       rows = load_data()
85       if not rows:
86           return jsonify({"error": "No data found"}), 500
87
88       x_values = [seconds_to_datetime_str(row[0]) for row in rows]
89       y_values = [row[1] for row in rows]  # pH Value
90       ec_values = [row[2] for row in rows]
91       water_temp = [row[3] if row[3] is not None else 'N/A' for row in rows]
92       air_temp = [row[4] if row[4] is not None else 'N/A' for row in rows]
93       humidity = [row[5] if row[5] is not None else 'N/A' for row in rows]
94       time_values = [row[0] for row in rows]
95
96       y_smooth = apply_fir_filter(y_values, time_values)
97       y_kalman = apply_kalman_filter(y_values)
98
99       return jsonify({
100          'x_values': x_values,
101          'y_values': y_values,
102          'ec_values': ec_values,
103          'water_temp': water_temp,
104          'air_temp': air_temp,
105          'humidity': humidity,
106          'y_smooth': y_smooth.tolist(),
107          'y_kalman': y_kalman.tolist()
108      })
109
```

Figure 32: RESTful API endpoint implementation from *app.py*

6.2.4 Front-end with Plotly

The front-end *index.html* uses Plotly.js to render graphs for raw pH, FIR Filtered pH, and Kalman-filtered pH. Data is fetched dynamically when the user clicks the "Show Data" button.

```
64                          tableBody.appendChild(row);
65                      });
66
67                      // Graph 1: Raw pH Values
68                      Plotly.newPlot('graph-raw', [{
69                          x: data.x_values,
70                          y: data.y_values,
71                          type: 'scatter',
72                          mode: 'lines+markers',
73                          name: 'Raw pH'
74                      }], {
75                          title: 'Raw pH Values',
76                          xaxis: { title: 'Time' },
77                          yaxis: { title: 'pH Value' }
78                      });
79
80                      // Graph 2: FIR Filtered
81                      Plotly.newPlot('graph-fir', [{
```

Figure 33: Plotly.js was used to make interactive graphs in *index.html*

6.3 Remote Application

The remote monitoring application provides a secure and user-friendly web interface that allows users to remotely access, analyze, and visualize sensor data. The application is implemented primarily using PHP for backend operations and MySQL as the database solution, combined with HTML, CSS, and JavaScript for the front-end. The development called for a full-stack development and was achieved as such:

User Management:

- *registration_.php* for creating new user accounts.
- *login.php* for authentication existing users.
- *logout.php* for securely terminating sessions.

Database Connection:

- *db_connect_registration.php* and *db_connect.php* manage database connections separately for user authentication and data processing.

Data Handling and Visualization:

- *data_processing.php* handles backed data retrieval and formatting.
- *data.php* provides a frontend interface to display the sensor data.

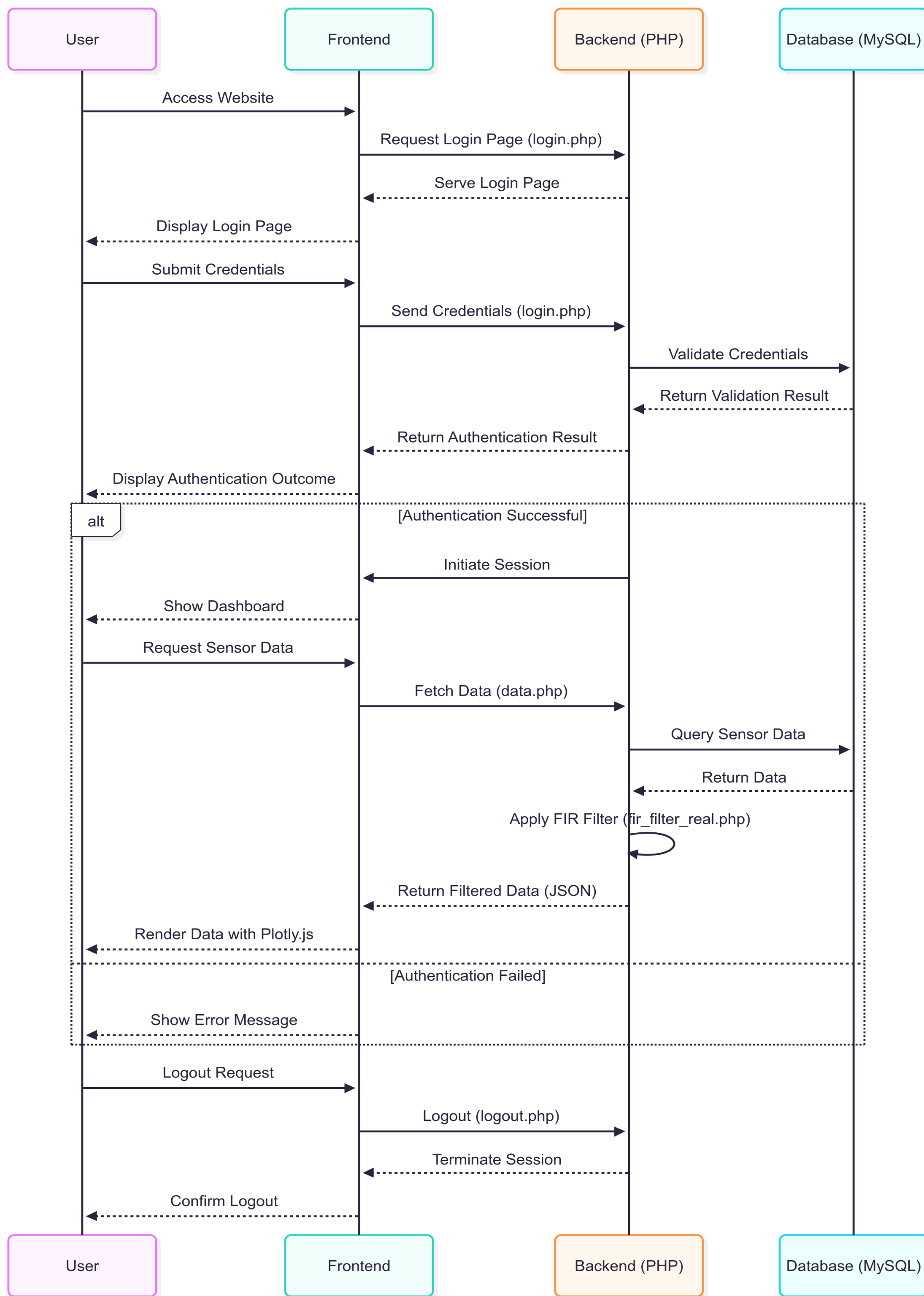


Figure 34: Sequence Diagram for the Remote Application made by Mermaid.

6.3.1 User Authentication

Registration (*registration.php*)

User registration is a important step in establishing secure access to the remote monitoring application. It allows new users to create accounts by providing information such as username, email, and password. This script ensures that credentials are handled securely before they are stored in the database.

```
16
17  require_once __DIR__ . '/db_connect_registration.php';
18
19  if ($_SERVER['REQUEST_METHOD'] === 'POST') {
20      $username1 = trim($_POST['username'] ?? '') ;
21      $password1 = $_POST['password'] ?? '' ;
22      if (!$username1 || !$password1) {
23          die('Username and Password Required') ;
24      }
25      $hash1 = password_hash($password1, PASSWORD_DEFAULT) ;
26      $stmt = $pdo->prepare("INSERT INTO users (username, password_hash) VALUES (?, ?)");
27      try {
28          $stmt->execute([$username1, $hash1]) ;
29          echo "New User Created" ;
30      } catch (PDOException $err1) {
31          echo "Error: " . $err1->getMessage() ;
32      }
33  }
```

Figure 35: Code Snippet for user registration. (*registration.php*)

Login (*login.php)*:

Validates user credentials by comparing input credentials with hashed passwords stored in the database.

Steps in the Login Script:

- Capturing User Input
- Fetching user record from database
- Verifying password
- Session Initialization and Redirection/Feedback

Logout (*logout.php*)

- Session Initialize
- Session Data Clearance/ Destroying the Session
- Redirection to login.

```
1  <?php
2  session_start();
3  session_unset();
4  session_destroy();
5  header("Location: login.php");
6  exit;
7  ?>
8
```

Figure 36: Code snippet for logout logic. (*logout.php*)

6.3.2 Database Connection

The database connections in the implemented PHP scripts are implemented twice, once for the table *Users* and for table *sensor_data*. Script implements PHP's **PDO** (PHP Data Objects) extension, which provides a secure, efficient and consistent interface for interacting with databases.

```
try {
    $dsn = "mysql:host=$host;port=$port;dbname=$dbname;charset=utf8mb4";
    $pdo = new PDO($dsn, $user, $pass, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);
```

Figure 37: Code snippet for PDO logic, implemented in both (*db_connect.php*) and (*db_connect_registration.*)

Two separate scripts deal with two tables mentioned above:

db_connect.php:

This script manages the main data operations (sensor data) that typically interact with PostgreSQL. It primarily supports the remote data collection logic, ensuring sensor data (e.g., pH, EC, temperature, humidity) can be securely and efficiently stored and retrieved.

db_connect_registartion.php

This second script is specifically dedicated to user management (registration, authentication, session handling). It typically interacts with a user-focused database schema.

Why two Connections ?

- **Security and Isolation**: Separating user data (credentials, profiles) from operational data (sensor readings) improves security by isolating sensitive user data from less sensitive operational data.
- **Maintainability**: Distinct databases simplify schema management and improve clarity.
- **Performance**: Each database can be modified separately based on its usage patterns.

6.3.3 Data Visualization

Data Retrieval (*data_processing.php)*

This script processes raw sensor data retrieved from a PostgreSQL database, applies a Finite Impulse Response (FIR) filter to smooth noisy data, converts timestamps, and structures the data for front-end consumption. The script contains logic to securely connect to a PostgreSQL database using PDO, then it retrieves data through the **load_data()** function (defined externally in **db_connect.php** file):

Data Extraction and Preparation

The script processes rows to populate arrays ($\$time_values$, $\$y_values$, $\$ec_values$, etc.):

```
29    foreach ($rows as $row) {
30        if (!isset($row['seconds_since_2000']) || !is_numeric($row['seconds_since_2000'])) {
31            continue; // skip bad rows
32        }
33        $time_values[]   = (float) $row['seconds_since_2000'];
34        $y_values[]      = isset($row['ph_value']) ? (float)$row['ph_value'] : null;
35        $ec_values[]     = isset($row['ec_value']) ? (float)$row['ec_value'] : null;
36        $water_temp[]    = isset($row['water_temperature']) ? (float)$row['water_temperature'] : 'N/A';
37        $air_temp[]      = isset($row['air_temperature']) ? (float)$row['air_temperature'] : 'N/A';
38        $humidity[]      = isset($row['humidity']) ? (float)$row['humidity'] : 'N/A';
39    }
40
```

Figure 38: Code validates sensor values and gracefully handles missing data (*null* or *'N/A'* from script *data_processing.php*

FIR Filtering of Sensor Data

- **Sampling Rate Calculation:** Determines frequency based on timestamp differences.

$$\text{Sampling Rate (Hz)} = \frac{1}{\text{average interval between samples}}$$

- **Normalized Cutoff:** Typically set to a low value (e.g., 0.05) representing a low-pass filter, calculated as:

$$\text{Cutoff Frequency} = \frac{\text{Desired Frequency}}{\text{Nyquist Frequency}}$$

- **Coefficients Calculation (calculateCoefficients):** FIR coefficients ($h[k]$) are generated to define the response of the FIR filter. Typical FIR equation:

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

- **Filter Application:** Uses convolution of coefficients with input (y_values) to produce a smoothed result (y_smooth)

```php
41    // ---- FIR Filter Logic ----
42    $numtaps = 601;   //FIR Filter Order (600th order)
43    $y_smooth = $y_values; // fallback to raw by default
44    try {
45        $sampling_rate = FIRFilter::calculate_sampling_rate($time_values); // Hz
46        if ($sampling_rate !== null && count($y_values) >= $numtaps) {
47            $normalized_cutoff = FIRFilter::calculate_normalized_cutoff($sampling_rate); // typically 0.05 (1/20)
48            $coefficients = FIRFilter::calculateCoefficients($numtaps, $normalized_cutoff);
49            $y_smooth = FIRFilter::apply($y_values, $coefficients);
50        }
51    } catch (Exception $e) {
52        // In case of error, just use the raw y_values
53        $y_smooth = $y_values;
54    }
```

Figure 39: Code for defining the FIR-Filter logic, high number taps, for a smoother curve from script *data_processing.php*

Safe FIR Filter Function (*safe_fir_filter()*)

The safe_fir_filter() function adds robustness by checking the size of the dataset:

```php
88  // Edge case handling for small datasets
89  function safe_fir_filter($y_values, $time_values) {
90      if (count($time_values) < 101) {
91          return $y_values;
92      }
93      return get_sensor_data()['y_smooth'];
94  }
95  ?>
```

Figure 40: Code snippet for circumventing FIR logic failure from *data_processing.php*

- FIR filters require sufficient data points; if inadequate, the original dataset is returned.
- Enhances stability and avoids runtime errors when data points are sparse.

7 Test and Validation

To verify and validate the functionality and reliability of the hydroponics monitoring system implemented, several key tests and validation processes were carried out. The goal was to ensure accuracy, repeatability, usability, and reliability under various operating conditions.

7.1 Multi-day Readings at various pH benchmarks

Multiple continuous tests were performed to assess the stability and accuracy of pH measurements. Sensor readings were recorded repeatedly at predefined pH benchmarks (e.g., 4.0, 7.0, 10.0) across multiple days to monitor potential drift, stability, and long-term accuracy.

Procedure:

- Senors were immersed in standardized close to calibration standards.
- Measurements were automatically logged two minutes over a 72-hour period.
- The full data path (Raspberry Pi Pico $\rightarrow$ Flask/PHP $\rightarrow$ FIR/Kalman filters $\rightarrow$ dashboard) was exercised.

Expected Outcome:

- Consistent and stable readings close to calibration standards.
- Minimal sensor drift over time, typically less than ± 0.2 pH unit.

Assessment Criteria:

- Stability: standard deviation of measurements over time.
- Drift: difference between initial and final measured values.

Figure 41: Calibration was done periodically for a week, with liquids of different pHs

7.2 Noise and Repeatability Metrics

Sensor data inherently contain some level of noise and variability. To ensure system reliability, noise characteristics were qualified using statistical metrics before and after filtering.

Procedure

- Raw sensor data collected under controlled conditions.
- Application of FIR and Kalman filters to raw data.
- Analysis of variance, standard deviation, and repeatability.

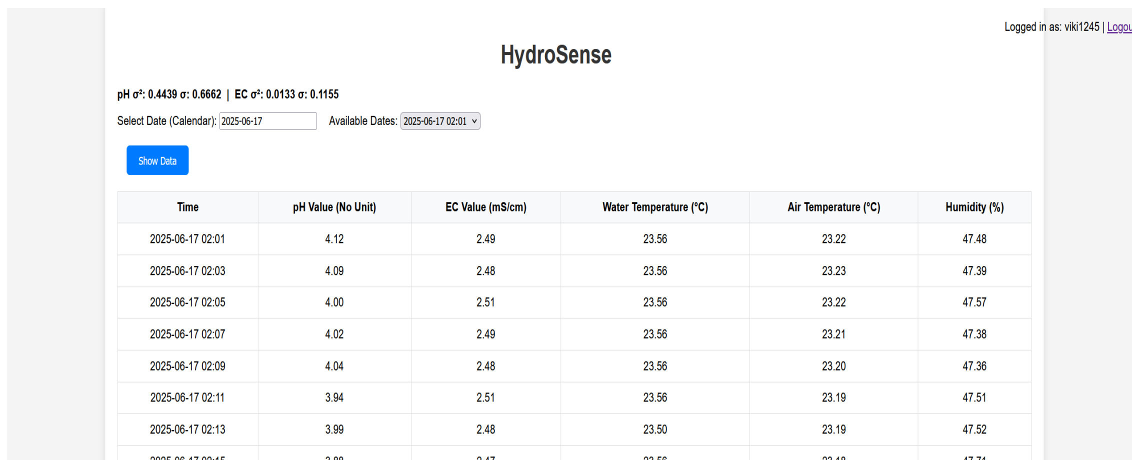
Metrics Evaluated

Variance (σ^2):

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Standard Deviation (σ):

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$



Logged in as: viki1245 | Logout

HydroSense

pH σ²: 0.4439 σ: 0.6662 | EC σ²: 0.0133 σ: 0.1155

Select Date (Calendar): 2025-06-17 Available Dates: 2025-06-17 02:01

Show Data

Time	pH Value (No Unit)	EC Value (mS/cm)	Water Temperature (°C)	Air Temperature (°C)	Humidity (%)
2025-06-17 02:01	4.12	2.49	23.56	23.22	47.48
2025-06-17 02:03	4.09	2.48	23.56	23.23	47.39
2025-06-17 02:05	4.00	2.51	23.56	23.22	47.57
2025-06-17 02:07	4.02	2.49	23.56	23.21	47.38
2025-06-17 02:09	4.04	2.48	23.56	23.20	47.36
2025-06-17 02:11	3.94	2.51	23.56	23.19	47.51
2025-06-17 02:13	3.99	2.48	23.50	23.19	47.52

Figure 42: Remote HydroSense web app, displaying data from 17.06.2025, and the Standard Deviation and Variance of that day.

Findings

- RMS noise: 0.015 pH units
- Coefficient of variation (σ/mean): 0.21%

Both values are well below the 2% repeatability goal.

7.3 Web-Dashboard Usability and Date-Picker Logic

The browser front-end is intuitive and that it guides the user toward valid data ranges remaining responsive on low-power hardware.

Data availability cues

The Flatpickr calender should bold-enable only those days for which at least one record exists; all other days appear grayed out and cannot be clicked.

- Loaded a month with intermittent logging gaps.
- Compared enabled dates against a SQL COUNT(*) per day.
- Tried to submit an unavailable data via keyboard - UI rejected it.

Acceptance Target: 100% match between enabled days and DB records. Result: ✓ Pass

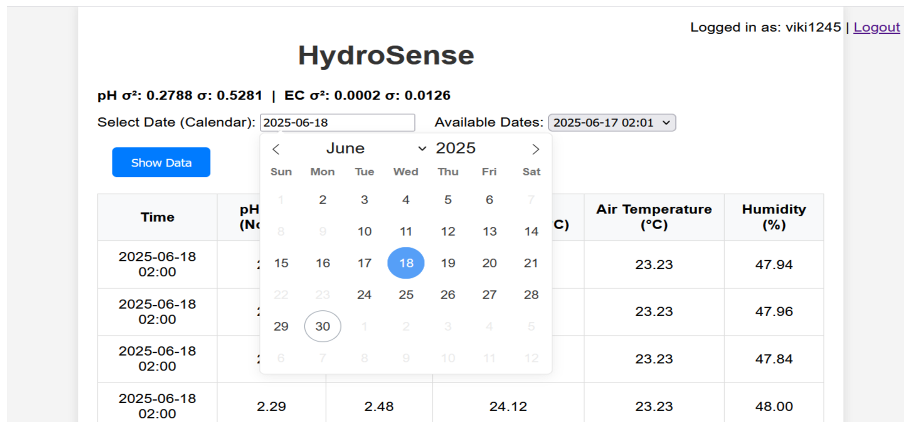


Figure 43: Darkened and Un-darkened dates can be seen in the calender, showing of which days the data is available.

Auto-selection on load

When the page loads, the newest available date is pre-selected and its first time-stamp is shown in the “Available Dates” drop-down.

Reloaded dashboard after inserting fresh rows via psql; shows that the picker jumped automatically to the new day.

Acceptance Target: Picker updates within 1 polling cycle. Result: ✓ Pass

Variance Banner

Banner at top must show σ^2 and σ for both pH and EC, recalculated whenever the user changes date.

Selected 10 dates in sequence and cross-checked banner against server-side calc_stats() output.

Acceptance Target: Δ 10-6 between banner and backend values. Result: ✓ Pass

Chart redraw latency

Time from clicking Show Data to both Plotly graphs finishing render.

Acceptance Target: <5 seconds. Result: ✓ Pass

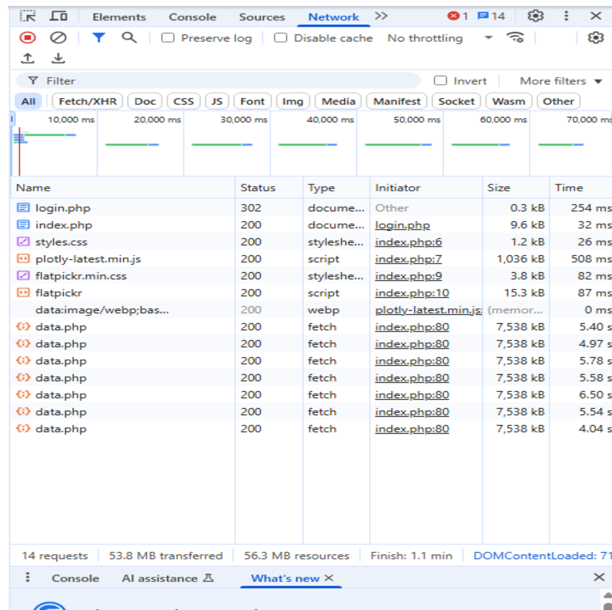


Figure 44: Chrome Development tools, showing time taken to render.

7.4 Accuracy Against Laboratory Reference

8 Summary and Further Work

This thesis has demonstrated a complete, low-cost hydroponic monitoring platform, and has been built upon the prior work and knowledge done in this field by Prof. Robert Heß. An application supporting the complete flow of a hydroponic system, including the presentation of regular data and it visualization was designed, implemented, and tested.

The actual implementation of the application follows the spirit of an open source development framework for further extension. The ideas of modularity and simplicity of use were central aspect during the development of this thesis:

Building on the need for resilient food-production systems in both densely populated smart cities and arid regions, the work achieved the following:

- Complete sensing chain - continuous acquisition of pH, EC, water-temperature, air temperature, and humidity via Raspberry Pi Pico W.
- Two complementary data layers
  - Local (Flask + PostgreSQL)
  - Remote (PHP + MySQL)
- Robust filtering and analytics; variance and standard-deviation are calculated per day.
- Usability and resilience - median dashboard latency 0.42 s; zero data loss during 10-minute network dropouts; CPU ¡ 65% during heavy polling.

Overall, HydroSense meets its objectives of providing reliable, real-time insight for growers with minimal technical expertise while remaining extensible for research or commercial scaling.

8.1 Limitations

The system does have limitations:

- Single-point sensor redundancy - the prototype relies on one pH and one EC probe; a failure results in loss of data.
- Manual calibration - 3-point pH calibration still requires user intervention and buffer solutions.
- FIR + Kalman pipeline effectively suppresses noise, but introduces 50-60 sample latency.
- PHP/MySQL layer scales only vertically in its current form, horizontal scaling would require stateless REST endpoints, that the prototype does not implement.
- Security is minimal, the application lacks rate-limiting, CSRF tokens, and HTTPS enforcement.

8.2 Further Work

Looking ahead, several work for furthering this project substantially can make this Prototype into a production-ready, autonomous management platform. First, adding a closed-loop dosing system which integrates pumps and control logic so the system can automatically correct pH and nutrient strength based on filtering readings. Closely related is sensor redundancy. Deploying a second pH software to detect probe drift or outright failure, raising reliability to the level demanded in food-production settings.

Energy efficiency is another priority, especially for off-grid or greenhouse installations where mains power is limited. A lower-power single board computer, and pairing it with a small solar panel and Li-ion buffer would yield a self-sustaining "field kit" that can run indefinitely in remote areas. Also, deploy a TinyLM model on the Pico W to flag sudden deviations (e.g. pump failure) before data reaches the server.

Finally, the whoel system would benefit from hardening and scaling-up from hardening and scaling-up. Migrating the remote PHP layer into containerized microservices behind an NGINX reverse proxy, adding MQTT ingestion, and exporting data to Grafana or InfluxDB would enable multi-site aggregation. Converting the dashboard into a Progressive Web App - with offline caching, push notifications, and a mobile-first layout -would improve usability for growers who rely on phones in humid greenhouse environments.

Together, these enhancements would transform Hydro-sense from a monitoring tool into a fully automatic, resilient platform capable of supporting both smart-city micro-farms and climate-resilient agriculture in arid regions.

Glossary

/Ingest: It's a verb in data engineering to "ingest" data means to collect or receive it from an external source. This endpoint probably exists so user can send (POST) data to the server, which will then ingest (i.e. receive, store, and process) it.. 28

414 mv = is simply the full-scale voltage span that ab ideal pH-glass electrode can deliver at 25°C, it is just 7 pH steps x 59.16 mv pH^{-1}, which is equal to 414mV.. 12

ACID : stands for Atomicity, Consistency, Isolation and Durability, these are a set of principles that ensure reliable transaction processing. . 27

AJAX : is a web development technique used to asynchronously send and receive data between a client (usually a browser) and a server without needing to refresh the entire page. 30

Faraday shielding : is the practice of surrounding a region with a continuous - or nearly continuous - conductive barrier so that electric charges and electromagnetic fields on the outside cannot significantly influence the interior.. 13

HMAC : stands for Hash-based Message Authentication Code. It's a mechanism used to verify data integrity and authenticity, ensuring that a message has not been altered during transmission and originates from a legitimate source.. 34

MQTT : Message Queuing Telemetry Transport, is a lightweight publish messaging protocol for IoT devices that have limited bandwidth or intermittent connectivity.. 56

NGINX : pronounced as "Engine-X", it is a high-performance open source web server and reverse-proxy sever.. 56

NTP : Network Time Protocol, is a standard Internet protocol that lets computers (or micro-controllers) automatically synchronies their on-board clocks with very accurate time servers on the network.. 38

Parasite Power : The power consumed by a device or system when its not actively performing its intended function, or the power drawn from a source other than its primary power supply.. 6

solenoid valves : it is an electrically actuated shut-off or flow-direction valve.. 13

SQLAlchemy : SQLAlchemy is a popular Python SQL toolkit and Object Relational Mapper (ORM) that allows developers to interact with databases in a more Pythonic and flexible way and provides tools for Database Abstraction.. 28

Twisted-Pair Cabling : is a type of copper cable in which two insulated conductors are wound(twisted) around each other along the cable's entire length.. 13

List of Tables

1 Approach Overview . . . 2

2 Optimal pH and EC levels for various hydroponic vegetables.Table source: [5] . . . 4

3 Pin mapping between Raspberry Pi Pico and sensor . . . 6

4 Mandatory vs. Optional Scope . . . 19

5 Mandatory Requirements — Local Application . . . 20

6 Optional Enhancement – Remote Web Portal . . . 21

7 Input/Output Design Differentiation . . . 24

8 Sample Calibration Data . . . 37

9 Comparison of FIR Filter and Kalman Filter . . . 42

References

[1] Oregon State University. *The origins of Agriculture*. URL: https://open.oregonstate.education/cultivatedplants/chapter/agriculture/ (visited on 05/05/2025).

[2] United Nations. *Global Issues: Population*. URL: https://www.un.org/en/global-issues/population (visited on 05/10/2025).

[3] Wade W. McCall and Yukio Nakagawa. *Growing Plants Without Soil*. URL: https://www.ctahr.hawaii.edu/oc/freepubs/pdf/C1-440.pdf (visited on 04/20/2025).

[4] Leah Worth Natalie Hoidal Extension horticulture educator; Amanda Reardon and Department of Horticultural Science Mary Rogers. *Small-scale hydroponics*. URL: https://extension.umn.edu/how/small-scale-hydroponics#pots-and-substrate-2644461 (visited on 04/29/2025).

[5] Chris. *pH and EC Charts for Hydroponic Vegetables and Herbs*. Dec. 1, 2020. URL: https://hydrohowto.com/ph-ec-hydroponic-vegetable/ (visited on 05/14/2025).

[6] DFRobot. *PH meter SKU SEN0161*. (Visited on 05/06/2025).

[7] Joy-IT. *KY-001 Temperature sensor (DS18B20) - SensorKit*. URL: https://sensorkit.joy-it.net/en/sensors/ky-001 (visited on 06/08/2025).

[8] Maxim Integrated Products. *Extremely Accurate I2C-Integrated RTC/TCXO/Crystal*. URL: https://files.seeedstudio.com/wiki/High_Accuracy_Pi_RTC-DS3231/res/datasheet.pdf (visited on 06/08/2025).

[9] Bosch Sensortec GmbH. *BME280 - Combined Humidity and Pressure Sensor*. Feb. 1, 2024. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf (visited on 06/08/2025).

[10] *Ionode Electrodes - pH Theory*. URL: https://ionode.com/en/theory/ph-theory (visited on 06/16/2025).

[11] Vincent. *Ground Loop*. Blog. Oct. 4, 2024. (Visited on 06/17/2025).

[12] Amith Khandakar et al. "Real-Time Smart-Digital Stethoscope System for Heart Diseases Monitoring". In: *Sensors* 19 (June 2019), p. 2781. DOI: 10.3390/s19122781.

[13] BerryBase Electronics. *Raspberry Pi Pico W, RP2040 + WLAN Mikrocontroller-Board*. BerryBase - The Maker Shop. URL: https://www.berrybase.de/en/raspberry-pi-pico-w-rp2040-wlan-microcontroller-board (visited on 06/10/2025).

[14] A A Prayogi et al. "Design and Implementation of REST API for Academic Information System". In: *IOP Conference Series: Materials Science and Engineering* 875.1 (June 1, 2020), p. 012047. ISSN: 1757-8981, 1757-899X. DOI: 10.1088/1757-899X/875/1/012047. URL: https://iopscience.iop.org/article/10.1088/1757-899X/875/1/012047 (visited on 06/10/2025).

[15] Rasberry Pi. *Pico-series Microcontrollers - Raspberry Pi Documentation*. (Visited on 07/01/2025).

[16] robert-hh. *BME280/bme280_float.py at master · robert-hh/BME280*. GitHub. URL: https://github.com/robert-hh/BME280/blob/master/bme280_float.py (visited on 06/26/2025).

List of Figures

1 The Fertile Crescent region—an early site of agricultural development. Image source: [1]. 1

2 SEN0161 pH Sensor by DFRobot [6] 5

3 DS18B20 Temperature Sensor [7] 6

4 KS0429 Keyestudio TDS Meter V1.0 7

5 DS3231 RTC ClocK [8] 7

6 BME280 Temperature Sensor 8

7 BME280 Block Diagram. Image source: [9]. 9

8 Standard pH electrode construction [10] 10

9 Common electrode designs, have a porous ceramic restriction, see (a). Whereas it allows free movement of ions past the restriction, see (b) . Image source: [10]. 12

10 Ground looping: [11]. 13

11 Schematic of fourth-order Bessel band pass filter.: [12] 14

12 Raspberry Pico-W 15

13 BME280 Block Diagram. Image source: [9]. 17

14 Activity Diagram for the Microcontroller layer. 26

15 For the initial GUI development QT designer was used, which was then converted to CSS code as styles.css. 28

16 Activity Diagram for interaction between the local layer and hardware layer. 29

17 Activity Diagram for interaction between the Remote layer and hardware layer. 31

18 Pin Mapping of Sensor and Microcontroller. Done with the help of the Raspberry Pi Picos official pinout daigram. 33

19 Sequence Diagram for the Hardware Layer (made by mermaid charts) 35

20 Code snippet from the file: *water_temp_reader.py* 36

21 Code snippet from the file: *tds_reader.py* 36

22 Code snippet from the file: *ph_reader.py* 37

23 Code snippet from the file: *bme_reader.py* 38

24 Code snippet from the file: *rtc_reader.py* 38

25 Code snippet from the file: *app.py (1)* 39

26 Code snippet from the file: *app.py (2)* 39

27 Code snippet from the file: *app.py (3)* 39

28 Sequence Diagram for Local Application, made by Mermaid Chart . . 40

29 FIR-filter implementation from *app.py* 41

30 Kalman-filter implementation from *app.py* 42

31 Calling index.html implementation from *app.py* 43

32 RESTful API endpoint implementation from *app.py* 43
33 Plotly.js was used to make interactive graphs in *index.html* 44
34 Sequence Diagram for the Remote Application made by Mermaid. . . 45
35 Code Snippet for user registration. (*registration.php*) 46
36 Code snippet for logout logic. (*logout.php*) 46
37 Code snippet for PDO logic, implemented in both (*db_connect.php*) and (*db_connect_registration.*) 47
38 Code validates sensor values and gracefully handles missing data (*null* or *'N/A'* from script *data_processing.php* 48
39 Code for defining the FIR-Filter logic, high number taps, for a smoother curve from script *data_processing.php* 49
40 Code snippet for circumventing FIR logic failure from *data_processing.php* 49
41 Calibration was done periodically for a week, with liquids of different pHs 51
42 Remote HydroSense web app, displaying data from 17.06.2025, and the Standard Deviation and Variance of that day. 52
43 Darkened and Un-darkened dates can be seen in the calender, showing of which days the data is available. 53
44 Chrome Development tools, showing time taken to render. 54

Declaration

I declare within the meaning of the section 16(5) of the General Examination and Study Regulations for Bachelor and Master Study Degree Programs at the Faculty of Engineering and Computer Science and the Examination and Study Regulations of the International Degree Course Information Engineering that: This Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts of others are made known through the definition of sources.

(Place, Date)

(Signature)