**Introduction**

Through the development of a temperature- and alarm-system. A range of aspects within the IoT environment is explored. Temperature measurements are to be sent to a RPI, where a alarm system reacts to the incoming live temperature readings, the alarm system includes an embedded interface for the RPI for operators through LEDS and buttons connected to the GPIO of the RPI. The data produced from the alarm system is then forwarded to a database as well as a cloud based data visualization service. The data is subsequently analyzed.

Developing a alarm system and establishing the necessary data flow explores the potential of several IoT technologies. The RPI sits at the core, through the GPIO’s of the RPI, the possibilities of sensor data collection and embedded interfacing is shown. The RPI is also the center for all of the data management, components like MQTT, databases and REST API is used on the RPI showing both the power of the RPI as well as the utility of these protocols.

The given system specification, asks for the RPI only to be used for data collection, with the server components residing on the PC. For a solutions that’s closer to production, and to apply more of the RPI’s potential. Both the server components, the database and the MQTT broker is running on the RPI. Additionally, an ESP32 is used as a ADC. An ESP32 is a microcontroller with built-in Wifi. This enables the temperature reading to be measured remotely, and becomes scalebable. Being much more aligned with the IoT mindset and showing more capabilities of protocols like MQTT.

**Methods**

**Hardware Overview:**

In setting up the temperature management system, several hardware components are needed. Both for development, but also for further production of the project. All of the hardware components are listed in Table #. The RPI hosts the MQTT broker, the relational database, the temperature handling, as well as the alarm handling. The ESP32 functions as an Analog to Digital converter, transmitting temperature readings to the RPI, either through the Serial protocol over USB, or through the MQTT broker. The remaining hardware components are quite simple and self explanatory in their table rows.

|  |  |
| --- | --- |
| **Hardware** | **Description** |
| Raspberry Pi 4 | Small Linux based computer with IO |
| ESP32 | Microcontroller with built in WiFi |
| Thermistor 1K | Analog temperature sensor |
| DS18B20 | Digital temperature sensor |
| Router | Wireless access for components |
| Resistors | 1k and 10k ohm |
| Jump Wires | For circuit connections |
| Breadboards | For Circuit development |
| Push Button | Operator interface for alarm system |
| LED | Operator interface for alarm system |
| PC | For project development |
|  |  |

**Software Overview:**

Creating an application with programs running on both a windows PC, a Raspberry Pi running Linux, and a ESP32 microcontroller requires an array of software. With Visual studio Code being at the core of it. Through the SSH extension for VS Code. Its possible to connect to the Raspberry Pi over the SSH protocol. This allows for development on the local computer, with the codebase residing on the Raspberry Pi. This eliminates the need for graphically interfacing the RPI, minimizing the strain on small computer. An additional extesnsion for VS Code is Platform IO, this functions the same way as Arduino IDE, development for the ESP32 module was done through this extension. For the MQTT broker, a broker called Mosquitto was used, this was ran on the Raspberry Pi and handled publish/subscribe. Postgres DB was the chosen relational database, it was initially intended to use Mongo DB locally on the RPI, but incompatible chip architecture prevented it.

|  |  |
| --- | --- |
| **Software** | **Description** |
| Visual Studio Code | IDE for programming |
| VS Code: SSH | Extension for remote work with VS code |
| VS Code: Platform IO | Extension for Microcontroller development |
| Mosquitto MQTT broker | Broker, handling the MQTT protocol |
| Postrgres SB | Relational Database |
| ThingSpeak | Cloudbased service for presenting data |
| Python | Programming language used in the RPI |
| C++ | Programming language used in microcontroller |
|  |  |

**Topology:**

For development, easy access to all of the programmable components was vital. Developing code directly on the RPI would make it cumbersome, as its straining on the RPI’s processor, and the need for ESP32 programming would require one to switch between developing environments. The SSH extension for Visual Studio Code was used, this allowed development on both the ESP32 and RPI to be performed at the same time. For accessing the RPI over SSH, a new network was established with a separate router. With the RPI connected to a router, the database- and MQTT server was made accessible, for the local computer and the ESP32. Additionally, internet access was required for the RPI, for the OS to be updated and specific python packages installed for the project. The RPI was therefore connected to the router over ethernet, while at the same time connected to a separate Wifi network with internet access.

A close-up of a person's face

Description automatically generated

**Temperature handling:**

The raspberry receives as mentioned temperature readings in three different ways. The DS18B20 reads the temperature and transmits it digitally through the 1-Wire protocol to the RPI. On the RPI the values can be found under “/sys/bus/w1/devices/”, the data is accessed through a python script, together with the OS library. The second way the RPI receives temperature readings is through a thermistor. The RPI is lacking Analog inputs on its GPIO, a ESP 32 is therefore used as an ADC. The microcontroller ESP32 enables temperature data to be transmitted through WiFi, Bluethoot, and as for this case, through Serial over USB. The ESP32 reads the analog voltage readings through the thermistor, scales the data, applies a Low pass filter, before transmitting it digitally over Serial. On the RPI a python script receives the data with the help of a Serial library on “/dev/ttyUSB0”.

The third and preferred way to transmit temperature data is over the MQTT protocol. The with the ESP32 connected to the router over Wifi, temperature data is published to a MQTT broker running on the RPI. On the ESP32, a C++ program is developed, that uses a Wifi library, as well as a MQTT library in order to be able to publish the data to the RPI. A python script running on the RPI subscribes to the same topic as the ESP32 is publishing on, this gives access to the live temperature data on the RPI for further processing.

Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


**Alarm Handling:**

The temperature data is to be utilized in a alarm system. The alarm system is built on Python and hosted on the RPI. When the temperature reaches above a certain threshold, the LED indicates an active alarm. When the temperature reaches below the same threshold, the LED gets dimmed, indicating an unacknowledged alarm. The alarm will then be able to acknowledge by pressing the button. This turns off the LED. Both of the schematics in Figure # are connected to a small breadboard and connected to the RPI.

Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


**Data Handling:**

The temperature readings are to be stored in a database together with the states of the alarm system. A Postgres Database server is set up on the RPI. On the database a single table is set up, the temperature process values, the alarm set-point and the alarm state is stored. Running in parallel on the RPI is a MQTT broker. The Open Source version Mosquitto is used as the MQTT broker, configured to run without any credentials on the local ip address of the RPI for easy access.

Transmitting the data from the ESP32 is as mentioned above, published through a Wifi connection over the MQTT protocol. The python script running on the RPI subrscribing to the temperature data is inserting the live temperature data into the Postgres table.

**Data visualization and Analysis:**

The data produced from the temperature values and alarm system. Is visualized through the cloud platform Thingspeak and analyzed with the help of the Pandas library on python. The python script running on the RPI that’s subscribing to the temperature data over MQTT is also sending the temperature values to Thingspeak. This is done with the help of the Requests library, the REST API protocol is used to send a HTTPS put command, sending the data securely to the cloud platform. The temperature data is then plotted in Line graphs.

For data analysis, a separate python script running locally on the PC is used. The python script queries the temperature data from the database running on the RPI. The temperature data is then structured into a Pandas Dataframe. Making it easy to utilize the Analyse command for pandas dataframes, and for plotting, both for temperature value line plotting, as well bar graphs for the analysis results.

**Results:**

**Temperature readings:**

For temperature data, there’s two ways of measuring the data, and three ways of transmitting the data. Looking first at the two different ways of measuring. Both the thermistor and the DS18B20 are satisfactory, they showed a temperature not far from each other around room temperature, and had a steady realistic increase when heated up by pressing them between two finger.

Without having a control value to compare them to, its hard to make any assumptions of which one where the most accurate, but with the thermistor temperature being based on the chosen electrical components and the measure voltage scaling in the code. Its reasonable to assume that the digital temperature sensor is more accurate, as well as more linear.

Although the digital temperature sensor appears more accurate, theres also the transmitting method which has to be taken into account. Using 1-wire is a fairly easy method for digital data transfer, but is limited in its scaleability. The same applies data transmition over Serial. A little bit mote hassle to set up the correct messaging encoding/decoding, to make it result in usable floats. But not very scaleable, Using MQTT on the other hand was with the MQTT broker already set up, the easiest way to transfer the temperature data, and by far the most scalable solution.

**Alarm Handling:**

With the purpose of the alarm handling system being to a way to easily interact with the RPI as well as showing off the customizable capabilities of the system