

TEMPERATURE MONITORING SYSTEM USING EDGEX

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

S.LAKSHMI NARAYANAN (37110402)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600 119**

MARCH - 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **S.LAKSHMI NARAYANAN (37110402)** who carried out the project entitled "**TEMPERTURE MONITORING SYSTEM USING EDGEX**" under my supervision from **November 2020** to **March 2021**.

Internal Guide

Dr.S.Vigneshwari, M.E., Ph.D.,

Head of the Department

Dr. L. Lakshmanan, M.E., Ph.D.,

Dr.S.Vigneshwari, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **S.LAKSHMI NARAYANAN (37110402)** hereby declare that the Project Report entitled “**TEMPERTURE MONITORING SYSTEM USING EDGEX**” is done by me under the guidance of **Dr.S.Vigneshwari, M.E., Ph.D.**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean, School of Computing, Dr. S. Vigneshwari, M.E., Ph.D., and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide, **Dr. S. Vigneshwari, M.E., Ph.D.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Temperature Monitoring System, as the name suggests is a Full-Stack IoT application that implements an automated approach towards monitoring temperature. The rise in temperature is a major concern to our ecosystem due to global warming and many man-made factors. We all are aware of the forest fires in the Amazon rainforest and the massive blast at Lebanon in 2020 and the cause behind them. The goal of this project is to minimize the chances of such massacres from happening.

The proposed project is “Temperature Monitoring System Using EdgeX” an Industrial Automation project which utilizes edge computing and IoT to analyze the data and act upon the situation to avoid such massacres. This project emphasizes more on how EdgeX supports and increases the scale of an IoT application/system. Using EdgeX the required temperature data is read from a sensor, the required actions on the data are performed and the data is pushed on to a database followed by a web server, and finally the data is then visualized through a user interface

TABLE OF CONTENTS

	ABSTRACT	v
	LIST OF FIGURES	ix
	LIST OF TABLES	xi
CHAPTER NO.	TITLE	PAGE NO.
1.	INTRODUCTION	
	1.1 INTRODUCTION	1
	1.2 MOTIVATION	2
2.	LITERATURE SURVEY	
	2.1 LITERATURE REVIEW	3
	2.2 PROPOSED SYSTEM.....	4
3.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	
	3.1 AIM	6
	3.2 OBJECTIV	
	3.3 LIMITATION AND SCOPE	7
	3.3.1 SCOPE	7
	3.3.2 LIMITATIONS	7
4.	SOFTWARE REQUIREMENTS AND METHODS	
	4.1 SOFTWARE REQUIREMENTS.....	8
	4.1.1 DOCKER.....	9
	4.1.2 ANDROID STUDIO	10
	4.1.3 RASPBIAN OS.....	10
	4.1.4 PYTHON 3	11
	4.1.5 FLASK.....	12

4.1.6	uWSGI	13
4.1.7	NGINX.....	14
4.2	HARDWARE REQUIREMENTS	15
4.2.1	RASPBERRY PI 3B+	16
4.2.2	SHT20 SENSOR.....	17
4.3	METHODOLOGY	18
4.3.1	IOT THE BACKBONE	19
4.3.1	PROJECT PLAN AND TIMELINE	19
5.	SYSTEM ARCHITECTURE AND ALGORITHM	
5.1	SYSTEM ARCHITECTURE.....	21
5.2	MODBUS.....	23
5.2.1	OVERVIEW.....	23
5.2.2	MODBUS MEMORY AREA	24
5.2.3	TYPES OF MODBUS.....	25
5.2.3.1	MODBUS RTU	25
5.2.3.1a	RTU MESSAGE FRAME.....	25
5.2.3.1b	RTU COMMUNICATION MODES	27
5.2.3.1 c	COMMUNICATION USING RS 485	27
5.2.3.2	MODBUS TCP	28
5.2.3.2a	TCP MESSAGE FRAME	28
5.2.4	MODBUS IN EDGEX	30
5.2.5	WORKING	30
5.3	PROJECT IMPLEMENTATION	32
6.	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS	
6.1	RESULT AND DISCUSSION.....	36
6.2	PERFORMANCE ANALYSIS	36

7.	SUMMARY AND CONCLUSIONS	
	7.1 SUMMARY	38
	7.2 CONCLUSION.....	38
	REFERENCES	39
	APPENDIX	
	A. JOURNAL PAPER	40
	B.PLAGIARISM REPORT.....	47
	C. SCREENSHOTS.....	49

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO.
4.1.1	DOCKER	9
4.1.2	EDGEX	10
4.1.3	RASPBIAN	11
4.1.4	PYTHON 3	12
4.1.5	FLASK	13
4.1.6	UWSGI	14
4.1.7	NGINX	15
4.2.1	RASPBERRY PI	17
4.2.2	SHT20	18
5.1	ARCHITECTURE	21
5.2.1	MODBUS	23
5.2.5	EDGEX WORKFLOW	31
5.3a	MICROSERVICES	32
5.3b	DEVICE PROFILE	33
5.3c	DEVICE CREATION	34
5.3d	OUTPUT DATA	35

LIST OF TABLES

TABLE NO	TABLE NAME	PAGE NO.
4.1	SOFTWARE REQUIREMENTS	12
4.2	HARDWARE REQUIREMENTS	16
5.2.2	MODBUS MEMORY AREA	24
5.2.3.1a	RTU MESSAGE FRAME	26
5.2.3.1b	RTU COMMUNICATION MODES	27
5.2.3.2a	TCP MESSAGE FRAME	29

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

EdgeX Foundry is a latest technology that incorporates both Edge Computing and Internet of Things into a single platform making life much easier for IoT developers. Temperature Monitoring System can be implemented in a variety of use cases ranging from industrial temperature monitoring to household temperature monitoring. Taking care of abnormal rise in temperature manually is a very hectic and time-consuming task, hence the aim of this project is to build an automated temperature monitoring system.

This project focuses on industrial temperature monitoring, where a 24/7 automated temperature monitoring system functions, pushes the data into the database followed by a web server, view the real-time data using a user interface and also alert the required when there are anomalies/threshold breaches in the data thus reduces human work/monitoring to a high extent. The key feature of this project is the usage of EdgeX as it is highly scalable and secure.

The UI plays a very crucial role by displaying the current and the historic data. The data is represented in the form of a graph for better visualization and analysis. Google Chart API has been used to visualize the data in the form of graphs. The UI and alerting system are the user side enhancements to solve the purpose of this project. This project is focused on solving fire accidents and a sudden rise or drop in temperature in various industries. Thus, producing an economically feasible and functionally scalable model to monitor the temperature in industries.

1.2 MOTIVATION

Massacres caused by fire have started to increase either because of the rise in temperature on our planet or because of man-made effects. Industrial fire and explosion data research shows that over 15 Lakh cases were registered under industrial fires and explosions throughout the world from 2018 to 2020, where most of the massacres were caused majorly because of human error or man-made effects.

From time to time, the world and the people living on it have started to update their living and upgrade their existence towards having a better and safe life. Thereby, awareness towards such massacres has also increased among the industries and to also avoid such situations from happening and repeating henceforth. This project has been established to rule out human error and avoid such massacres from getting repeated.

CHAPTER 2

LITERATURE SURVEY

2.1 LITERATURE REVIEW

The temperature monitoring system is not that rare in the industry and has various implementations and projects developed on that but in this project, the integration of EdgeX and the full-stack system makes it much more efficient, reliable, scalable and independent.

2.1.1 Storage facilities using ATMEGA 328 Microcontroller

[3] is the most closely related existing system when compared with the proposed system. It monitors the temperature of the storage rooms in industries using an ATmega 328 microcontroller along with a wifi module and GPRS, the aim of the system is to monitor the temperature regularly and preserve the items in the room for a longer duration.

2.1.2 Rotor Temperature of Flight's Engine

In [1] the rotor temperature of the flight's engine is monitored to avoid malfunctioning of the engine. A series of 16 sensors are used to achieve the task. An alert is generated to the captain of the flight when the temperature reaches the threshold.

2.1.3 Food Industry

[8] describes how important is temperature monitoring in meat supply chains and in the food industry. An extensive temperature monitoring is done on the surroundings and on the meat itself and accordingly, vary the temperature of the surroundings to keep the meat fresh as long as possible.

2.1.4 Temperature Monitoring System using ESP8266 module

[4] uses an ESP8266 microcontroller with an in-built wifi module and a temperature sensor to monitor the temperature of the surroundings. The data is collected and pushed to the cloud, an alert is triggered when the assigned threshold has been reached. A cloud dashboard is created for real-time monitoring.

2.1.5 Home Automation

In [6] temperature monitoring is concentrated more towards Home Automation. The real-time temperature data of various rooms can be viewed through a UI along with past history. It uses WSN (Wireless Sensor Network) to collect the data and communicate with the cloud and other microcontrollers. It modulates the temperature by turning off/on the air conditioner whenever required.

2.2 PROPOSED SYSTEM

The “Temperature Monitoring System using EdgeX” is a complete hardware and software integrated full-stack Internet of Things based system to make temperature monitoring in industries a much easier task. The system is divided into three parts, the hardware segment followed by the interface segment and ending at the software segment. The first segment comprises, a raspberry-pi micro-computer which is connected with a DHT11, temperature and humidity monitoring sensor. The raspberry-pi has the EdgeX software running in it, on top of a docker as a docker container/image.

In the interface segment, we have the flask micro web-framework, uWSGI application-server and Nginx web-server running inside the EdgeX. The mentioned software applications are responsible for the communication between the hardware and the UI, they act as a bridge allowing both the ends of the system to communicate with ease. The software segment comprises, a User Interface and an alerting system, which are functioning outside of the EdgeX framework. The UI enhances visualizing the data through graphs and monitoring the data at regular intervals.

The proposed system is much more scalable and secure compared to the other systems/projects due to the implementation of EdgeX. Through EdgeX updating, the working of the system is much easier and scalable as 100 such systems can be updated at one go without having to modify the code individually on each system. The EdgeX uses a reverse-proxy during the communication of data making it much secure than the other systems. EdgeX provides software and hardware enhancements to the system through its various micro-services which lie under the skin of its service layers. The outcome of this system is to make temperature monitoring a completely automated process and visualize the monitored data.

CHAPTER 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

3.1 AIM

The aim is to build an automated industrial temperature monitoring system that can monitor its surrounding temperature at regular intervals, process the data, collect and store the data in a database, upload the data to a web-server, couple the web-server with a UI and prepare an alerting system. To reduce risk factor by implementing this real-time temperature monitoring system in households and factories/industries.

The key goal is to produce a system which works continuously without any hindrance and provides the correct real-time data as expected. To implement this system on a large scale in various use cases and expand the scope of this project. To make sure the system is working intact and data is visualized as required.

3.2 OBJECTIVES

- Monitor temperature data without any hindrance - The sensor and software integration has to remain intact, sensor has to work continuously and successfully for long hours. The wiring should be precise so that the data does not change its path and reach else where.
- Process the data rapidly - The sensor data should travel from sensor to board, edgex has to be able to access the data from the board, data has to be pushed into the sqlite database.
- Pushing data to the server - The data has to be pushed to the server at regular intervals. There should not be any loss of data during this the process as that may lead to old/existing data from not getting updated.

- Establishing User Interface - A user interface has to be created and successfully interfaced with the web server. The connection should be intact, if not may lead to failure of the system. The UI must easily understandable and scalable to suite various use cases.

3.3 LIMITATION AND SCOPE

3.3.1 Scope

- It can be implemented in household along with fire alarms
- It can be implemented in dense forests to identify forest fires
- It can be coupled with a compressor in cold storage facilities to automatically modify and stabilize the temperature.
- It can be used in cargo/shipment containers to monitor its temperature during its journey to its destination.
- Can add location services in the future.

3.3.2 Limitations

- It requires continuous and uninterrupted internet connection.
- Sensors have to be replaced once in a while so that they don't wear out completely.
- There are possibilities of short circuiting if multiple sensors are connected.
- EdgeX micro services must be up and running throughout.

CHAPTER 4

SOFTWARE AND HARDWARE REQUIREMENTS

The proposed project is a combination of hardware devices and software applications working synchronously to produce the desired data. The hardware devices and software applications are as follows.

4.1 SOFTWARE REQUIREMENTS

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Software	Description
Docker	Collection of containers and behaves like a virtual machine
EdgeX Foundry	IoT Edge Gateway
Raspian OS	OS which runs in the raspberry pi
Python 3	To write and run python scripts
Flask	Micro web framework
uWSGI	Application Server
Nginx	Web Server
HTML, CSS, jQuery	User Interface Development

Table 4.1: Software Requirements of the application

4.1.1 Docker

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.

Docker is also described as a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application. And importantly, Docker is open source. This means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that aren't available out of the box.

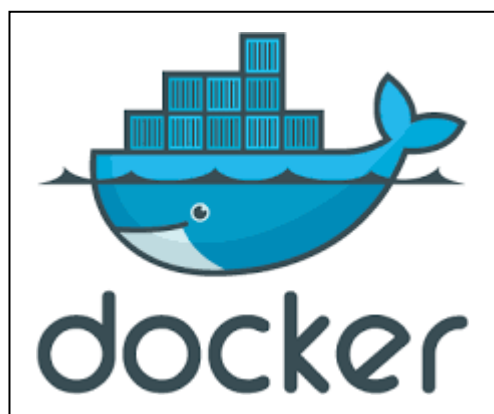


Fig 4.1.1 Docker

4.1.2 EdgeX Foundry

EdgeX Foundry is an open source, vendor neutral, flexible, interoperable, software platform at the edge of the network, that interacts with the physical world of devices, sensors, actuators, and other IoT objects. In simple terms, EdgeX is edge middleware - serving between physical sensing and actuating "things" and our information technology systems. The EdgeX platform enables and encourages the rapidly growing community of IoT solution providers to work together in an ecosystem of interoperable components to reduce uncertainty, accelerate time to market, and facilitate scale.



Fig 4.1.2 EdgeX

By bringing this much-needed interoperability, EdgeX makes it easier to monitor physical world items, send instructions to them, collect data from them, move the data across the fog up to the cloud where it may be stored, aggregated, analyzed, and turned into information, actuated, and acted upon. So EdgeX enables data to travel northwards towards the cloud or enterprise and back to devices, sensors, and actuators. The goal of EdgeX is the simplification and standardization of the foundation for tiered edge computing architectures in the IoT market while still enabling the ecosystem to provide significant value-added differentiation.

4.1.3 Raspbian OS

Raspbian OS is one of the official Operating systems available for free to download and use. The system is based on Debian Linux and is optimized to work efficiently with the Raspberry Pi computer. As we already know an OS is a set of basic programs and utilities that runs on a specified hardware, in this case the Pi. Debian is very lightweight and makes a great choice for the Pi.

The Raspbian includes tools for browsing, python programming and a GUI desktop. The Raspbian desktop environment is known as the "Lightweight X11

Desktop Environment” or in short LXDE. This has a fairly attractive user interface that is built using the X Window System software and is a familiar point and click interface.

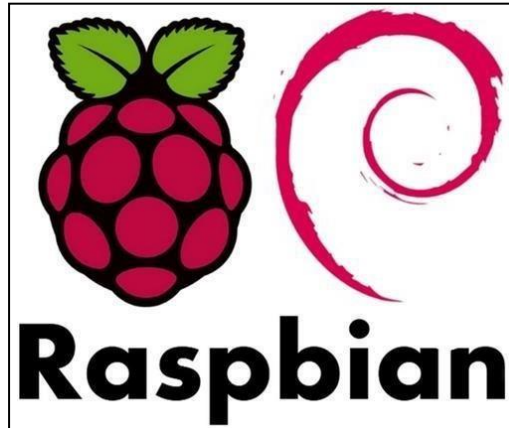


Fig 4.1.3 Raspbian

Raspberry Pi OS is the recommended operating system for normal use on a Raspberry Pi. It is a free operating system based on Debian, optimised for the Raspberry Pi hardware, comes with over 35,000 packages such as precompiled software bundled in a nice format for easy installation on your Raspberry Pi. Raspbian OS is a community project under active development, with an emphasis on improving the stability and performance of as many Debian packages as possible.

4.1.4 Python 3

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

- Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.



Fig 4.1.4 Python 3

4.1.5 Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

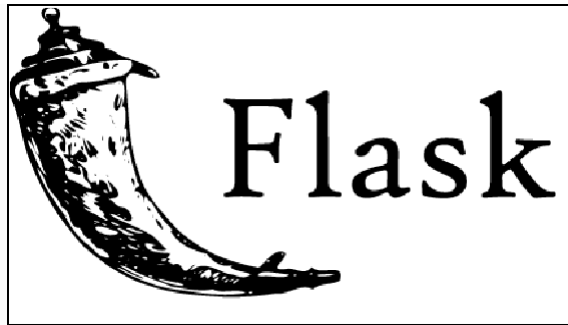


Fig 4.1.5 Flask

Flask is considered more Pythonic than the Django web framework because in common situations the equivalent Flask web application is more explicit. Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running. Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

4.1.6 uWSGI

uWSGI is a software application that "aims at developing a full stack for building hosting services". It is named after the Web Server Gateway Interface (WSGI), which was the first plugin supported by the project. uwsgi (all lowercase) is the native binary protocol that uWSGI uses to communicate with other servers.

uWSGI (Web Server Gateway Interface) is an application server interface that acts as a communication pipe between the python application and the web-server. It is implemented to provide smooth communication between the two ends.

uWSGI is often used for serving Python web applications in conjunction with web servers such as Cherokee and Nginx, which offer direct support for uWSGI's native uwsgi protocol. For example, data may flow like this: HTTP client ↔ Nginx ↔ uWSGI ↔ Python app.

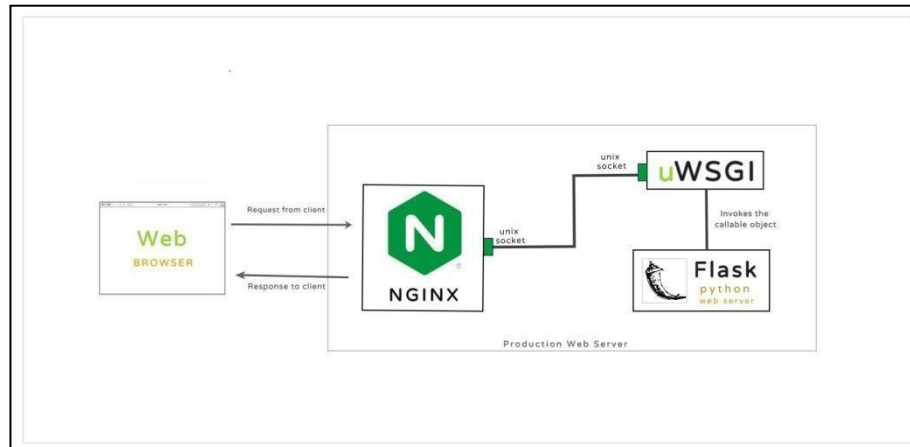


Fig 4.1.6 uWSGI

4.1.7 Nginx

Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. The software was created by Igor Sysoev and publicly released in 2004. Nginx is free and open-source software, released under the terms of the 2-clause BSD license. A large fraction of web servers use NGINX, often as a load balancer.

Nginx is a web server that connects and communicates between the HTTP client/web application and the backend software application/ application server. It serves the requirements of the HTTP client and maintains a log/record of the happenings. Nginx can be deployed to serve dynamic HTTP content on the network using FastCGI, SCGI handlers for scripts, WSGI application servers or Phusion Passenger modules, and it can serve as a software load balancer.

Nginx uses an asynchronous event-driven approach, rather than threads, to handle requests. Nginx's modular event-driven architecture can provide more predictable performance under high loads.

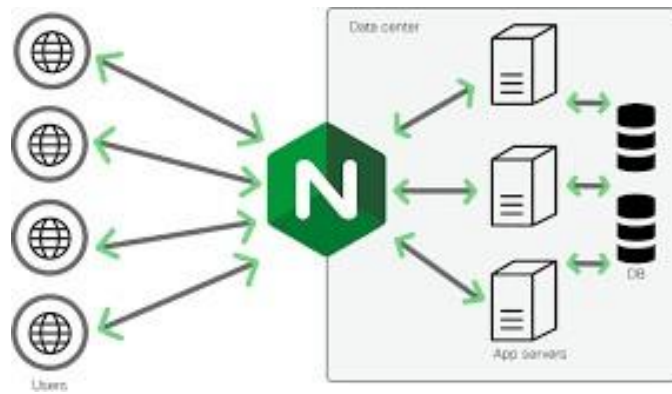


Fig 4.1.7 Nginx

4.2 HARDWARE REQUIREMENTS

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements. A second meaning of the term of system requirements, is a generalization of this first definition, giving the requirements to be met in the design of a system or sub-system.

Hardware	Description
Raspberry Pi 3B+	1 GB RAM, 16GB ROM with Raspbian Lite OS

SHT20	Temperature monitoring sensor
Micro SD card	16 GB acts as ROM
Windows 7 or above installed PC	To work with Raspberry Pi on headless mode

Table 4.2: Hardware Requirements of the application

4.2.1 Raspberry Pi 3B+

The latest Raspberry Pi 3 Model B+ has a faster 64-bit 1.4GHz quad core processor, 1GB of RAM, faster dual-band 802.11 b/g/n/ac wireless LAN, Bluetooth 4.2, and significantly faster 300Mbit/s ethernet. It can be considered as a single board computer that works on LINUX operating system. The board not only has tons of features it also has terrific processing speed making it suitable for advanced applications.

- 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU (BCM2837)
- 1GB RAM (LPDDR2 SDRAM)
- On-board wireless LAN - dual-band 802.11 b/g/n/ac (CYW43455)
- On-board Bluetooth 4.2 HS low-energy (BLE) (CYW43455)
- 4 x USB 2.0 ports
- 300Mbit/s ethernet
- 40 GPIO pins
- Full size HDMI 1.3a port
- Combined 3.5mm analog audio and composite video jack
- Camera interface (CSI)
- Display interface (DSI)
- microSD slot
- VideoCore IV multimedia/3D graphics core @ 400MHz/300MHz

The Raspberry Pi hardware has evolved through several versions that

feature variations in the type of the central processing unit, amount of memory capacity, networking support, and peripheral-device support. The dual-band wireless LAN comes with modular compliance certification, allowing the board to be designed into end products with significantly reduced wireless LAN compliance testing, improving both cost and time to market.

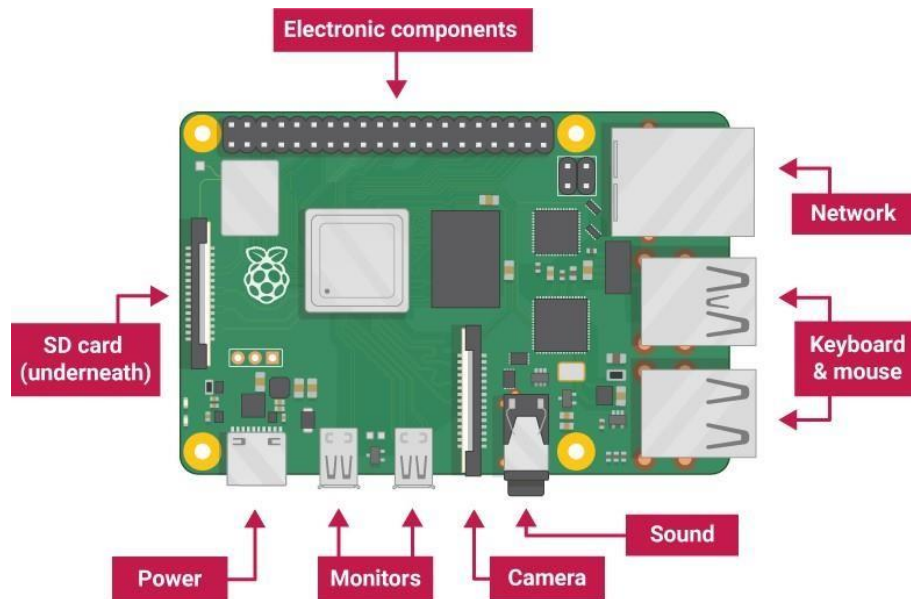


Fig 4.2.1 Raspberry Pi

4.2.2 TEMPERATURE SENSOR (SHT20)



Fig 4.2.2 SHT20

SHT20 adopts industrial grade chip and high-precision imported SHT20 temperature and humidity sensor to ensure excellent reliability, high precision and interchangeability. RS485 hardware interface (with lightning protection design), the protocol layer is compatible with the standard industrial Modbus-Rtu protocol. This product integrates MODBUS protocol with common protocol. Users can choose communication protocol by themselves. The common protocol has an automatic upload function connecting RS485 will automatically output temperature and humidity through serial port adjustment tool.

FEATURES

- Support MODBUS RTU protocol
- RS485 supports 1000 meters communication
- Standard DIN35 mounting rails
- High precision

4.3 METHODOLOGY

Requirement specification: The phase wherein requirements and needs are defined.

Planning: In the requirement specification phase, needs were determined. In the planning phase, we set goals and that we hope to accomplish.

Implementation and Testing: Phase 1 and phase 2 indicated the needs and goals. In the implementation and testing phases, the goals are being subdivided into smaller, more specific categories, with emphasis on priorities. The subcategories such as designing, testing and distribution are being tested.

Closure: The phase wherein every document and result gathered from the project is assessed. Best practices are delivered here, and a careful evaluation of positive and negative outcomes of projects is then used for current and future tasks.

4.3.1 IOT THE BACKBONE

Internet Of Things is basically connection of various devices via internet which helps in the communication of data among the devices. The Internet of Things is essentially a system of machines or objects outfitted with data collecting technologies so that those objects can communicate with one another. IOT is all about the Network, Devices and Data.

A system is designed and implemented that gathers information about the surrounding temperature from its sensors, which will report to a lower level controller; which in turn will send to a local controller which is responsible to send the information to the server. The information gathered by the sensors need to be accurate for the information to be accepted. Based on the information from the sensors the server flash the results on a UI in a format that is suitable for the users to access it from their mobile devices to get the information about the surrounding temperature of the place where the system is installed.

4.3.2 PROJECT PLAN AND TIMELINE

The project is majorly divided into four steps,

- **Step 1:** Planning and Design.
- **Step 2:** Hardware configuration.
- **Step 3:** Software Development.
- **Step 4:** Testing.

In the **Planning and Design Stage**, a lot of tinkering took place by experimenting on different ideas while maintaining a clear milestone and other objectives of the project. After having the general understanding of the project, the analysis process started where the system is viewed from a working point of view, by taking into account implementation, considering feasibility to the basic component level. The experimentation to place with well defined and clear ideas.

On the **Hardware Configuration stage**, the sensors and controllers were configured which is the base line for the overall project. The next step was object(vehicle) detection and communication among the controllers, the testing took place over different communication modules and the one with the least cost and better performance and meets the standards that were set at the beginning was selected. The sensors were configured to send data gathered from sensors to the EdgeX.

On the third stage, **Software Development**, user end applications with simple, easy to use interface and design was implemented. Information that is gathered by sensors and sent by the micro-controller to the database for processing. In the later stage the focus was more towards the development of web application. The app development has two components ,one is displaying temperature data and another is to visualize as graph. The user can accesses data for the most part using the web application. The sensors keep sensing data for every 10 seconds thereby increasing accuracy and reducing error.

The last stage is **Testing and Debugging**, each module as individual components and as components of the whole system was checked precisely. Tests were designed based on the initial requirements of such a system that is feasible for the city of Chennai.

CHAPTER 5

SYSTEM ARCHITECTURE AND ALGORITHM

5.1 SYSTEM ARCHITECTURE

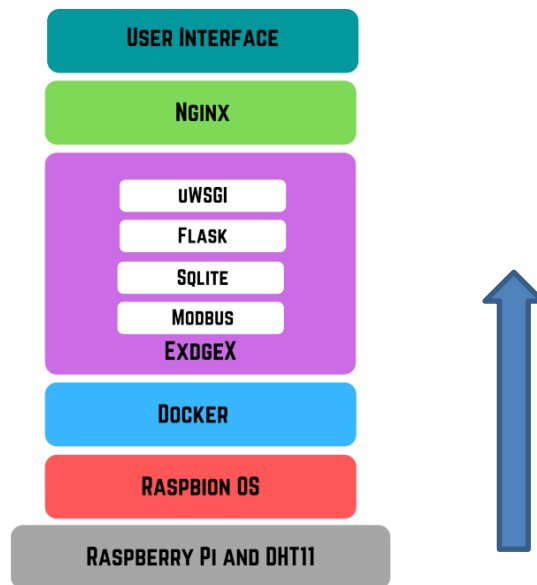


Fig 5.1 Architecture

1. The DHT11 sensor is connected to the Raspberry-Pi via jumper cables. DHT11 senses the surrounding temperature and humidity also pass that data to Raspberry Pi through the data pin.
2. The Raspberry-Pi being a micro-computer requires an OS to run on it. Raspbian OS runs as the OS on the Raspberry-Pi, it is also the native OS of Raspberry-Pi.
3. The Docker is then installed on top of the OS to work as an interface between the containerized application and the OS. The docker helps the containers to individually access and make use of the features of the OS.

4. The EdgeX Foundry software is downloaded as a container on top of a docker. The data is accessed, modified and sent to the web-server through EdgeX and its micro-services.
5. The MODBUS micro-service is responsible for the communication of data from and to the sensor. This micro-service lies in the device-service layer.
6. SQLite micro-service is the database management system used in this project. It collects the data from the MODBUS device service and stores the data and represents them in the form of tables, modifications can be performed on the data present in tables using SQLite commands.
7. Flask micro-service is present in the application services layer and is the backbone of the web application. The code for what should happen and when it should happen is present inside the flask framework and is responsible for the communication of data from the database to the application-server (uWSGI).
8. uWSGI micro-service acts as the application-server inside the EdgeX and is present in the application services layer. It is responsible for the peaceful communication of data between the web-server (Nginx) and the flask framework.
9. The Nginx web-server is outside the bounds of the EdgeX software and behaves like an interface between the UI and application-server (uWSGI). It sends data to and receives data from EdgeX.
10. The User Interface is the final endpoint where the entire temperature and humidity can be viewed and analyzed. The data is visually represented with the help of a graph that was generated using the Google Chart API. The UI increases the transparency of the system and delivers the needs of the users. The requests made by the user are sent to the web-server and the desired result is then displayed on the UI.

5.2 MODBUS

5.2.1 OVERVIEW

Modbus is an application layer communication protocol for exchanging data between electronic devices over serial lines (RS 232 or RS 485) or Ethernet, and it's widely used in process and factory automation.

Modbus is an open protocol and can be used by any organization or individual, Schneider Electric USA holds the trademark.

Modbus provides a common language for devices and equipment to communicate with one and another. For example, Modbus enables devices on a system that measures temperature and humidity connected on the same network and communicate the results to a supervisory computer or Microcontroller.

Modbus serial protocol is a single master, multiple slaves protocol, where each slave serves the requirement of the master on its request. Modbus TCP/IP is a client and server protocol where the master is the server and its slaves are the clients.

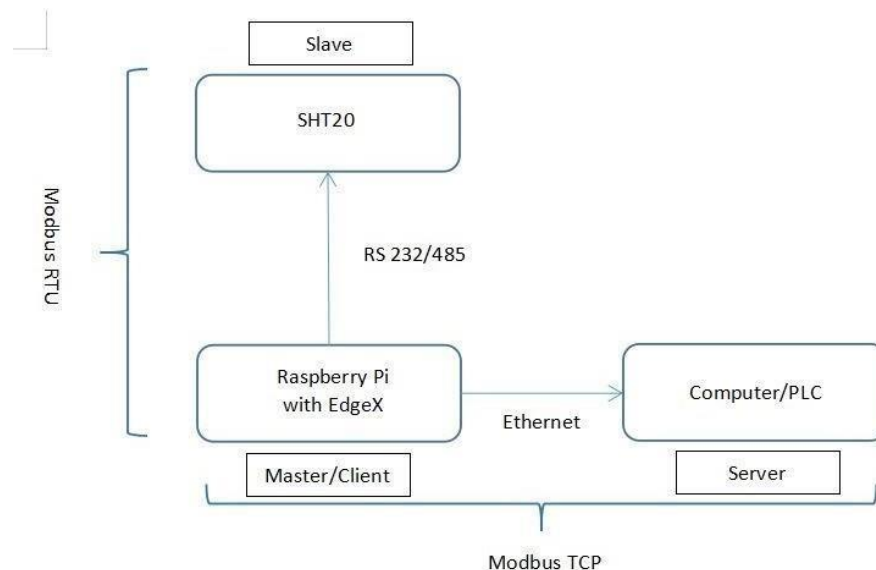


Fig 5.2.1 Modbus

5.2.2 MODBUS MEMORY AREA

Address Range	Access	Object Type	Size
00001-09999	Read/Write	Coil (Discrete Output)	1 bit
10001-19999	Read	Discrete Input	1 bit
30001-39999	Read	Input Register	16 bits
40001-49999	Read/Write	Holding Register	16 bits

Table 5.2.2 Modbus Memory Area

The Modbus memory area is split into four parts where the first is coils (discrete output) which is of 1 bit in size and stores the actuation requests such as to open and close a valve. Both read and write functions can be performed on coils i.e reading the current status of valve and assigning open or close operations on the valve.

The second is the discrete input which is also of size one bit and stores on/off status, like that of a switch. Only read operation can be performed on this memory area and thus stores on/off status.

Input register is the third part of the memory area, it is of size 16 bits, and stores the values read from sensor devices such as SHT20. It is a read only memory area and thereby data present in this memory area can only be read and not modified. Thus, temperature, humidity, current and power sensor devices use the Input register memory area since the data generated from these sensor devices cannot be modified.

The fourth part is the holding register. It is also of size 16 bits, stores values read from sensors such as water valves. Both read and write operations can be performed on the memory register thereby data can be read and modified as well. For example, consider a water valve when completely open is flowing at a rate of

1L for every 10 seconds. The data Valve is 100% open is stored in holding register and if rate has to decreased later it can be modified on the holding register itself.

5.2.3 TYPES OF MODBUS

Modbus is majorly of two types Modbus RTU and Modbus TCP, where the sooner one is implemented in a serial network and latter one is implemented in an Ethernet network.

5.2.3.1 Modbus RTU (Remote Terminal Unit)

Modbus RTU is an open serial protocol derived from Modicon's Master/Slave architecture. It is a commonly used serial level protocol due its ease of use and reliability. Its major area of application is in Industrial Automation Systems. This protocol primarily uses an RS-232 or RS-485 serial interfaces for communication and is supported by almost every commercial and data acquisition software program in the marketplace. This makes it very easy to integrate Modbus compatible equipment into new or existing monitoring and control applications.

Modbus RTU being a serial protocol uses master/slave architecture with only one master which can connect to a minimum of one slave and a maximum of 247 slaves. The master can be any host supervisory computer or even a microcontroller with a host software and the slave can be any sensor device such as SHT20.

5.2.3.1a Modbus RTU Message Frame

Communication between master and slave takes place through message packets. These message packets have a particular frame or template to be followed during communication.

The message frame if of two main parts,

- Protocol Data Unit : It comprises of Data and Function code. This data unit is more specific towards the data and the function to be performed on the data. PDU has the same structure irrespective of change in device.
- Application Data Unit : It comprises of PDU, Address and CRC Error Check. This data unit encapsulates the entire message. ADU's device might change in respect to the change of a device.

The message frame consists of four sections,

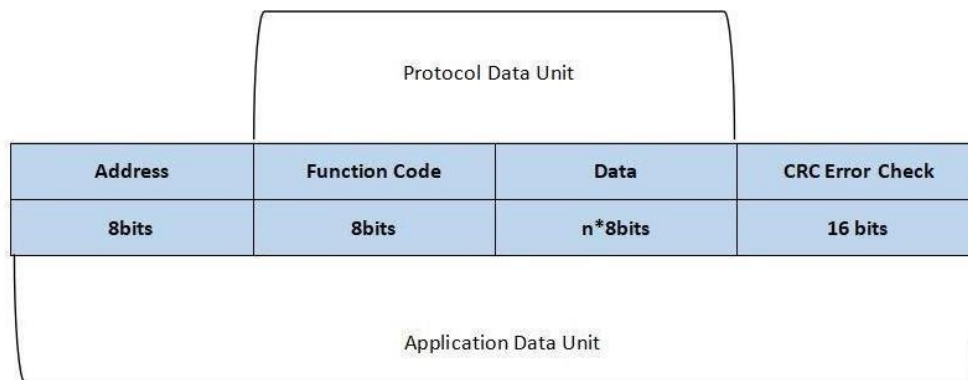


Table 5.2.3.1a RTU Message Frame

- Address: This section consists of the slave address the master wants to communicate with.
- Function Code: The function code comprises the action to be performed on the data such as read/write.
- Data: This section consists of the data such as temperature data.
- CRC Error Check: Cyclic Redundancy check is an error checking mechanism performed on the data to verify if there is error in the data or loss of data during communication.

5.2.3.1b Modbus RTU Communication Modes

RS 232	RS 485
Peer to peer, one master is connected to only one slave.	One to many/multi drop, one master is connected to multiple slaves.
3 wire communication, RX, TX and ground.	Two wire communication
Full Duplex Communication	Half Duplex Communication
Short distance communication (50 feet)	Long distance communication (1000 feet)
Noise in data is generated due to the ground wire	Noise immunity since ground is not involved.

Table 5.2.3.1b RTU Communication Modes

5.2.3.1 c Message Communication Using RS 485

In Modbus RTU messages flow through RS 485 which is a multi-drop communication mechanism. One master is connected to multiple slaves through a RS 485.

In RS 485, the master broadcasts the message to all the slaves, the slave which matches the specified address sends a response to the request, whereas the slaves which do not match the specified address field in the message frame just dismiss or suspend the request.

The slave sends a response with its address followed by data and CRC error check as a response message frame. The CRC is 0 if there is no error else it is returned as 1/high.

5.2.3.2 Modbus TCP

Modbus TCP is the same as RTU but it uses a TCP interface that runs on Ethernet. Modbus TCP uses Ethernet for the traversal of message packets instead of serial lines such as RS 232 or 485. Modbus TCP uses both TCP (Transmission Control Protocol) and IP (Internet Protocol) to route the message packets between the master and the slave. TCP/IP allows blocks of binary data to be exchanged between computers. The primary function of TCP is to ensure that all packets of data are received correctly, while IP makes sure that messages are correctly addressed and routed.

The Modbus TCP uses the client/server architecture, where the master is the client and the slave is the server. The slave serves the requirements of the client/master. One client is interfaced with multiple servers and these servers are in turn interfaced with multiple other clients. For example, let's say client 1 is in need of temperature data and it sends a request to the server. But currently, the server is not having temperature data, so it waits until client 2 sends the sensor data and then transmits that data back to client 1 as requested.

5.2.3.2a Modbus TCP Message Frame

The Modbus TCP message frames inherits the PDU of an RTU and adds a header to it for the purpose of routing the message.

The message frame is of three parts,

- Protocol Data Unit: As inherited from RTU, contains function code and data.
- Modbus Application Protocol Header: Contains all the required information for routing message packets.
- Application Data Unit: Encapsulates both MBAP header and PDU into a single unit.

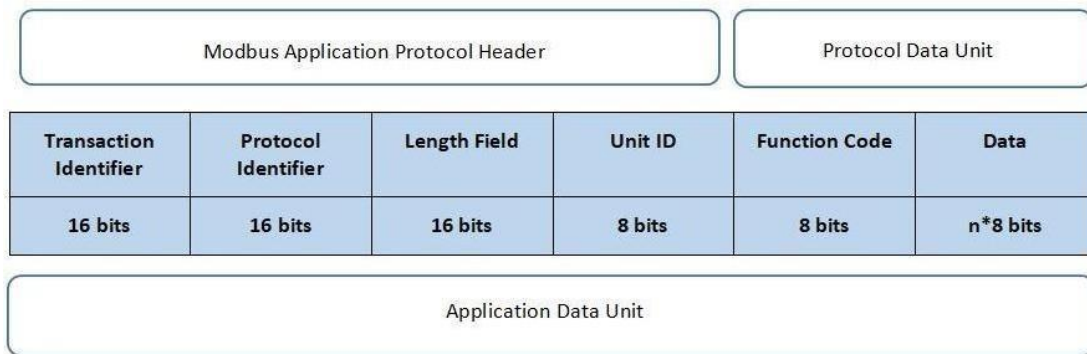


Table 5.2.3.2a TCP Message Frame

The message frame consists of 6 sections,

- **Transaction Identifier:** When a client sends several messages over the same TCP link without waiting for an answer, this identification field is used for transaction pairing.
- **Protocol Identifier:** For Modbus services, this field is always 0 and for further extensions, other values are reserved.
- **Length:** This field is a byte count of the remaining fields and includes the unit identifier byte, function code byte, and the data fields.
- **Unit Identifier:** This field is used to identify a remote server located on a non TCP/IP network. In a typical Modbus TCP/IP server application, the unit ID is set to 00 or FF, ignored by the server, and simply echoed back in the response.
- **Function Code:** The function code comprises the action to be performed on the data such as read/write.
- **Data:** This section consists of the data such as temperature data.

The Modbus error checking field (checksum) is not used, as the standard Ethernet TCP/IP, link-layer checksum methods are instead used to guaranty data integrity. The Modbus TCP/IP Device Data Unit is inserted in the data field of a standard TCP

frame and sent over TCP to the well-known Modbus system port 502, which is dedicated to Modbus applications. Modbus TCP/IP clients and servers use port 502 to listen for and receive Modbus data.

5.2.4 Modbus In EdgeX

Modbus is a microservice under the device service layer. It is responsible for the communication between edgex and the sensor device. It communicates with all the sensor devices which support modbus protocol. The modbus device service is present at port 49991.

The data can move in two directions through modbus, from the sensor device to core services layer or from core services layer to sensor device. Let's first discuss the sooner, the modbus communicates with the sensor device such as SHT20 temperature monitoring sensor and sends the sensor data to the core data microservice present in the core services layer where it is stored in a database such as redis. In the latter approach, command microservice present in the core services layer sends data to the sensor device.

5.2.5 Woking of Modbus in EdgeX

Let us consider an industrial automation system where an SHT20 temperature monitoring sensor device is used in a food storage facility and a compressor is used to regulate the temperature of the facility. These two sensor devices are connected to edgex via modbus device service.

The SHT20 sensor device communicates to edgex via modbus device service and pushes temperature data to the core data at the device services layer every 5 seconds. The data present inside the core data microservice is stored in a redis database and this data is then pulled by the application services layer via a message queue called ZeroMQ. Various functions are being performed on the data and the data is then sent to rules engine micro service present in the support services layer, where a Kuiper engine checks a set of conditions such as if the temperature data is

The diagram illustrates the EdgeX Foundry Platform Architecture, showing the flow of data and services between different layers:

- "NORTHBOUND" INFRASTRUCTURE AND APPLICATIONS:** This layer at the top includes icons for a router, server racks, and a cloud with "IHM" (Industrial HMI/SCADA) text. It connects to the "APPLICATION SERVICE" via a blue arrow.
- LOOSELY-COUPLED MICROSERVICES FRAMEWORK:** The central part of the architecture, divided into several functional blocks:
 - EXPORTING AND APPLICATION SERVICES:** Contains "ADDITIONAL SERVICES" and the "APPLICATION SERVICE". It connects to the northbound layer and the core services.
 - SUPPORTING SERVICES:** Includes "RULES ENGINE", "SCHEDULING", "ALERTS & NOTIFICATIONS", and "LOGGING". It connects to the application service and core services.
 - CORE SERVICES:** A central dark purple block containing "CORE DATA", "COMMAND", "METADATA", and "REGISTRY & CONFIG". A note states "ALL MICROSERVICES INTERCOMMUNICATE VIA APIs".
 - DEVICE SERVICES (ANY COMBINATION OF STANDARD OR PROPRIETARY PROTOCOLS VIA SDK):** Includes "REST", "OPC-UA", "MODBUS", "BACNET", "ZIGBEE", "BLE", "MQTT", "SNMP", and "GPIO". It connects to the core services and southbound devices.
- SECURITY:** A vertical block on the left side, including "REVERSE PROXY", "ADDITIONAL SECURITY SERVICES", and "SECRET STORE".
- MANAGEMENT:** A vertical block on the right side, including "ADDITIONAL SERVICES", "MGMT SERVICE AGENT", and "REMOTE/LOCAL GUI".
- "SOUTHBOUND" DEVICES, SENSORS AND ACTUATORS:** The bottom layer, featuring various icons for devices like sensors, actuators, and communication modules. It connects to the device services via a blue arrow.
- SDK (Software Development Kit):** Represented by a briefcase icon with a plug, it is shown in the top right and bottom right, indicating its role in connecting applications and devices to the platform.

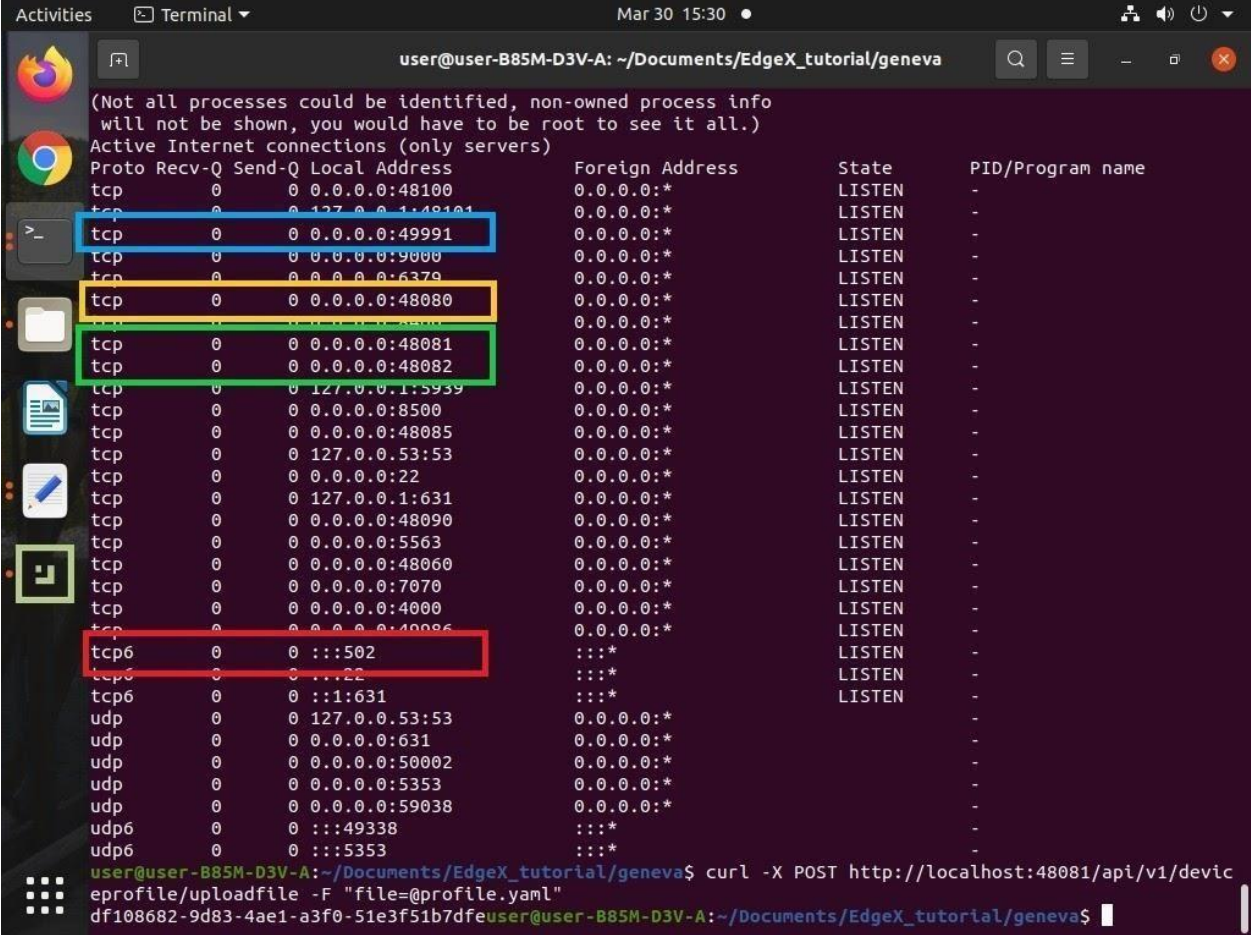
Arrows indicate the flow of data and services: blue arrows show the primary path from southbound devices through the core services to northbound infrastructure, while orange arrows show secondary or supporting paths between services.

31

5.3 PROJECT IMPLEMENTATION

1. Verify if edgex microservices are up and running

Use the command “ netstat -tulpn ” to view the status of the microservices.



The terminal window shows the output of the command `netstat -tulpn`. The output is a table with columns: Proto, Recv-Q, Send-Q, Local Address, Foreign Address, State, and PID/Program name. Several rows are highlighted with colored boxes: a blue box around the row for port 49991 (Modbus), a yellow box around the row for port 48080 (edgex-core-data), a green box around the rows for ports 48081 and 48082 (edgex-core-metadata and edgex-core-command), and a red box around the row for port 502 (ort). At the bottom, a curl command is shown: `curl -X POST http://localhost:48081/api/v1/deviceprofile/uploadfile -F "file=@profile.yaml"`.

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	0.0.0.0:48100	0.0.0.0:*	LISTEN	-
tcp	0	0	127.0.0.1:48101	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:49991	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:9000	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:6379	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:48080	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:48081	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:48082	0.0.0.0:*	LISTEN	-
tcp	0	0	127.0.0.1:5939	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:8500	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:48085	0.0.0.0:*	LISTEN	-
tcp	0	0	127.0.0.53:53	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN	-
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:48090	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:5563	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:48060	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:7070	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:4000	0.0.0.0:*	LISTEN	-
tcp	0	0	0.0.0.0:40086	0.0.0.0:*	LISTEN	-
tcp6	0	0	:::502	:::*	LISTEN	-
tcp6	0	0	:::22	:::*	LISTEN	-
tcp6	0	0	:::1:631	:::*	LISTEN	-
udp	0	0	127.0.0.53:53	0.0.0.0:*	-	-
udp	0	0	0.0.0.0:631	0.0.0.0:*	-	-
udp	0	0	0.0.0.0:50002	0.0.0.0:*	-	-
udp	0	0	0.0.0.0:5353	0.0.0.0:*	-	-
udp	0	0	0.0.0.0:59038	0.0.0.0:*	-	-
udp6	0	0	:::49338	:::*	-	-
udp6	0	0	:::5353	:::*	-	-

```
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/deviceprofile/uploadfile -F "file=@profile.yaml"
df108682-9d83-4ae1-a3f0-51e3f51b7dfeuser@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
```

Fig 5.3a Microservices

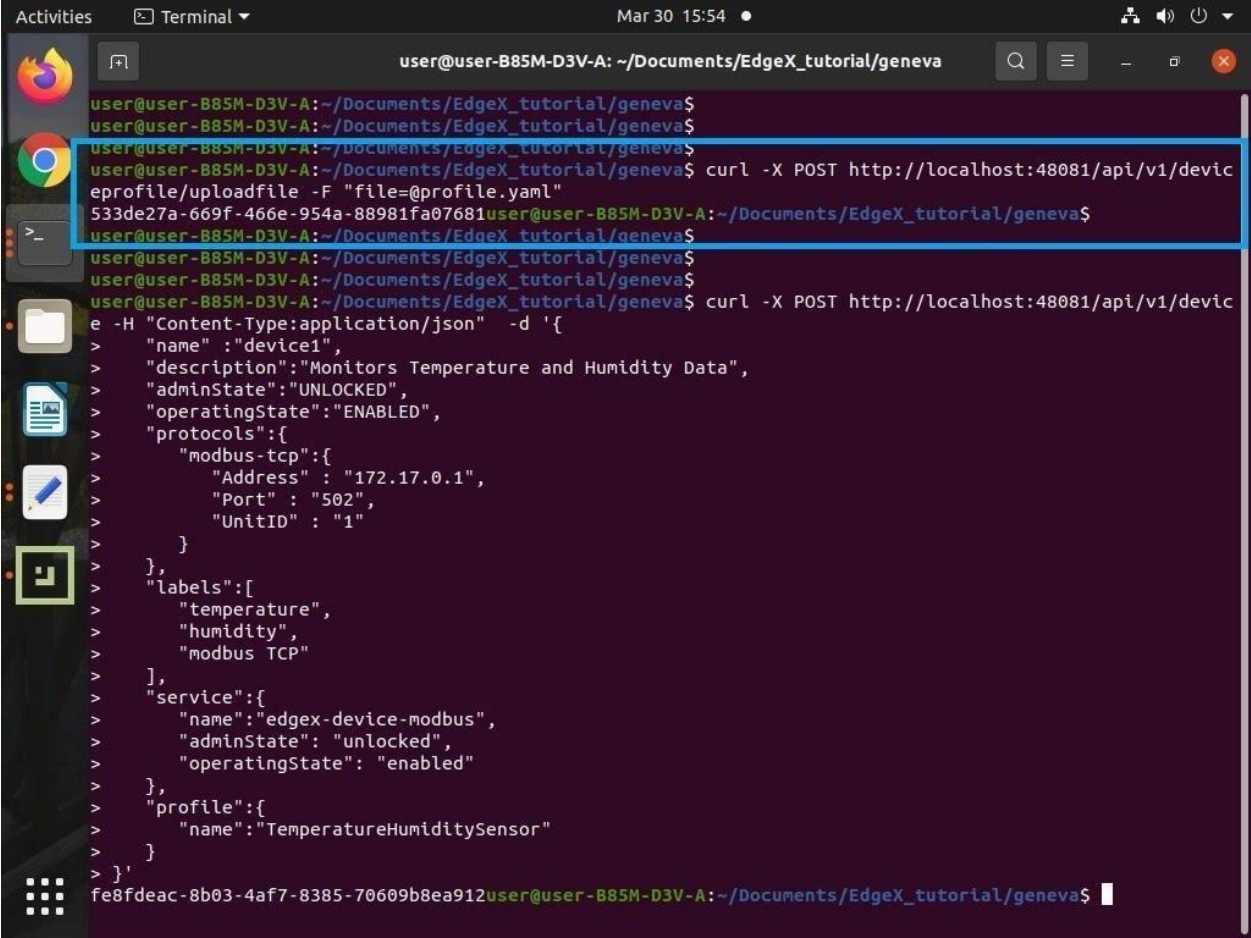
- Blue indicates that Modbus is up and running at port 49991.
- Yellow indicates that edgex-core-data is up and running.
- Green indicates edgex-core-metadata at 48081 and edgex-core-command are up and running.
- Red indicates that ort 502 is also active.

2. Upload device profile

The device profile is a basic description of the device and its dependencies. It describes the operations to be performed on the data such as read/write, starting address of the holding/input registers and details of the sensor device.

The device profile can be uploaded using the post command as follows,

```
curl -X POST http://localhost:48081/api/v1/deviceprofile/uploadfile -F "file=@device_profile.yaml"
```

A terminal window titled 'user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva' shows the execution of a curl command to upload a device profile. The command is: `curl -X POST http://localhost:48081/api/v1/deviceprofile/uploadfile -F "file=@profile.yaml"`. The output shows a 201 status code and a long alphanumeric ID: `533de27a-669f-466e-954a-88981fa07681`. Below this, the user enters a second curl command with a JSON payload: `curl -X POST http://localhost:48081/api/v1/device -H "Content-Type:application/json" -d '{`. The JSON payload defines a device named 'device1' with a description 'Monitors Temperature and Humidity Data', admin state 'UNLOCKED', operating state 'ENABLED', and a modbus-tcp protocol with address '172.17.0.1', port '502', and unit ID '1'. It also includes labels for 'temperature', 'humidity', and 'modbus TCP', and a service named 'edgex-device-modbus' with admin state 'unlocked' and operating state 'enabled'. The profile is identified as 'TemperatureHumiditySensor'. The terminal output shows the JSON is successfully processed, ending with a 201 status code and another long alphanumeric ID: `fe8fdeac-8b03-4af7-8385-70609b8ea912`.

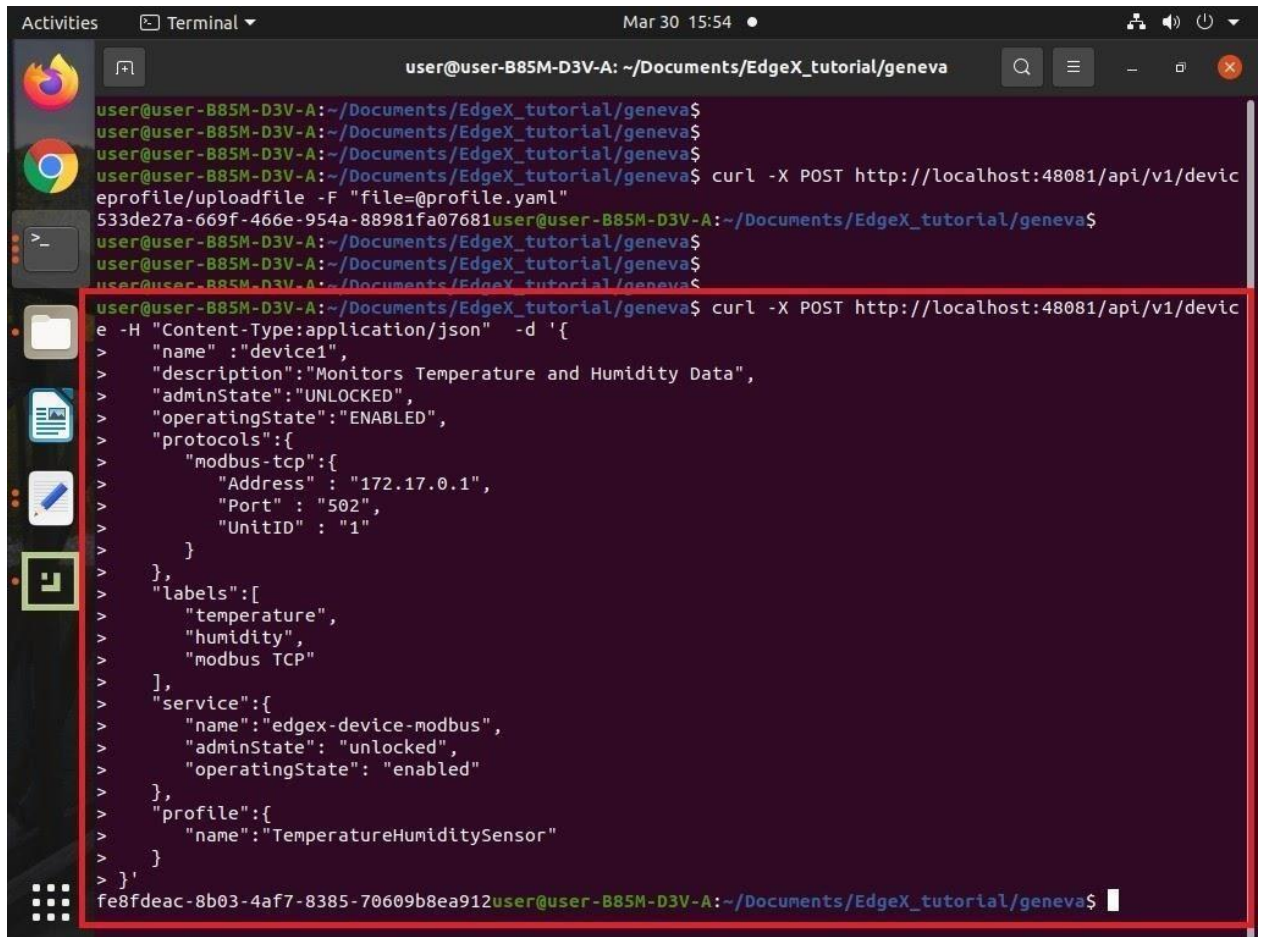
```
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/deviceprofile/uploadfile -F "file=@profile.yaml"
533de27a-669f-466e-954a-88981fa07681user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/device -H "Content-Type:application/json" -d '{
>   "name": "device1",
>   "description": "Monitors Temperature and Humidity Data",
>   "adminState": "UNLOCKED",
>   "operatingState": "ENABLED",
>   "protocols": {
>     "modbus-tcp": {
>       "Address": "172.17.0.1",
>       "Port": "502",
>       "UnitID": "1"
>     }
>   },
>   "labels": [
>     "temperature",
>     "humidity",
>     "modbus TCP"
>   ],
>   "service": {
>     "name": "edgex-device-modbus",
>     "adminState": "unlocked",
>     "operatingState": "enabled"
>   },
>   "profile": {
>     "name": "TemperatureHumiditySensor"
>   }
> }'
fe8fdeac-8b03-4af7-8385-70609b8ea912user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
```

Fig 5.3b Device Profile

The Blue section indicates that a device profile has been successfully uploaded using the above command and a temporary id has been generated for the uploaded device profile. The id generation also confirms that the action was successful.

3. Create device using post command

The device can be created using the POST command as mentioned in the image.

A terminal window titled 'user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva' with a timestamp of 'Mar 30 15:54'. The terminal shows a series of commands and their outputs. The first three commands are simple directory listings. The fourth command is a curl POST request to 'http://localhost:48081/api/v1/deviceprofile/uploadfile' with a file named '@profile.yaml'. The output is a long alphanumeric string: '533de27a-669f-466e-954a-88981fa07681'. The fifth command is another curl POST request to 'http://localhost:48081/api/v1/device' with a JSON payload. This command and its output are highlighted with a red rectangular box. The output is a JSON object containing device details like name, description, adminState, operatingState, protocols, labels, service, and profile. The final line shows a new alphanumeric ID: 'fe8fdeac-8b03-4af7-8385-70609b8ea912'.

```
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$  
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$  
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$  
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/deviceprofile/uploadfile -F "file=@profile.yaml"  
533de27a-669f-466e-954a-88981fa07681user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$  
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$  
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$  
user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$ curl -X POST http://localhost:48081/api/v1/device  
-H "Content-Type:application/json" -d '{  
>   "name" : "device1",  
>   "description": "Monitors Temperature and Humidity Data",  
>   "adminState": "UNLOCKED",  
>   "operatingState": "ENABLED",  
>   "protocols": {  
>     "modbus-tcp": {  
>       "Address" : "172.17.0.1",  
>       "Port" : "502",  
>       "UnitID" : "1"  
>     }  
>   },  
>   "labels": [  
>     "temperature",  
>     "humidity",  
>     "modbus TCP"  
>   ],  
>   "service": {  
>     "name": "edgex-device-modbus",  
>     "adminState": "unlocked",  
>     "operatingState": "enabled"  
>   },  
>   "profile": {  
>     "name": "TemperatureHumiditySensor"  
>   }  
> }'  
fe8fdeac-8b03-4af7-8385-70609b8ea912user@user-B85M-D3V-A:~/Documents/EdgeX_tutorial/geneva$
```

Fig 5.3c Device Creation

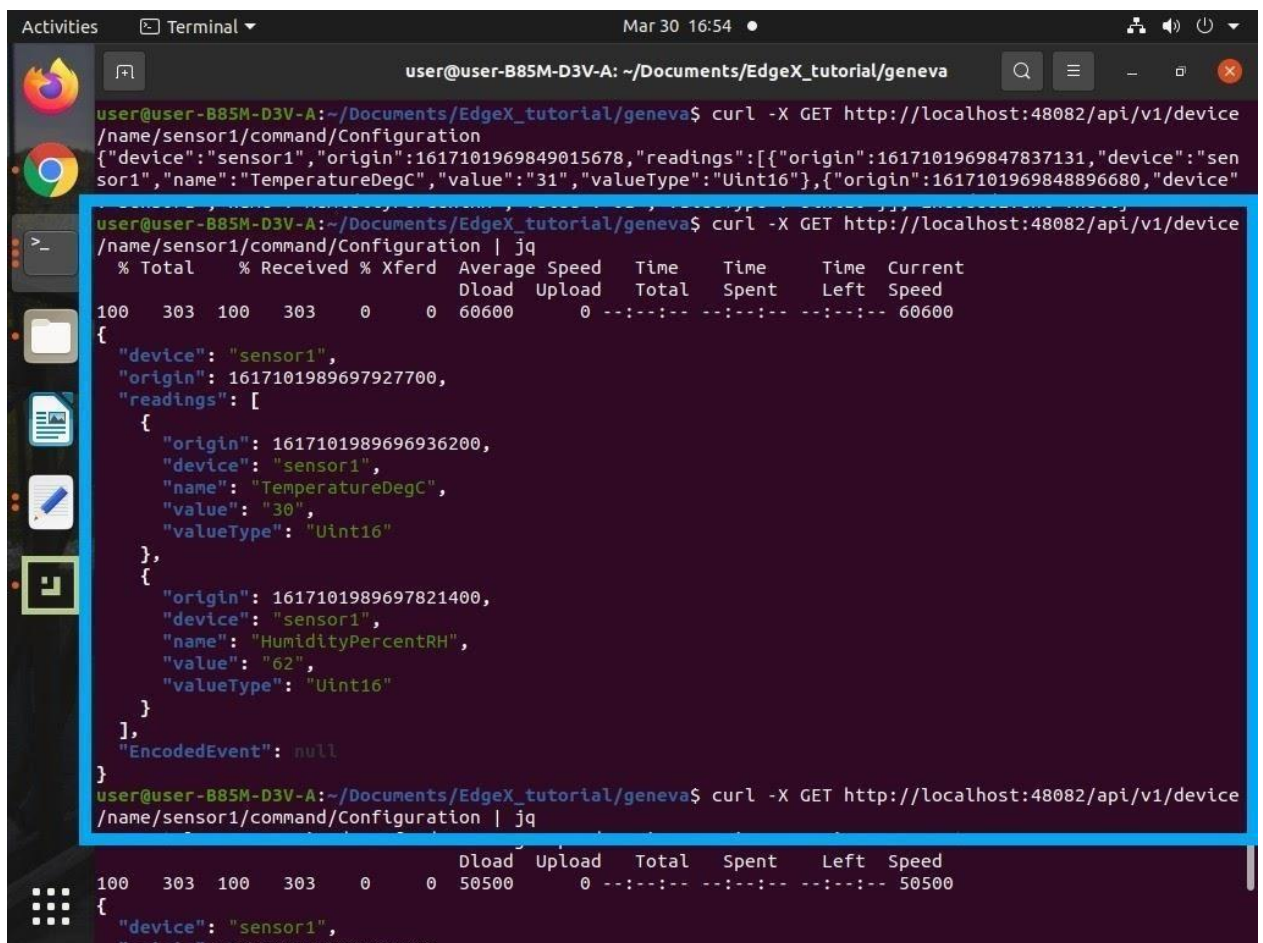
The red section indicates the POST command used to create the device and its configurations. The id has also been generated successfully confirming that the action was successful.

4. View sensor data using GET command

The sensor data can be viewed by using the following command,

curl -X GET

<http://localhost:48082/api/v1/device/name/sensor1/command/Configuration>



A terminal window titled 'user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva' shows the execution of a curl command. The command is: `curl -X GET http://localhost:48082/api/v1/device/name/sensor1/command/Configuration`. The output is a JSON object representing sensor configuration. A blue rectangular highlight is drawn around the JSON output, indicating successful data retrieval. The terminal also shows a progress bar for the curl command, indicating 100% completion.

```
user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$ curl -X GET http://localhost:48082/api/v1/device/name/sensor1/command/Configuration
{"device": "sensor1", "origin": 1617101969849015678, "readings": [{"origin": 1617101969847837131, "device": "sensor1", "name": "TemperatureDegC", "value": "31", "valueType": "Uint16"}, {"origin": 1617101969848896680, "device": "sensor1", "name": "HumidityPercentRH", "value": "62", "valueType": "Uint16"}], "EncodedEvent": null}

user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$ curl -X GET http://localhost:48082/api/v1/device/name/sensor1/command/Configuration | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100    303  100    303    0     0    60600      0 --:--:-- --:--:-- --:--:-- 60600
{
  "device": "sensor1",
  "origin": 1617101989697927700,
  "readings": [
    {
      "origin": 1617101989696936200,
      "device": "sensor1",
      "name": "TemperatureDegC",
      "value": "30",
      "valueType": "Uint16"
    },
    {
      "origin": 1617101989697821400,
      "device": "sensor1",
      "name": "HumidityPercentRH",
      "value": "62",
      "valueType": "Uint16"
    }
  ],
  "EncodedEvent": null
}

user@user-B85M-D3V-A: ~/Documents/EdgeX_tutorial/geneva$ curl -X GET http://localhost:48082/api/v1/device/name/sensor1/command/Configuration | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100    303  100    303    0     0    50500      0 --:--:-- --:--:-- --:--:-- 50500
{
  "device": "sensor1",
  "origin": 1617101989697927700,
  "readings": [
    {
      "origin": 1617101989696936200,
      "device": "sensor1",
      "name": "TemperatureDegC",
      "value": "30",
      "valueType": "Uint16"
    },
    {
      "origin": 1617101989697821400,
      "device": "sensor1",
      "name": "HumidityPercentRH",
      "value": "62",
      "valueType": "Uint16"
    }
  ],
  "EncodedEvent": null
}
```

5.3d Output Data

The blue section indicates the sensor data and connection has been established successfully.

CHAPTER 6

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

6.1 RESULT AND DISCUSSION

The proposed Temperature Monitoring System using EdgeX is reliable and efficient in monitoring the surrounding temperature and displaying the results in a user interface. The major motive is to reduce human effort and that has been achieved successfully.

The UI displays the current temperature and humidity on the first page and this page refreshes every 10 seconds to update the data. The second page is where the user can find historic data with date and time along with a graph plotted with temperature in the x-axis, date and time in the y-axis. The user can navigate through old data by picking the respective data and time to view the data. Thus, an entirely automated temperature monitoring system has been achieved.

6.2 PERFORMANCE ANALYSIS

- The web application is successfully running at “ 192.168.1.2/lab_temp “ and displaying the current temperature and humidity of the surroundings.
- The web page is refreshing for every 10 seconds
- The historic data page is successfully displaying past data for every 10 mins and has successfully generated a graph for better visualization.
- The date picker feature is working successfully and providing the desired result.

- The sensor and device integration is successfully intact and data is sent to edgex.
- EdgeX micro-services are running without any hinderance.
- The data is getting published in server successfully.
- The server is communicating data from both ends successfully.
- The UI is working as expected

CHAPTER 7

SUMMARY AND CONCLUSIONS

7.1 SUMMARY

By keeping the embedded devices in the environment for monitoring enables self protection (i.e., smart environment) to the environment. To implement this need to deploy the sensor devices in the environment for collecting the data and analysis. By deploying sensor devices in the environment, we can bring the environment into real life i.e. it can interact with other objects through the network. Then the collected data and analysis results will be available to the end user through the Wi-Fi. The smart way to monitor environment and an efficient, low cost embedded system is presented with different models in this paper. In the proposed architecture functions of different modules were discussed. The temperature, humidity and CO value can be monitored with Internet of Things (IoT) concept experimentally tested for monitoring three parameters. It also sent the sensor values to the web server and reflected the data on a User Interface. This data will be helpful for future analysis and it can be easily shared to other end users. This model can be further expanded to monitor the developing cities and industrial zones for weather monitoring. To protect the public health from pollution, this model provides an efficient and low cost solution for continuous monitoring of environment.

7.2 CONCLUSION

The aim to minimize human work and maximize automation has been achieved successfully. The data is reflecting on the UI at ease and there is no loss of data. The graphical interface is helpful to perform some analysis on the acquired data. The date picker helps in viewing the past history with an interval of ten minutes between each data. The system is highly scalable through the implementation of EdgeX and makes development and updating work much easier. Through the implementation of this system, the risk of disasters can be highly avoided.

REFERENCES

- [1] Martin Ganchev, Bernhard Kubicek Hansjoerg Kappeler, "Rotor Temperature Monitoring System", IEEE, 11614874, 2010.
- [2] A.F. Fernandez, P. Rodeghiero, B. Brichard, F. Berghmans, A.H. Hartog, P. Hughes, K. Williams, A.P. Leach, "Temperature Monitoring System for Large Nuclear Infrastructures", IEEE, 8767982, 2005.
- [3] Ananya Roy, Prodipto Das, Rajib Das, "Temperature and Humidity Monitoring System for Storage Rooms of Industries", IEEE, 17595765, 2015
- [4] Saraswati Saha, Anupam Majumdar, "Data centre Temperature Monitoring with ESP8266 based Wireless Sensor Network and Cloud-based Dashboard with the Real-time Alert System", IEEE, 17279784, 2017.
- [5] T Tamura, T Togawa and M Murata, "Patient's Bed Temperature Monitoring System for Assessing Body Movement During Sleep", IOP Publishing Ltd, 2006.
- [6] Vongsagon Boonsawat, Jurarat Ekchamanonta, Kulwadee Bumrunghet, and Somsak Kittipiyakul, "XBee Wireless Sensor Networks for Temperature Monitoring", 2015.
- [7] Hasmah Mansor, Muhammad Helmy Abdul Shukor, Siti Sarah Meskam, Nur Quraissyia Aqilah Mohd Rusli, "Body Temperature Measurement for Remote Health Monitoring System", IEEE, 14046824, 2014.
- [8] Verena Raab, Brigitte Petersen, Judith Kreyenschmidt, "Temperature monitoring in meat supply chains", British Food Journal, 0007-070X, 2011.
- [9] Wen-TsaiSung, Yao-ChiHsu, "Designing an industrial real-time measurement and monitoring system based on embedded system and ZigBee", ELSEVIER, Volume 38, Issue 24, 2011.
- [10] Marc Nikles, Bernhard H. Vogel, Fabien Briffod, "Leakage detection using fibre optics distributed temperature monitoring", SPIE, 2004.
- [11] Akash, Amit Birwal, "IoT-based Temperature and Humidity Monitoring System for Agriculture", IJRSET, Volume 6, Issue 7, 2017
- [12] Aman Jatian, "Temperature Monitoring in Home Automation System using Predictive Analytics & IoT" IJART, Volume 7, Issue 5, 2019.
- [13] Ming L Wang, Yang Zhao, Jin-suk Yim, Yinghong Cao, "Temperature effects on the cable-stayed bridge using a health monitoring system", SAGE, Volume 10, Issue 5, 2011.

APPENDIX

A. JOURNAL PAPER

Temperature Monitoring System using EdgeX

by Temperature Monitoring System Using Edgex Temperature Monitoring
System Using Edgex

Submission date: 09-Mar-2021 04:25AM (UTC-0800)

Submission ID: 1528324535

File name: Temperature_Monitoring_System_using_EdgeX.pdf (717.73K)

Word count: 2552

Character count: 13734

Temperature Monitoring System using EdgeX

S.Lakshmi Narayanan¹, Dr.S.Vigneshwari²

B.E Student, ¹Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai, India¹

HOD, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai, India²

Abstract: Temperature Monitoring System, as the name suggests is a Full-Stack IoT application that implements an automated approach towards monitoring temperature. The rise in temperature is a major concern to our ecosystem due to global warming and many man-made factors. We all are aware of the forest fires in the Amazon rainforest and the massive blast at Lebanon in 2020 and the cause behind them. The goal of this project is to minimize the chances of such massacres from happening. The proposed project is "Temperature Monitoring System Using EdgeX" an Industrial Automation project which utilizes edge computing and IoT to analyze the data and act upon the situation to avoid such massacres. This project emphasizes more on how EdgeX supports and increases the scale of an IoT application/system. Using EdgeX the required temperature data is read from a sensor, the required actions on the data are performed and the data is pushed on to a database followed by a web server, and the data is then visualized through a user interface.

Keywords: IoT, EdgeX, Temperature Monitoring System, DHT11, sensors, python, UI.

1.INTRODUCTION

EdgeX Foundry is a latest technology that incorporates both Edge Computing and Internet of Things into a single platform making life much easier for IoT developers. Temperature Monitoring System can be implemented in a variety of use cases ranging from industrial temperature monitoring to household temperature monitoring. Taking care of abnormal rise

in temperature manually is a very hectic and time-consuming task, hence the aim of this project is to build an automated temperature monitoring system. This project focuses on industrial temperature monitoring, where a 24/7 automated temperature monitoring system functions, pushes the data into the database followed by a web server, view the real-time data using a user interface and also alert the required when there are anomalies/threshold breaches in the data thus reduces human work/monitoring to a high extent. The key feature of this project is the usage of EdgeX as it is highly scalable and secure. The UI and alerting system are the user side enhancements to solve the purpose of this project. This project is focused on solving fire accidents and a sudden rise or drop in temperature in various industries. Thus, producing an economically feasible and functionally scalable model to monitor the temperature in industries.

2.MOTIVATION

Massacres caused by fire have started to increase either because of the rise in temperature on our planet or because of man-made effects. Industrial fire and explosion data research shows that over 15 Lakh cases were registered under industrial fires and explosions throughout the world from 2018 to 2020, where most of the massacres were caused majorly because of human error or man-made effects. From time to time, the world and the people living on it have started to update their living and upgrade their existence towards having a better and safe life. Thereby, awareness towards such massacres has also increased among the industries and to also avoid such situations from happening and repeating henceforth.

This project has been established to rule out human error and avoid such massacres from getting repeated.

3.LITERATURE SURVEY

The temperature monitoring system is not that rare in the industry and has various implementations and projects developed on that but in this project, the integration of EdgeX and the full-stack system makes it much more efficient, reliable, scalable and independent.

[3] is the most closely related ⁵existing system when compared with the proposed system. It monitors the temperature of the storage rooms in industries using an ATmega 328 microcontroller ⁶along with a wifi module and GPRS, the aim of the system is to monitor the temperature regularly and preserve the items in the room for a longer duration.

In [1] the rotor temperature of the flight's engine is monitored to avoid malfunctioning of the engine. A series of 16 sensors are used to achieve the task. An alert is generated to the captain of the flight when the temperature reaches the threshold.

[8] describes how important is ⁷temperature monitoring in meat supply chains and in the food industry. An extensive temperature monitoring is done on the surroundings and on the meat itself and accordingly, vary the temperature of the surroundings to keep the meat fresh as long as possible.

[4] uses an ESP8266 microcontroller with an in-built wifi module and a temperature sensor to monitor the temperature of the surroundings. The data is collected and pushed to the cloud, an alert is triggered when the assigned threshold has been reached. A cloud dashboard is created for real-time monitoring.

In [6] temperature monitoring is concentrated more towards Home Automation. The real-time temperature data of various rooms can be viewed through a UI along with past history. It uses WSN (Wireless Sensor Network) to collect the data and communicate with the cloud and other

microcontrollers. It modulates the temperature by turning off/on the air conditioner whenever required.

4.AIM AND OBJECTIVE

The aim is to build an automated industrial temperature monitoring system that can monitor its surrounding temperature at regular intervals, process the data, collect and store the data in a database, upload the data to a web-server, couple the web-server with a UI and prepare an alerting system.

The objective is to monitor temperature data without any hindrance, process the data rapidly, store the data in a database without breakage and loss of data, and create a sustainable and user-friendly UI.

5.PROPOSED SYSTEM

The "Temperature Monitoring System using EdgeX" is a complete hardware and software integrated full-stack Internet of Things based system to make temperature monitoring in industries a much easier task. The system is divided into three parts, the hardware segment followed by the interface segment and ending at the software segment. The first segment comprises, a raspberry-pi micro-computer which is connected with a DHT11, temperature and humidity monitoring sensor. The raspberry-pi has the EdgeX software running in it, on top of a docker as a docker container/image.

In the interface segment, we have the flask micro web-framework, uWSGI application-server and Nginx web-server running inside the EdgeX. The mentioned software applications are responsible for the communication between the hardware and the UI, they act as a bridge allowing both the ends of the system to communicate with ease. The software segment comprises, a User Interface and an alerting system, which are functioning outside of the EdgeX framework. The UI enhances visualizing the data through graphs and monitoring the data at regular intervals.

The proposed system is much more scalable and secure compared to the other systems/projects due to the implementation of EdgeX. Through EdgeX

updating, the working of the system is much easier and scalable as 100 such systems can be updated at one go without having to modify the code individually on each system. The EdgeX uses a reverse-proxy during the communication of data making it much more secure than the other systems. EdgeX provides software and hardware enhancements to the system through its various micro-services which lie under the skin of its service layers. The outcome of this system is to make temperature monitoring a completely automated process and visualize the monitored data.

6. SOFTWARE/HARDWARE REQUIREMENTS

6.1 HARDWARE REQUIREMENTS

6.1.1 Raspberry-Pi

Raspberry-Pi is a credit card-sized micro-computer, which has all functionalities and also performs all the tasks as that of a computer. It has 1 GB inbuilt RAM and up to 256 GB ROM, USB ports for connecting input devices, HDMI for connecting to a display, a built-in Wifi module and Bluetooth.

6.1.2 DHT11 Sensor

DHT11 is a temperature and humidity detection sensor that can detect temperature from 0-50°C and humidity from 20-90%RH. This one most widely used sensor in industrial temperature monitoring systems as its economical and easy to integrate.

6.2 SOFTWARE REQUIREMENTS

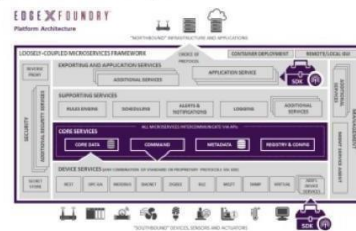
6.2.1 Docker

Docker is a widely used PaaS (Platform as a Service) infrastructure that encapsulates application/software and its dependencies such as library into a single package called a container. Docker allows multiple containers to run on the same OS, containers do not have individual OS's unlike virtual machines but share and utilize the functionalities of an OS through

docker. Docker is majorly used and implemented in the DevOps infrastructure.

6.2.2 EdgeX Foundry

The EdgeX Foundry is a prevalent and common open-source framework for Industrial IoT Edge Computing. It combines IoT and Edge Computing into a single platform, making the development and maintenance of industrial IoT products much easier. It is a loosely coupled four-layered architecture where each layer is considered as a service containing various micro-services assigned to perform numerous tasks. The four layers are the device service layer, core services layer, support services layer and finally application services layer.



6.2.3 Flask

Flask is a micro web framework that uses python scripts to develop web applications. It contains code libraries and reusable extensions for making web application development easier. It is an integral part of the development of a web application using python through the functionalities it provides.

6.2.4 uWSGI

uWSGI (Web Server Gateway Interface) is an application server interface that acts as a communication pipe between the python application and the web-server. It is implemented to provide smooth communication between the two ends.

6.2.5 Nginx

Nginx is a web server that connects and communicates between the HTTP client/web application and the backend software application/

application server. It serves the requirements of the HTTP client and maintains a log/record of the happenings.

HTML, CSS and jQuery were used for the construction of the UI.

7. SYSTEM ARCHITECTURE



1. The DHT11 sensor is connected to the Raspberry-Pi via jumper cables. DHT11 senses the surrounding temperature and humidity also pass that data to Raspberry Pi through the data pin.
2. The Raspberry-Pi being a micro-computer requires an OS to run on it. Raspbian OS runs as the OS on the Raspberry-Pi, it is also the native OS of Raspberry-Pi.
3. The Docker is then installed on top of the OS to work as an interface between the containerized application and the OS. The docker helps the containers to individually access and make use of the features of the OS.
4. The EdgeX Foundry software is downloaded as a container on top of a docker. The data is

accessed, modified and sent to the web-server through EdgeX and its micro-services.

5. The MODBUS micro-service is responsible for the communication of data from and to the sensor. This micro-service lies in the device-service layer.
6. SQLite micro-service is the database management system used in this project. It collects the data from the MODBUS device service and stores the data and represents them in the form of tables, modifications can be performed on the data present in tables using SQLite commands.
7. Flask micro-service is present in the application services layer and is the backbone of the web application. The code for what should happen and when it should happen is present inside the flask framework and is responsible for the communication of data from the database to the application-server (uWSGI).
8. uWSGI micro-service acts as the application-server inside the EdgeX and is present in the application services layer. It is responsible for the peaceful communication of data between the web-server (Nginx) and the flask framework.
9. The Nginx web-server is outside the bounds of the EdgeX software and behaves like an interface between the UI and application-server (uWSGI). It sends data to and receives data from EdgeX.
10. The User Interface is the final endpoint where the entire temperature and humidity can be viewed and analyzed. The data is visually represented with the help of a graph that was generated using the Google Chart API. The UI increases the transparency of the system and delivers the needs of the users. The requests made by the user are sent to the web-server and the desired result is then displayed on the UI.

8. RESULT AND DISCUSSION

The proposed Temperature Monitoring System using EdgeX is reliable and efficient in monitoring the surrounding temperature and displaying the results in a user interface. The major motive is to reduce human effort and that has been achieved successfully.

The UI displays the current temperature and humidity on the first page and this page refreshes every 10 seconds to update the data. The second page is where the user can find historic data with date and time along with a graph plotted with temperature in the x-axis, date and time in the y-axis. The user can navigate through old data by picking the respective data and time to view the data. Thus, an entirely automated temperature monitoring system has been achieved.

9. APPLICATION SCREENSHOTS



Fig.1 Current Data Page

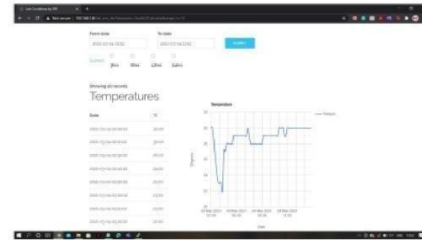


Fig.2 Temperature Data and Graph with the Timestamp

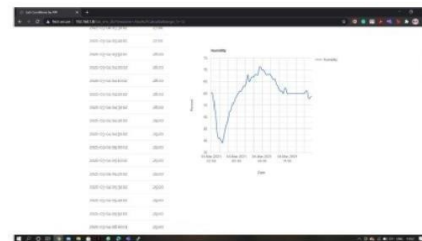


Fig.3 Humidity Data and Graph with the Timestamp

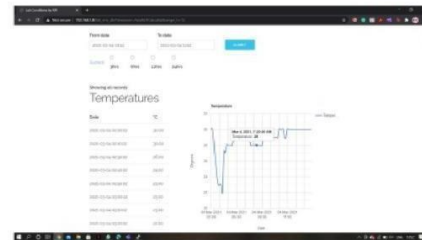


Fig.4 Extensive Graph Data

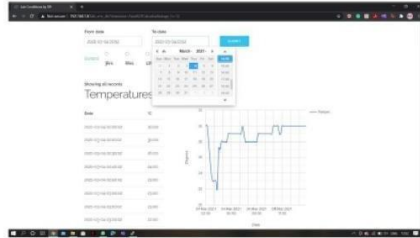


Fig.5 Date Picker

10. Conclusion

The aim to minimize human work and maximize automation has been achieved successfully. The data is reflecting on the UI at ease and there is no loss of data. The graphical interface is helpful to perform some analysis on the acquired data. The date picker helps in viewing the past history with an interval of ten minutes between each data. The system is highly scalable through the implementation of EdgeX and makes development and updating work much easier. Through the implementation of this system, the risk of disasters can be highly avoided.

11. REFERENCES

- [1] Martin Ganchev, Bernhard Kubicek Hansjoerg Kappeler, "Rotor Temperature Monitoring System", IEEE, 11614874, 2010.
- [2] A.F. Fernandez, P. Rodeghiero, B. Brichard, F. Berghmans, A.H. Hartog, P. Hughes, K. Williams, A.P. Leach, "Temperature Monitoring System for Large Nuclear Infrastructures", IEEE, 8767982, 2005.
- [3] Ananya Roy, Prodipto Das, Rajib Das, "Temperature and Humidity Monitoring System for Storage Rooms of Industries", IEEE, 17595765, 2015
- [4] Saraswati Saha, Anupam Majumdar, "Data centre Temperature Monitoring with ESP8266 based Wireless Sensor Network and Cloud-based Dashboard with the Real-time Alert System", IEEE, 17279784, 2017.
- [5] T Tamura, T Togawa and M Murata, "Patient's Bed Temperature Monitoring System for Assessing Body Movement During Sleep", IOP Publishing Ltd, 2006.
- [6] Vongsagon Boonsawat, Jurarat Ekchamanonta, Kulwadee Bumrunghet, and Somsak Kittipiyakul, "XBee Wireless Sensor Networks for Temperature Monitoring", 2015.
- [7] Hasmah Mansor, Muhammad Helmy Abdul Shukor, Siti Sarah Meskam, Nur Quraisyia Aqilah Mohd Rusli, "Body Temperature Measurement for Remote Health Monitoring System", IEEE, 14046824, 2014.
- [8] Verena Raab, Brigitte Petersen, Judith Kreyenschmidt, "Temperature monitoring in meat supply chains", British Food Journal, 0007-070X, 2011.
- [9] Wen-Tsai Sung, Yao-Chi Hsu, "Designing an industrial real-time measurement and monitoring system based on embedded system and ZigBee", ELSEVIER, Volume 38, Issue 24, 2011.
- [10] Marc Nikles, Bernhard H. Vogel, Fabien Briffod, "Leakage detection using fiber optics distributed temperature monitoring", SPIE, 2004.
- [11] Akash, Amit Birwal, "IoT-based Temperature and Humidity Monitoring System for Agriculture", IJIRSET, Volume 6, Issue 7, 2017
- [12] Aman Jatian, "Temperature Monitoring in Home Automation System using Predictive Analytics & IoT" IJART, Volume 7, Issue 5, 2019.
- [13] Ming L Wang, Yang Zhao, Jin-suk Yim, Yinghong Cao, "Temperature effects on the cable-stayed bridge using a health monitoring system", SAGE, Volume 10, Issue 5, 2011.

B. PLAGIARISM REPORT

Temperature Monitoring System using EdgeX

ORIGINALITY REPORT

3%

SIMILARITY INDEX

3%

INTERNET SOURCES

2%

PUBLICATIONS

2%

STUDENT PAPERS

PRIMARY SOURCES

1

www.ijircce.com

Internet Source

1%

2

www.irjet.net

Internet Source

1%

3

"International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018", Springer Science and Business Media LLC, 2019

Publication

<1%

4

"Proceeding of International Conference on Intelligent Communication, Control and Devices", Springer Science and Business Media LLC, 2017

Publication

<1%

5

mafiadoc.com

Internet Source

<1%

6

www.researchscript.com

Internet Source

<1%

7

www.uni-bonn.de

Internet Source

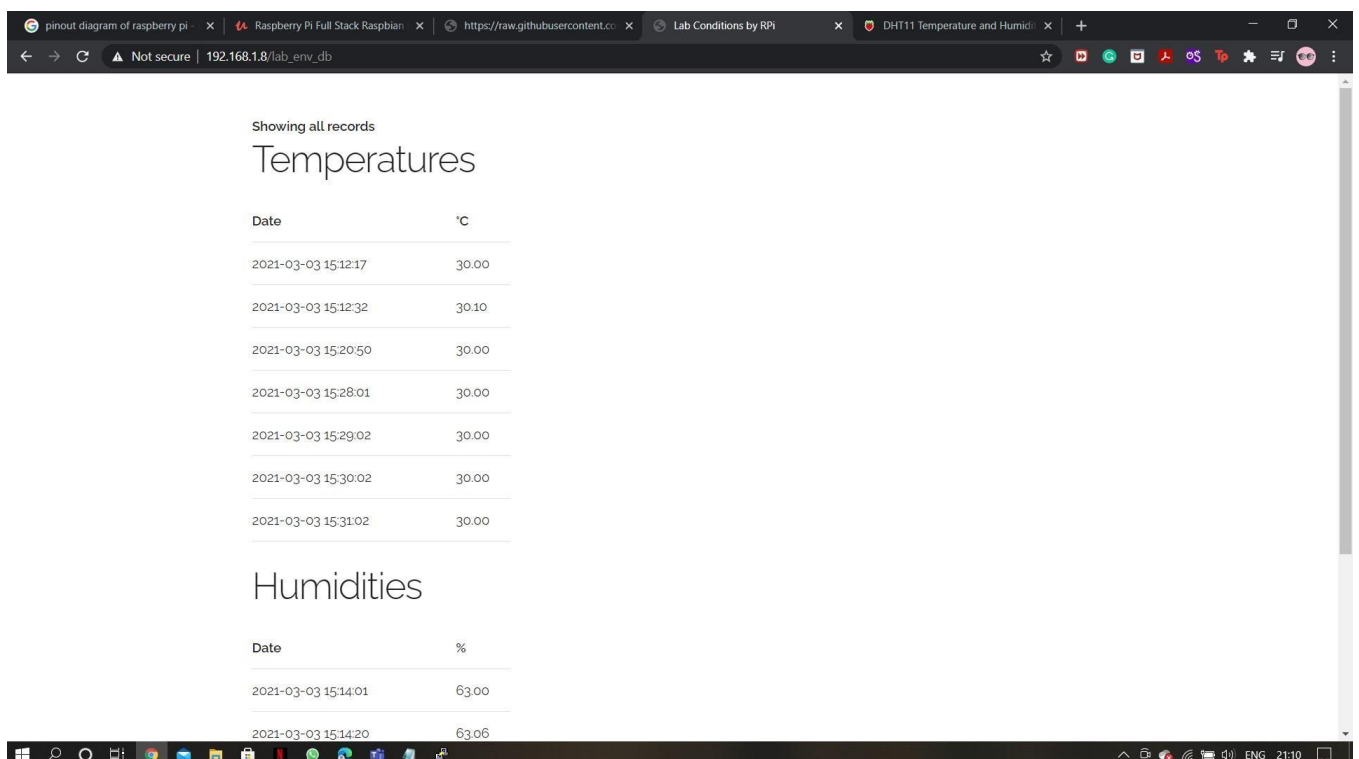
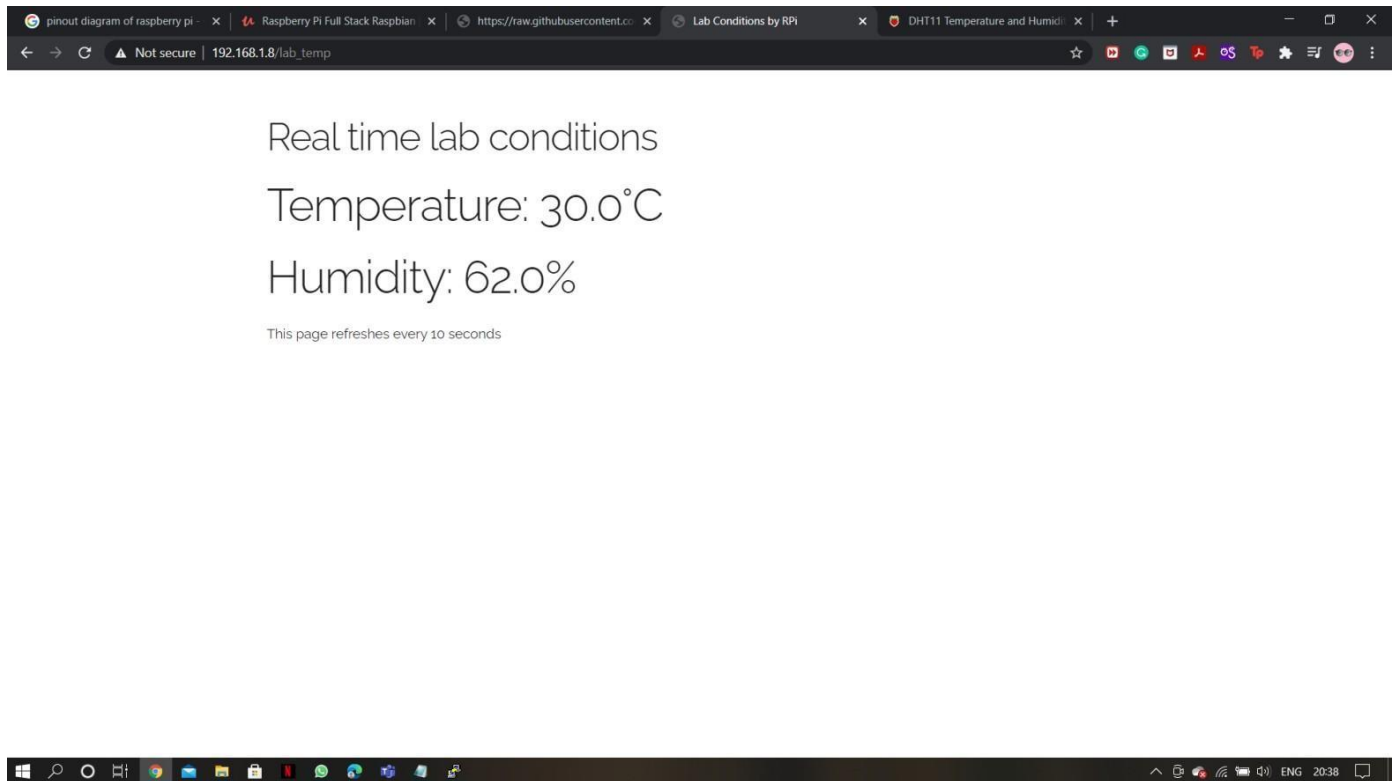
<1%

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off

C. SCREENSHOTS



pinout diagram of raspberry pi x Raspberry Pi Full Stack Raspbian x https://raw.githubusercontent.com x Lab Conditions by RPi x DHT11 Temperature and Humidity x +

Not secure | 192.168.1.8/lab_env_db

2021-03-03 15:20:50	30.00
2021-03-03 15:28:01	30.00
2021-03-03 15:29:02	30.00
2021-03-03 15:30:02	30.00
2021-03-03 15:31:02	30.00

Humidities

Date	%
2021-03-03 15:14:01	63.00
2021-03-03 15:14:20	63.06
2021-03-03 15:20:50	64.00
2021-03-03 15:28:01	65.00
2021-03-03 15:29:02	65.00
2021-03-03 15:30:02	65.00
2021-03-03 15:31:02	65.00

Windows taskbar: 21:10

pinout diagram of raspberry pi x Raspberry Pi Full Stack Raspbian x RaspberryPiFullStack_Raspbian/ x Lab Conditions by RPi x DHT11 Temperature and Humidity x +

Not secure | 192.168.1.8/lab_env_db?range_h=3

3hrs 6hrs 12hrs 24hrs

Showing all records

Temperatures

Date	°C
2021-03-03 15:12:17	30.00
2021-03-03 15:12:32	30.10
2021-03-03 15:20:50	30.00
2021-03-03 15:28:01	30.00
2021-03-03 15:29:02	30.00
2021-03-03 15:30:02	30.00
2021-03-03 15:31:02	30.00
2021-03-03 15:40:01	30.00
2021-03-03 15:50:01	30.00
2021-03-03 16:00:02	30.00

Windows taskbar: 22:52

