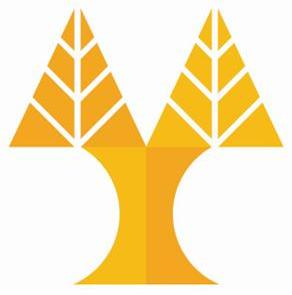
**University Of Cyprus**

**Computer Science Department**

**DRAFT**

**Photovoltaics Monitoring System using**

**IOT**



**MARIOS DIKAIAKOS**

Stefanos Ioannou

May 2019

**Summary**

Chapter **1** **Abstract**………………………………………………………………… **X**

1.1 Project Purpose

1.2 ENEDI Project Correlation

1.3 General Structure

Chapter **2** **On-site Gathering Node** ……………………………………………. **X** 2.1 Sensors

2.2 Arduino

2.3 XBEE Transmitter

2.4 Battery

Chapter **3** **Router Node** ……………………………………………. **X** 3.1 Raspberry Pi

3.2 XBEE Receiver

Chapter **4** **Prometheus** ………………………………………….. **X**

4.1 Introduction

4.2 Hierarchy

4.3 Configuration

Chapter **5** **Virtual Machine** ………………………………………….. **X**

5.1 Grafana

5.2 Database

Chapter **6 Large Scale Monitoring** …………………………….… **X**

7.1 Networking Algorithm

7.2 Apparatus

Chapter **7** **Architecture** **Justification & Financial Analysis** …………….… **X**

6.1 Architecture Justification

6.2 Financial Analysis

**References** .……………………………………………………………………... **X**

**Appendix** ……………………….……..…………………………….………… **X**

**Chapter 1**

**Abstract**

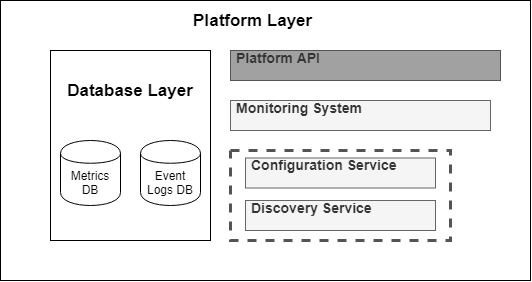
1.1 ***Purpose of Project***

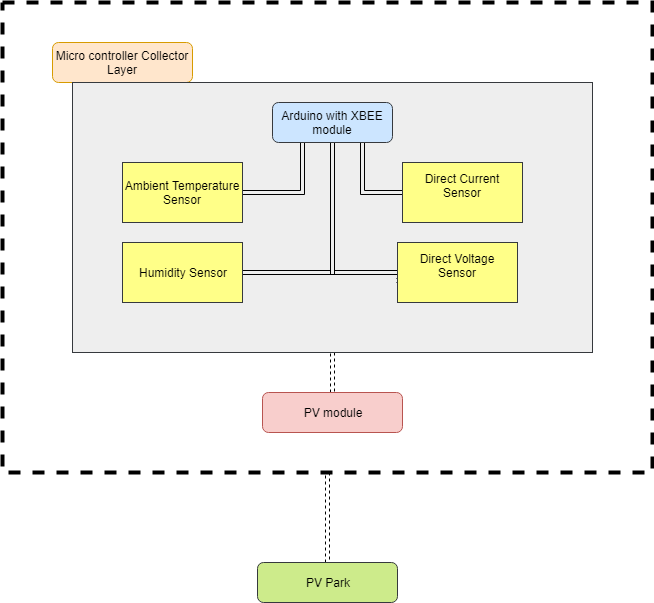
Monitoring a photovoltaic system is considered a big challenge nowadays as the solar PV energy generation is growing. The increasing cost of thermal power generation has resulted in a shift to renewables. Many research centers as well as firms seek to possess a powerful monitoring structure to check the generated electricity from the solar modules in order to compute expenses and return-on-investments. Furthermore by maintaining an observation on performance metrics on various solar panel modules we can build an efficient PV module to increase the efficiency on solar power collection. Another use of this structure is to monitor home solar panels for personal usage.

1.2 ***ENEDI Project Correlation***

The ENEDI correlation of this project is to build a fully functional cheap monitoring structure using microcontrollers and sensors that can gather the proposed metrics the engineers that work at ENEDI need. In other words the proposed system is an alternative to the one that is used in ENEDI. The monitoring system designed in ENEDI project is quite expensive because the machines that are used have low observational error (error margins of sensors in Enedi found at Enedi P.3.3a ,page 33) and the Data Loggers are scientific expensive components. Particularly the Campbell Scientific CR3000 costs up to 3000$.(radwell.co.uk online shop). The modules that are used are DC Power & DC current & DC voltage Sensors, PV temperature sensor, Wind Speed Sensor, DataLogger, Ambient Temperature & Humidity Sensor, HTTP module, Global Pane of Array Irradiance (Enedi P.3.3a ,page 34). Building a cheap monitoring structure may help reduce the expenditure of the project. Moreover the microcontrollers that are embedded to this proposed system can even provide more abilities to the researchers when measuring.

1.3 ***High Level Reference System Architecture***





The structure is built upon many levels. The high level architecture is the following:

* The first level is the on-site sensor gathering node.
* The second level is the router node which is heavily based on Raspberry Pi.
* The third level is the cloud storage which is based on the gateway application Prometheus which rests on a Virtual Machine.
* The fourth level is a front end web application which is used for visualizing data using graph charts.

Figure X: Multi-Tier architecture of the proposed monitoring system

Upon the next chapters there will be an excessive discussion on the implementation of each level followed by a justification of this model and a financial analysis.

PV Technology & Characteristics

The power output of a solar panel depends on the intensity of solar radiation, the amount of aggregated area of solar cells and the solar cell power efficiency of the modules.

[page 8 , F. Kong and X. Liu]

Solar cell power efficiency refers as the amount of energy in the form of sunlight that can be converted via PV modules into direct current. The proposed system must be able to provide all these parameters in real time to the observer. Other environmental factors such as humidity and temperature are necessary for measuring the performance of the modules. For example power output degrades when ambient temperature is getting higher, by a rate of 0.50% for 1 Celcius. ”According to estimation for every degree rise in temperature, efficiency of PV module decreases 0.5 percent”.(Improvement in Solar Panel Efficiency paper).

Pyranometers are very expensive in the market. A pyramoterer that can be integrated on a microcontroller can cost up to []. The cost of each collector node that is on a PV module must be low. For cost reduction solar irradiance can be measured using only direct current sensor. Estimation of solar radiation [W/m2] is feasible by calculating power from the solar cell and dividing by its cell area.

From this equation solar area is already know by measuring the length and width of the solar panel. Power [Watts] is the product of the maximum voltage [V] output of solar cell times the electric current [Amps].

Therefore by have a voltage sensor and a current sensor with simple calculations we can

**Chapter 2**

**Collector Node**

Arduino

Arduino Uno is a microcontroller board based on the ATMEGA328 next generation microchip. ATMEGA328 is based on an advanced RISC architecture with 32x8 general purpose working registers. The Arduino clock rate is up to 16MHz resonator with a crystal oscillator for dealing with time issues. Inside Arduino there is a self programmable flash program memory where developers can upload a script that will keep executing without any human intervention. Also it is capable of working between temperatures of - 40°C to 85°C. At 25°C with 1.8 V as operating voltage, its consumption is 45 mA, but by using power saving techniques the consumption can fall. With all these diverse features Arduino is definitely a neat embedded computer platform choice for monitoring purpose projects. The internet community has published thousands of projects that can be done using arduino.

XBEE module

|  |  |  |
| --- | --- | --- |
| Module | Model | Range |
| XBee 1mw Trace Antenna | Trace Antenna- Series 1 | 300 ft |

XBee 1mw Trace Antenna is a radio module that communicates using the protocol:

*802.15.4 / ZigBee standard protocol*. The main mechanism of the module is to perform frequency modulation. The power output of XBEE is about 1 mW and outdoor range is up to 300 ft. RF data rate can be approached to 250 Kbps. Today XBEE is the de facto solution for internet of things for point to point communication and multi-point networks. XBEE module however needs a board on top of an

|  |  |  |
| --- | --- | --- |
| Module | Model | Range |
| XBEE explorer regulated board | Trace Antenna- Series 1 | 300 ft |

The board is populated with 3.3V regulator and carries a FT231X microchip which offers a compact bridge to full handshakes UAT interfaces. XBEE modules use UART to communicate with USB. The digital port takes the values and then exposes them to a serial port which is used for communication between the Arduino and the XBEE.

Sensors

Sensor List

|  |  |  |  |
| --- | --- | --- | --- |
| Sensor | Model | Metric Type | Accuracy |
| Ambient Temperature | DHT22 | Celcius (T) | ±2℃ |
| Relative humidity | DHT22 | RH (Percentage) | ±5％RH |
| Direct Current | ACS712 | Amperes (A) | 1.5% at TA = 25°C |
| Direct Voltage |  |  |  |

[DHT11 Technical Sheet, ACS712 technical sheet ,

For monitoring temperature, DHT11 is selected as it has a very high humidity and temperature measuring accuracy. ACS712 All the sensors listed above demand power supply of about 3-5.5V.

DC and is composed of 4 pins. Measurement range for relative humidity is between 0-100% and -40C to 125C for ambient temperature. It uses an 8-bit chip for controlling a polymer humidity capacitor and DA18B20 is used for detecting temperature. Furthermore for monitoring electric current ACS712 is a fully integrated, low resistance current sensor. The main mechanism of this sensor relies on hall effect - based linear with 2.1 kV RMS isolation. An average sensing period is 2 seconds with full interchangeability. In the market is typically addressed as a very good economical solution for DC sensing.

Below is the apparatus on a breadboard that it was used for extracting the selected parameters from the sensors:

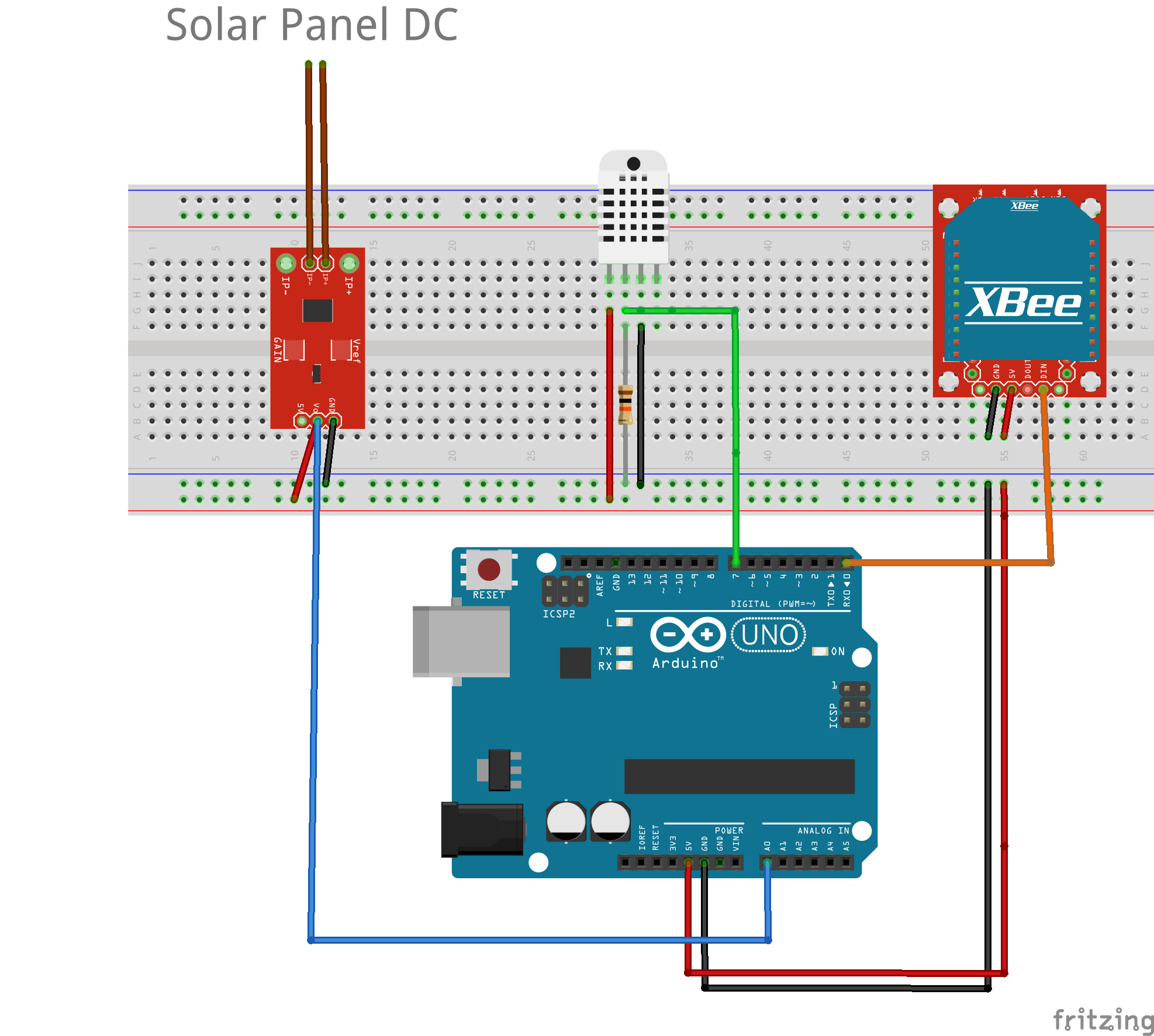


Figure 1.2

Proposed Microcontroller Collector

XBEE Configuration:

Digi provides a platform application to enable developers to interact with XBEE modules, the XCTU. For the XBEE transmitter we must first set the channel and the coordinator ID must be identical to the receivers as well as the Baud Rate. In this project the configuration the corresponding ID is 1001 ,the Baud Rate is 9600 bps and channel is set to C.

Figure X

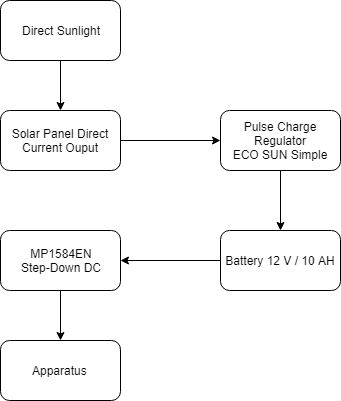
Arduino has about 28 pins PDIP, 14 digital I/O pins of which 6 provide PWM output The apparatus only takes advantage of pin 7 and A0 for input temperature and input current. Pin 7 is connected with the second data pin of DHT22 which requires a usual 4.7K pull up resistor.

Pin A0 is an analog pin and is connected with the pin analog in of ACS712. The pin can read signal from an analog sensor (ACS712) and convert it into a digital value for the microprocessor.Pin 1 TX which is used for serial receiving from the XBEE. When operating, the TX led flashes with baud speed while sending the serial data and RX led flashes during the receiving process. The other two pins that are shown in the diagram are used for voltage and ground. Through breadboard as a constructor base, sensors operate at 5V. Each sensor must be connected to the ground and the voltage.

## 

Power Regulation & Battery

A 12V Battery supplies energy to the apparatus. A typical 12V battery has a capacity about 7 Ah. That means if we draw 7A with 12V , the battery will last an hour. Arduino needs about 50mA for normal operation. For making sure arduino will draw correct voltage, a high frequency step down generator must be inserted in the apparatus. By forcing the current to pass form two input and output wires, ground the voltage from 12V can be safely converted to 5V with the module MP1584.

Another adjustable potentiometer to use is the ECO SUN simple pulse charge regulator 24V to 12V. This is the main regulator that connects the solar panel to the battery. Direct current from the solar panel modules pass through the regulator and is stored inside the battery. The flow of electricity follows the order as shown in figure 1.3. Every component in the gather node has different voltage demands. This complex regulation might be unpleasant but necessary. 

With this model there are multiple possible scenarios to be consider of. For instance direct sunlight might be deficient as a result of an overcast. Suppose that battery is fully charged, contains up to 10A. To calculate the sustainability of the node without direct sunlight ,the formula of output power is needed:

A battery 12V 10A ensures a power Figure X output of 120 Watts/Hour. Arduino draws 0.05A at working stage as mentioned before. Therefore arduino can last on the particular battery for about 480 hours or about 20 days. An observer has to mention the failure of the component or the overcast and preserve the power supply to the apparatus. The system furthermore will commit frequent checks and monitor the supply for the credibility of the power system using other sensors. In conclusion the theoretical span for the node to fully work for 20 days after any termination of green power supply.

**Chapter 3**

**Router Node**

Raspberry Pi

Raspberry Pi 3 is a single board computer integrated with the operating system Raspbian. It uses the processor Broadcom BCM2837B0 SoC with a 1.4 GHz 64-bit quad-core [ARM Cortex-A53](https://en.wikipedia.org/wiki/ARM_Cortex-A53). of storage. It supports ethernet up to a gigabit (1000Mbps, 1000Base-T) and the radio supports 802.11ac WiFi networks running on the 2.4GHz and 5GHz frequency bands, Bluetooth 4.2, and Bluetooth Low Energy (BLE) connections. Raspberry Pi is selected as the core module of a router node because it demands much greater energy than Arduino. Because of the minuscule energy consumption of Arduino on a typical working stage, every end node will be feeded from a 12V 22Ah battery outside in the solar park but raspberries are required to have a standard alternating-current (AC) electric power supply.

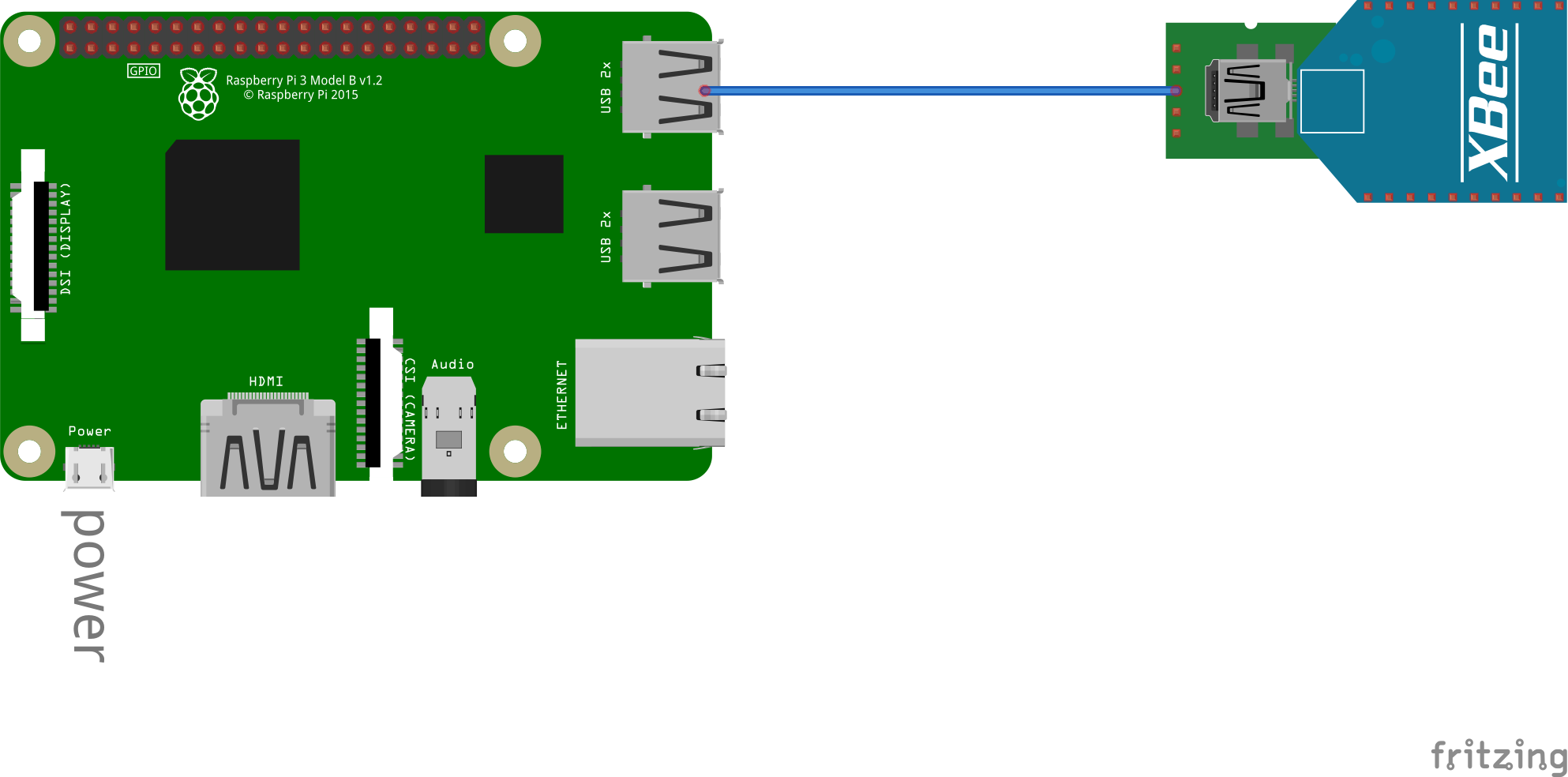
Setting the Raspberry Pi

First the RP must be configured with Raspbian. The installation of Raspbian Image must be done on the SD card without the module (headless install). After this a configuration for the Secure Shell is vital for connecting to the Raspberry pi later on. Enabling SSH is done by placing a file ‘ssh’ onto the boot partition of the SD card.

Structure of Router Node

The XBEE receiver is connected to a XBEE USB board. Thus the router node has a simple implementation. The raspberry Pi is connected only to a power cable from a standard wall power and with the USB board from the USB port. The USB board is to provide the feature of USB to serial base unit for the Digi XBee line . It’s main embed component is an FT231X USB-to-Serial converter chip.

Figure Router Node



Python 2.7 & Libraries

TTYUSB0 is the abbreviation for the 8-N-1 concept. The reading of bits happens every eight bits and one stop bit. The baud rate is 9600 bps. In the python library Serial (dev/ttyUSB0) interprets to open a port at 9600 bps and follow the 8-N-1 notation with no timeout.

To open a serial port and fetch values using python we must first call the serial library and set up a listener. The philosophy is to implement a forever loop where a readline command will keep reading data from the serial port where the XBEE is connected. By calling the function serial.Serial() and setting as a parameter the string (‘dev/ttyUSB0’), the listener is commanded to follow the 8-N-1 concept. The conditional forever loop is composed using a while(true) and curries inside a serial readline that gathers characters until a ‘\n’ (newline ASCII) occurs. The script Collector Python in the appendix illustrates the this scenario.

Using python 2.7 ,the installation of prometheus\_client package using PyPi needed first. The pip install prometheus\_client is executed on the raspberry Pi. When instrumenting ,there must be a web server exposing the registry. The prometheus registry is a set of collectors. These two steps where implemented in order for the prometheus to scrape from python script:

1)Using prometheus python library a registry is created with one gauge. (Appendix)

2)Create a simple web server and expose the registry for the Prometheus to scrape it

For the second step Pyramid web framework was used. Pyramid is a Python framework under the BSD-like license.

Software Accessing Raspberry Pi & Port Forwarding

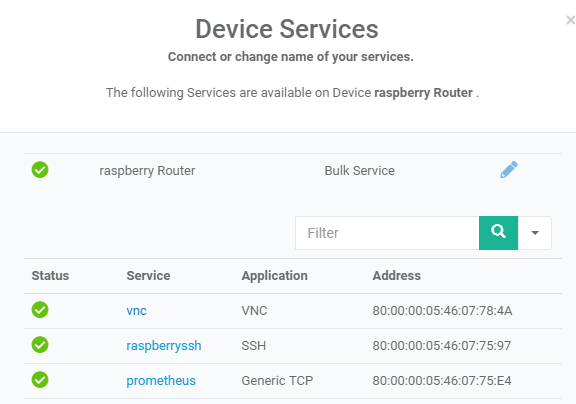
Communication with Raspberry Pi can be done via SSH. PUTTY is an excellent SSH and telnet client that provides a graphical user interface for setting any option of the SHH. Hostname is the local ip address of the Raspberry Pi, which is dynamic if not configure static via the dhcp configuration file. The port used for SSH is 22. Another program used was VNC connect which offers a secure ready to use remote access with graphical desktop sharing.

A problem occurs with the exposition of the IP address of a microcontroller.

All the router nodes which are consisted of Raspberry Pis are connected to a routing network device for getting a connection over the internet. In order to access any raspberry pi , port forwarding must be done. Port forwarding can become complicated when there are many instances we want to access with specific ports inside our network. For secure port forwarding Remote.it services were used. Remote.it assigns a different ip address to the module using secure proxies. To install Remote.it services first the weavedconnectd package is needed. After that by calling weavedconnectd service a menu appears where a user must enter credentials and configure the device several services.

For a router node the following services were configured:

SSH , generic TCP for prometheus and VNC for remote access. An example of new address for SSH application was proxy51.rt3.io : 30805. Thus for connecting with ssh the command changed to *ssh -l pi proxy51.rt3.io -p 30805.*



XBEE configuration & Message Structure & Collector Module

XBEE receiver is setup as a coordinator with Baud Rate of 9600 and ID similar to the end points. Every router is receiving and the expose data from XBEEs. Now the critical part is that the gather node must send data in a sophisticated manner. The smaller the message it is ,the better for less delay. A possible tuple that can represent an XBEE message can be indicated as:

.

For example the message 13T0 means that the apparatus gathered from the node with ID 0 the temperature 13 °C. This means that a single message is consisted of 1 integer and 2 characters. That is about 6 bytes that propagate from the XBEEs. Each node has a unique ID. Each identity is a predefined known signed integer. The three metrics abbreviations are the follow: T for temperature, H for relative humidity and C for current. The values are integers because there is no interest for precision metrics due to the nature of the problem. Precisions metrics such as floats or doubles need more memory and more computational power. To sum up, each row represents a message:

|  |  |  |
| --- | --- | --- |
| Value | Metric | Identification |
| Integer | C | Short Integer |
| Integer | H | Short Integer |
| Integer | T | Short Integer |

In python, for decomposing the message the regular expression (lib/py.re) script is used. For example *re.search(r'(\d\*)'+\_CURRENT,incoming,re.I)* finds the value of current inside an incoming xbee message. After collecting the three parameters a request is initiated to the local server of each raspberry, localhost:8000/store\_metrics (Pyramid web framework). All this work is done inside the collector module (collector.py). Synopsis of the collector module:

1. Open the serial port following 8-N-1 concept.
2. Set up a listener for capturing the incoming XBEE message.
3. Decompose the XBEE message to metric, value and ID of the gathered node.
4. Create & send a request to the local server with the above parameters.

Local Pyramid Server

The local python Pyramid server is under the URL localhost:8000. For starting the server a special bash script ‘server.sh’ calls the function *serve()* inside the server.py. Before starting the server a configuration is needed. A configuration is a special class configurator that has many properties. For the current implementation each server has two accessible paths: /store\_metrics and /expose\_ metrics. By adding a route to them, each time a request appears, a special function that is configured to that route initiates. Routes are added manually on the properties of the configurator class. The *store\_metrics()* function that routes to store\_metrics path has to safely store the request parameters of the collector module in a database. First it gets the parameters of the request body and then another mysql behaviour module creates an insert query. The *expose\_metrics()* function that routes to expose\_metrics path has to fetch the most fresh record of each ID with the three metrics. For example if there are three gather nodes a response from the expose\_metrics function should encode nine most recent records, that is, three metrics (T, H, C) times three nodes with the latest timestamp.

MYSQL Database

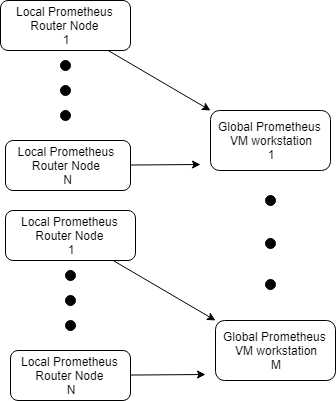
Every router node has a simple mysql database installed and rests on localhost. The database is managed by the database handling module. This module is the combination of an insertion & selection & deletion script along with a database connector. As mentioned before the collector module calls the database handling module for storing data. When insertion happens, the database handling module adds a timestamp with the exact time of the insertion on the tuple. On the other hand, when exposing the values, the selection of the freshest tuples is done by the selector script using a select query. Delete module is a basic mysql event that triggers every 14 days and delete the metrics according to their timestamp.

**Chapter 4**

**Prometheus**

Prometheus is a state of the art open source monitoring system. It provides client libraries that allow easy instrumentation of many services. All the data that it gathers are stored as time-series. Each value that is collected, is followed by a timestamp. The client library offers a variety of metric types. Particularly since the monitor parameters are temperature, humidity and current, the choice is gauge, which is a metric that represents a numerical value that can arbitrary go up or down. A very important concept to discuss is the Prometheus endpoint. An endpoint is where the timestamp values are available for scraping. This is also called an instance. Many instances are a job in Prometheus terms. Each instance targets a host and port. Example of what is included in this project can be seen in the appendix.

Global Prometheus

The job of each router node has the name raspberry\_Pi\_router. As the name suggest, the target an instance of a Raspberry Pi that gathers the data from the XBEE. The instance cannot undergo any alteration because the target is the local pyramid web server. Inside the yml configuration file , the host and port are those of the pyramid web server and scraping is by default every 15 seconds. 

Hierarchical Prometheus

Prometheus offers a federation which allows a high level Prometheus to scrape values from another Prometheus server. In this case we label a Prometheus as global that collects aggregate data from lower local server. Prometheus that rests on the Virtual Machine is referred as global and the local Prometheus that rests on-site gather node. The configuration for the global Prometheus is the following as shown in figure Prometheus:

The honor\_labels enables us to rewrite any label exposed by the server. Federation job targets the router nodes. The current IP address as above was given from the LINC stuff.

Prometheus Instrumentation

When instrumenting we should be sure that is an internal part of code. The router node export everything in Python. Thus again the usage of Client libraries in python is essential. The python library allows to expose the metric to the local Prometheus via the http endpoint. Each time an instrumentation of a metric happens a new gauge class must be instantiated. A gauge class constructor takes the arguments of a the name of the gauge metric, its description and the registry to be assign. When implementing the collector, it is prohibited to use the usual direct instrumentation approach and each time set the metrics different value on each scrape. Instead each time we instantiate a new class gauge and add it to the registry. After the addition happens, the gauge is setted to the value from the exposition. This client-server model implementation makes the Prometheus to act as a scraper, not a pusher. Prometheus every time lapse collects data from a specific endpoint (/expose\_metrics). Another different implementation was to set a Prometheus as an endpoint and push json files from the local pyramid web server. Local Prometheus with only one scrape collects the metrics that the local pyramid web server expose with the expose\_metric handler.

**Chapter 4**

***Architecture Justification***

To implement these ideas and deliver a real working prototype, there must be enough justification of the hierarchy of this model. The main qualities that the proposed system must have, are derived from the architecture hierarchy. These qualities are sustainability and modularity.

Sustainability & Modularity

From top to bottom , each node must be interchangeable. This means that if a fault come across, it needs to be changed without dissipate a lot of time and money. Beginning from the metric gathering nodes, the apparatus must be inexpensive and efficient. Moving on, each router node must be again an inexpensive microcontroller that can handle many requests per second. Because of the absence of computations a virtual machine is able to withstand the storage phase. The virtual machine is basically the last level architecture. To be further precise sustainability is basically affect by a lot of factors:

Environmental Factor

To protect the apparatus from the environment, a shield for each node is compulsory. Water from storms, heat waves, wandering animals or even misdoers can damage the machines. A basic shield can be 120 \* 120 cm box made from wood and metal ,integrated with a locker.

Interchangeability

Each component that is found on the node can be cheaply purchased in case of any malfunction. The most costly component is the Raspberry Pi ,which costs about 50$. The time needed to change any component is very slight because all the sensors and modules are not embedded.

Electricity Drawing

As mentioned before, the power supply of raspberry Pi is about 2.5A and has a voltage of 5V. It takes roughly 80 hours of operation to peak the consumption to about 1kw. Assuming that there are 8765 hours in a year the consumption aggregation is 110 KW. The single domestic use tariff of AHK is about 9,46 cents. Therefore a router node will consume about 10,40 € per year. Arduino however have much lower consumption. 0.05A and voltage 5V it will take about 4000 hours to consume 1 KW. Therefore a node gathering node will consume about 2,20 € per year. A desktop computer uses about 45-250 W but most laptops about 15W-60W. Let’s suppose that a VM might theoretically use 20 W. It might need about 16,5 € per year.

Data Volume

Data that is generated costs to store. Sophistication on data collecting is shrinking the data to the point only to gather them without any unnecessary text characters, timestamps. All these unnecessary junk data must be dispatched. When gathering, a set of rules is applied to decrease data volume as mentioned before.

**Chapter 5**

**Virtual Machine**

Virtual Machine

Global Prometheus is on a VM. This provides modularity and makes the disassembly easy if needed. A remote machine has two frameworks installed: Grafana and Prometheus. Only these two softwares are needed.

Grafana

For data visualization , a leading open source software for time series analytics is Grafana. It combines a friendly graphical user inteface with a powerful dashboard for any database. Workflow integration includes authentication and a variety of themes to choose , in order to set home dashboard. It also provides a build in prometheus support with a query editor integraded with a metric name lookup a template queries for dashboard and discover pattern of series.

**Chapter 5**

**Large Scale Monitoring**

Scaling for XBEE communication

Suppose there are n Solar Panels inside a clean energy park. For every solar panel there is a demand for an onsite gather node, an end-point. There is a need for gathering from multiple endpoints , a router will handle. The number of routers/coordinators is m, where n>m. A big issue is to find the optimum number for which one coordinator will handle. We cannot send data from both endpoints simultaneously. Data must be timed so only one endpoint will transmit to a router. Timing two microcontrollers needs precision. The objective is to minimize errors ,reporting without delays failing modules and most importantly create a sustainable time schedule on the propagation of information so that all modules acquiesce with each other. A good configuration, a networking algorithm and an observer for reporting the failing modules are the keys for implementing the previous monitoring structure on large scale basis.

Scheduling Microcontrollers

For successful scheduling, a very important value to set is a timeout threshold. A timeout threshold defines the length of time where all the end nodes transmit their data to the selected

router. Therefore the threshold is mathematically defined as follows:

where x denotes each endpoint node delay that sends the information for the specific router and c is an add up idle time for spacing the propagations. Endpoint delay is consisted of processing delay, transmission delay, queuing delay and propagation delay. [MORE]

**APPENDIX**

UART: UART is a common protocol used for duplex serial communication.

IOT: Abbreviation for internet of things.

Baud Rate: Serial communication speed between modules

Serial Bus: Serial bus is consisted of a transmitter wire and the receiver

wire (RX & TX)

Solar Power

Efficiency: Solar power efficiency is a measurement of how much

electricity is produced.

PWM : Pulse Width Modulation

Port forwarding: Forwarding data to a node inside a local network with the address of gateway node and the port that much to the target node

**Code**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Arduino Script:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Pyramid Module**

from wsgiref.simple\_server import make\_server

from pyramid.config import Configurator

from pyramid.response import Response

from prometheus\_client import generate\_latest, CONTENT\_TYPE\_LATEST

from prometheus\_client import CollectorRegistry, Gauge

from pyramid.view import view\_config

import insert

import database

def fetch\_mysql():

conn = database.createConn()

cur = conn.cursor()

sql = "SELECT \* FROM WHERE "

cur.execute(sql)

def \_\_initRegistry():

registry = CollectorRegistry()

id,temperature,humidity,current = fetch\_mysql()

gT = Gauge("Temperature", "Temperature Of Solar Panel0", registry=registry)

gC = Gauge("Current0", "Current Of Solar Panel0", registry=registry)

gH = Gauge("Humidity0", "Temperature Of Solar Panel0", registry=registry)

return registry , gT,gC,gH

def prometheus\_values(t,c,h,ID):

strID = str(ID)

gT.set(t)

gC.set(c)

@view\_config(

route\_name = "expose"

)

def expose\_metrics(request):

registry = \_\_initRegistry()

return Response(generate\_latest(registry),

content\_type=CONTENT\_TYPE\_LATEST)

@view\_config(

route\_name = 'store'

)

def store\_metrics(request):

if 'metric' in request.params and 'value' in request.params and 'ID' in request.params:

print request.params['metric']

print request.params['value']

print request.params['ID']

metric = request.params['metric']

value = int(request.params['value'])

ID = int(request.params['ID'])

insert.insert(value,metric,ID)

return Response(body = "Stored to database",

content\_type=CONTENT\_TYPE\_LATEST)

else:

return Response(body = "Not stored to database")

def config():

config = Configurator()

config.add\_route('expose','/expose\_metrics')

config.add\_route('store','/store\_metrics')

config.scan()

app = config.make\_wsgi\_app()

server = make\_server('127.0.0.1', 8000, app)

return server

def serve():

config().serve\_forever()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Collector Python Module:**

import time

import datetime

import server

import request

import serial

import re

import time

from prometheus\_client import generate\_latest, CONTENT\_TYPE\_LATEST

from prometheus\_client import CollectorRegistry, Gauge

\_CURRENT = "C"

\_TEMPERATURE = "T"

\_HUMIDITY = "H"

# index = ID

# 0 = Temperature value

# 1 = Humidity value

# 2 = Current value

#Enable USB Communication

ser = serial.Serial('/dev/ttyUSB0', 9600,timeout=.5);

while True:

incoming = ser.readline().strip()

searchID =re.search(r'ID(\d\*)',incoming,re.I)

searchC = re.search(r'(\d\*)'+\_CURRENT,incoming,re.I)

searchT = re.search(r'(\d\*)'+\_TEMPERATURE,incoming,re.I)

searchH = re.search(r'(\d\*)'+\_HUMIDITY,incoming,re.I)

if searchID is not None:

ID = int(str(searchID.group(1)))

print incoming

if searchT is not None:

request.createRequest(int(str(searchT.group(1))),\_TEMPERATURE, ID)

elif searchH is not None:

request.createRequest(int(str(searchH.group(1))),\_HUMIDITY, ID)

elif searchC is not None:

request.createRequest(int(str(searchC.group(1))),\_CURRENT, ID)

time.sleep(0.5)

#Start the server with the prometheus values

updateTables.update\_\_(gatherNodes)

**(Collector Module Another script )**

import requests

def createRequest(value,metric,ID):

URL= "http://localhost:8000/store\_metrics"

\_PARAMS = {'value':value,'metric':metric,'ID':ID}

r = requests.get(url = URL, params = \_PARAMS)

print r

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Database Handler Module:**

**Insert Module**

import database

import time

import datetime

def insert(value, metric, id):

conn = database.createConn()

cur = conn.cursor()

ts = time.time()

timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')

if metric == 'C':

sql = "INSERT INTO current (ID,value,time) VALUES (%s, %s, %s)"

elif metric == 'H':

sql = "INSERT INTO humidity (ID,value,time) VALUES (%s, %s, %s)"

elif metric == 'T':

sql = "INSERT INTO temperature (ID,value,time) VALUES (%s, %s, %s)"

value = (id,value,timestamp)

cur.execute(sql,value)

conn.commit()

print(cur.rowcount, "record inserted.")

**Connector Module**

import mysql.connector

from mysql.connector import Error

from mysql.connector import errorcode

def createConn():

connection = mysql.connector.connect(host='localhost',

database='nodes',

user='root',

password='root')

return connection

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*