
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

# Ανάπτυξη συσκευής IOT για την παρακολούθηση και εξοικονόμηση ενέργειας

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΥΡΙΑΚΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΤΖΙΑΛΛΑΣ ΓΡΗΓΟΡΙΟΣ
Καθηγητής

Λαμία ..... έτος .....

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

# Ανάπτυξη συσκευής IOT για την παρακολούθηση και εξοικονόμηση ενέργειας

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΥΡΙΑΚΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΤΖΙΑΛΛΑΣ ΓΡΗΓΟΡΙΟΣ
Καθηγητής

Λαμία ..... έτος .....


UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

# Development of an IOT device for monitoring and energy saving

KONSTANTINOS KYRIAKOS

FINAL THESIS

ADVISOR

TZIALLAS GRIGORIOS
Professor.

Lamia ..... year .....

*«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:*

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

*3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

*4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

*(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση*
*του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων*
*σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»*

# ΠΕΡΙΛΗΨΗ

Σκοπός αυτής της πτυχιακής εργασίας είναι η μελέτη και υλοποίηση μιας συσκευής IOT για την παρακολούθηση και εξοικονόμηση ενέργειας. Θα γίνει μελέτη πάνω στις τεχνολογίες «Διαδίκτυού των Πραγμάτων» (Internet of Things) οπού θα παρουσιαστούν συσκευές IoT καθώς και τα πρωτόκολλά επικοινωνίας που χρησιμοποιούν με τον σκοπό ανάπτυξη μιας εφαρμογής για την εξοικονόμηση ενέργειας. Συγκεκριμένα θα χρησιμοποιηθούν μικροελεγκτες τύπου ESP32 οπού θα συλλέγουν δεδομένα θερμοκρασίας και θα τα προωθούν σε ένα σύστημα συλλογής, αποθήκευσης και οπτικοποίησης δεδομένων.

# ABSTRACT

The purpose of this thesis is the research and development of an IOT device for monitoring and energy saving. Research will be done on «Internet of Thins» (IoT) technologies, were IoT devices and the protocols they use will be showcased with the goal of creating an application for energy saving. In detail, ESP32 type microcontrollers will be used for collecting and transmitting temperature data to a system for data collection, storage and visualization.

Table of Contents

# Chapter 1 Introduction

## 1.1 Introduction

The purpose and goals of this thesis is to research, evaluate and test an IoT System used for energy saving and temperature monitoring. The research for the development of such system will include various protocols used by IoT devices, IoT devices commonly used for IoT network applications along with their necessary sensing equipment and an IoT framework that was developed using JavaScript, Java, the C++ based language used by the Arduino platform, a PostgreSQL database for data storage, a Grafana server for data visualization and Docker for the deployment of the systems sub-components.

# Chapter 2 Introduction to IoT Concepts

## 2.1 Introduction to IoT

### 2.1.1 Introduction

The term IoT (Internet of Things) refers to a network of physical devices, such as sensors, instruments, cameras, vehicles, wearables and other devices equipped with the necessary hardware and software that allows them to communicate and exchange data. An IoT environment allows for the remote sensing, monitoring and control of devices over an existing network infrastructure with the purpose of better integration of such devices with a computer-based environment resulting in greater efficiency in their specific field of application.

The term “Internet of Things” (IoT) was first used in 1999 by British technology pioneer Kevin Ashton to describe a system in which objects in the physical world could be connected to the Internet by sensors. Ashton coined the term to illustrate the power of connecting Radio-Frequency Identification (RFID) tags used in corporate supply chains to the Internet in order to count and track goods without the need for human intervention. Today, the Internet of Things has become a popular term for describing scenarios in which Internet connectivity and computing capability extend to a variety of objects, devices, sensors, and everyday items [1].

### 2.1.2 Timeline of IoT

- **1990**: A “toaster” that could be turned ON and OFF from the internet was showcased by John Romkey in the INTEROP ’89 conference. The toaster used TCP/IP and is considered to be the first IoT device.

- **1993**: Created by Quentin Stafford-Fraser and Paul Jardetzky the Trojan Room Coffee Pot was located in the 'Trojan Room' within the Computer Laboratory of the University of Cambridge and was used to monitor the pot levels with an image being updated about 3x a minute and sent to the buildings server.

- **1999**: The term Internet of Things is coined by Kevin Ashton, Executive Director of the Auto-ID Center in Massachute Institute of Technology. (

- **2000**: LG announces plans to develop the first refrigerators with IoT capabilities.

- **2003**-**2004**: RFID is deployed on a massive scale by the US Department of Defense in and Wal-Mart in the commercial world.

- **2008**: Recognition by the EU and the First European IoT conference is held.

- **2011**: IPV6 public launch. The new protocol allows for $2^{128}$ possible addresses.

- **Today**: IoT has become mainstream, being used daily in home and commercial applications.

**Figure 1 The “First” IoT Device pictured in INTEROP 89.**

## 2.2 IoT Models

IoT Devices and Systems follow different models of communication depending the devices type and field of use. Those models include the Device to Device, Device to Gateway and the Device to Cloud models.

### 2.2.1 Device to Device IoT

The Device-to-Device (D2D) IoT model refers to a direct communication between two or more devices in an IoT network without the need for intermediaries such as a centralized server or the internet. This communication is established using either Wi-Fi, Bluetooth, or NFC protocols, allowing the devices to communicate with each other directly and securely. The D2D model is particularly useful in scenarios where a centralized server is not available or where data privacy and security are critical. In this model, devices can share information, collaborate on tasks, and exchange data, making the IoT network more resilient, efficient, and autonomous. D2D communication also reduces the burden on the central server, conserves network resources, and improves response times, making the overall system more scalable and responsive [2].

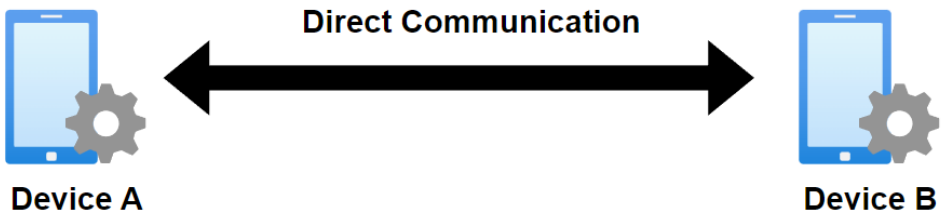



**Figure 2 D2D model**

### 2.2.2 Device to Cloud IoT

The Device-to-Cloud (D2C) IoT model refers to a communication architecture in which IoT devices communicate with a centralized server, known as the "cloud," to transmit data and receive commands. In this model, IoT devices are equipped with sensors, actuators, and other components that collect data and send it to the cloud for storage, processing, and analysis. The cloud acts as a central repository of data and provides a platform for data analytics and decision-making. This centralized approach enables data to be easily accessible and shared across multiple devices, systems, and applications. The D2C model also provides a single point of control and management, allowing administrators to monitor and manage large networks of IoT devices from a single location. This model is commonly used in applications where a large amount of data needs to be processed and analysed, such as in the fields of healthcare, industrial automation, and smart homes [25].

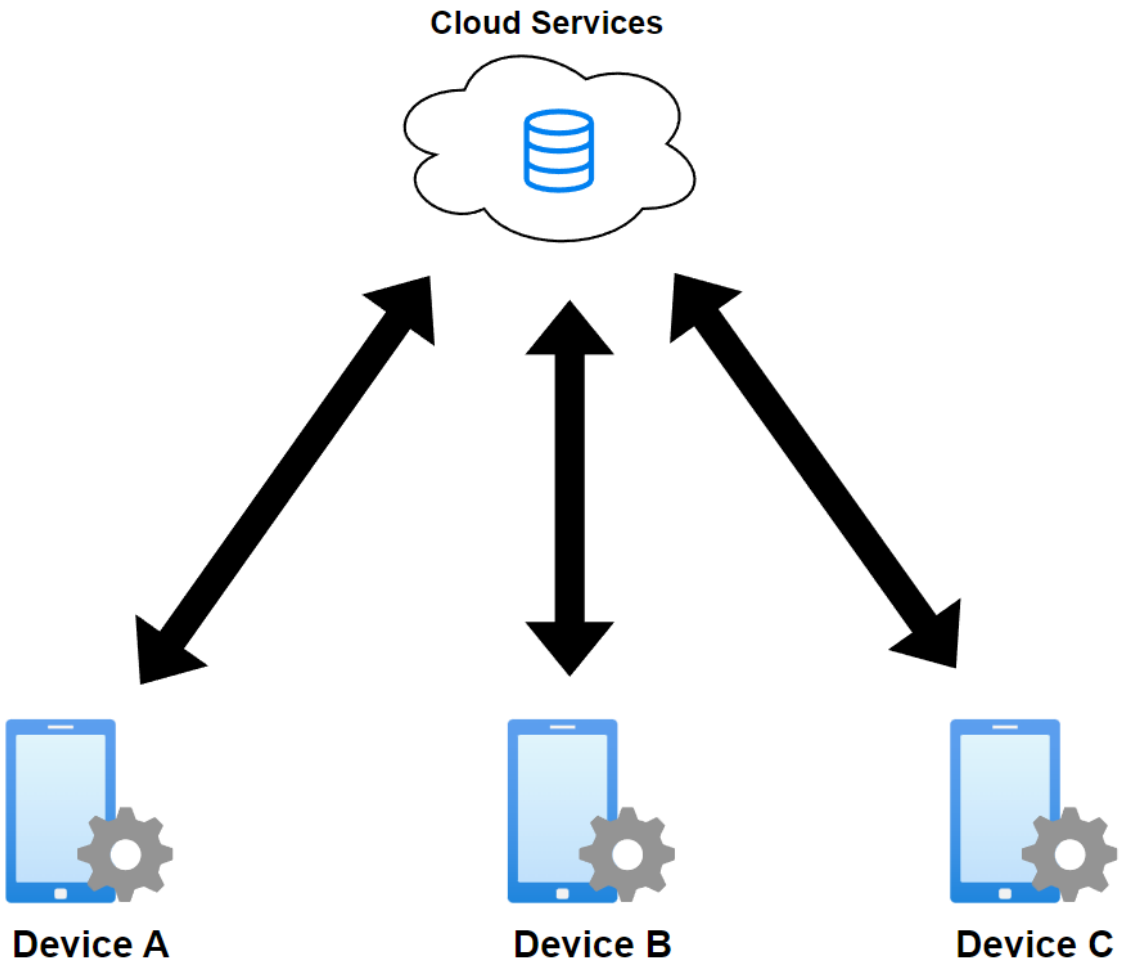



**Figure 3 D2C model.**

### 2.2.3 Device to Gateway IoT

The Device-to-Gateway (D2G) IoT model refers to a communication architecture in which IoT devices communicate with a local gateway, which acts as a bridge between the devices and the cloud. In this model, IoT devices send data to the gateway, which then forwards the data to the cloud for processing, analysis, and storage. The gateway is responsible for managing the network of devices, handling security, and ensuring the data is transmitted in a reliable and secure manner. The D2G model is beneficial for applications where the IoT devices are located in remote or inaccessible areas, where a direct connection to the cloud is not possible or feasible. The gateway acts as an intermediary, providing a secure connection to the cloud and enabling the devices to communicate even if they are located far from the cloud. The D2G model also provides a local point of control, reducing the amount of data that needs to be transmitted over the network, and conserving network resources. This model is commonly used in industrial and commercial applications, such as remote monitoring and control of assets in the field.

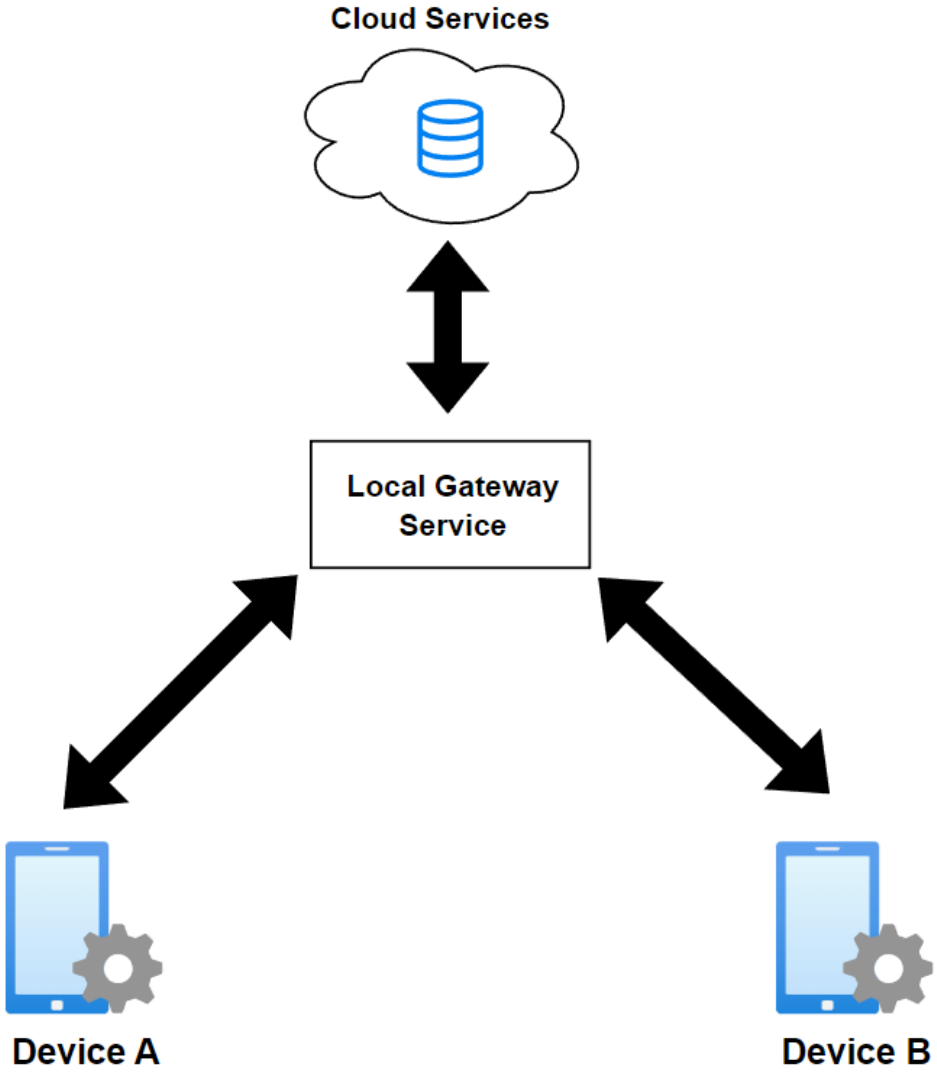



**Figure 4 D2G model.**

## 2.3 IoT Systems and Applications

### 2.3.1 Home IoT

In recent years IoT has seen widespread use in home applications or as it is more commonly referred to as smart homes. IoT has enabled the development of smart homes, which are homes equipped with a network of connected devices that can be controlled remotely and automatically. IoT devices such as smart thermostats, lighting systems, and home security systems can be programmed to adjust their settings based on user preferences or environmental conditions. Smart home devices can be controlled via smartphones or voice-activated assistants, making it easy for users to manage their homes' systems from anywhere.

IoT sensors can be used to detect changes in temperature, humidity, and air quality, allowing smart home systems to make adjustments for maximum comfort and energy efficiency. With the ability to monitor and control their homes from anywhere, smart home owners can reduce their energy bills, enhance their security, and enjoy a more convenient and comfortable living experience.

### 2.3.2 IoT in business and Industry

Apart from home use IoT has also seen widespread use in the commercial and Industrial sectors. The Internet of Things has transformed the way businesses operate by providing real-time insights and enabling smarter decision-making. IoT devices such as sensors, cameras, and tracking systems can be used to monitor assets, processes, and supply chains, providing valuable data on performance, efficiency, and quality. IoT data can be analysed to identify patterns and trends, enabling businesses to make data-driven decisions and optimize their operations. IoT devices can also be used to improve customer experiences, such as through the use of smart shopping carts [3] or personalized offers based on customer data. IoT applications can also be used to enhance workplace safety and productivity, such as through the use of stationary or wearable sensors to monitor employee health and safety in hazardous or demanding working environments.

As companies and organizations continue to aim for a reduced carbon footprint in order to combat global warming, but also seek for reduced energy and resource usage can rely on IoT applications to achieve those goals. Smart thermostats, lighting systems, and appliances can be programmed to adjust their settings based on user preferences or environmental conditions, reducing energy waste and lowering utility bills. IoT systems can also be used to optimize the use of renewable energy sources, such as solar or wind power, by managing energy production and storage. IoT-enabled smart grids can help utilities better manage energy distribution and prevent power outages. In the following chapters we will see in detail the creation of energy monitoring and conservation system using IoT sensors for pilot use in the premises of the Department of Informatics and Telecommunications of University of Thessaly.

### 2.3.3 IoT in Healthcare

The Internet of Things has a wide range of potential applications in the field of medicine. IoT medical applications can be used to improve patient outcomes, enhance the quality of care, and reduce healthcare costs. IoT medical devices such as wearable sensors, medical imaging equipment, and remote monitoring systems can provide real-time data to healthcare providers, allowing them to monitor patient health and intervene as necessary. IoT devices can also be used to collect data on patient behaviour, such as sleep patterns, exercise habits, and diet, providing valuable insights into patient health and helping to identify potential health risks. IoT medical applications can also be used to improve medication adherence, reducing the risk of adverse drug interactions and improving patient outcomes [4].

### 2.3.4 Wearables

A large sub-group of IoT enabled devices include the wearables devices. Wearable devices are electronic devices that are designed to be worn on the body, either as an accessory or as an integrated part of clothing or accessories. These devices typically have sensors that can collect data about the wearer, such as heart rate, steps taken, and sleep patterns. Wearable devices can come in various forms, including smartwatches, fitness trackers, smart glasses, and smart clothing. They can be used for a range of purposes, such as fitness tracking, health monitoring, navigation, and communication. IoT enabled wearable devices that are either connected to the cloud or to a local network, enable them to receive notifications and updates and allowing users to access information and services remotely. The data collected by wearable devices can be analysed to provide insights into the wearer's behaviour and health, enabling them to make more informed decisions about their lifestyle and health choices. Some examples of popular IoT enabled wearable devices include:

- Fitbit, a fitness tracker that is able to read, store and analyse the wearers health status and behaviour.

- Microsoft's HoloLens, an AR (Augmented Reality) headset designed to enhance the wearers experience by overlaying virtual object and information in the real world.

**Figure 6 Fitbit tracker.**

**Figure 5 Microsoft HoloLens**

### 2.3.5 Robots and UxVs

IoT enabled robots can enhance and expand their capabilities but also create new advanced autonomous robotic systems. IoT technology can provide robots with the ability to connect to other devices and systems, enabling them to gather and exchange data with other machines and humans. This data can include information such as environmental data, location data and machine-to-machine communication. With IoT, robots can be remotely controlled, monitored, and configured, making them more versatile and adaptable to different environments and situations. IoT technology can also provide robots with enhanced sensing and perception capabilities. With the ability to collect and analyse real-time data from sensors and cameras, robots can detect and respond to changes in their environment and adapt their behaviour accordingly. For example, robots equipped with IoT sensors and cameras can detect changes in temperature, humidity, or air quality, and adjust their operations accordingly to optimize energy usage or to avoid potential safety hazards. In addition, IoT can provide robots with access to cloud-based data analytics and machine learning, enabling them to learn from past experiences and improve their performance over time. This can lead to the development of more sophisticated and intelligent robotic systems that can perform a wide range of tasks in various industries, such as manufacturing, healthcare, transportation. Developments in wireless communications and the adoption of 5G have greatly impacted the field of unmanned autonomous vehicles. Such vehicles equipped with sensors and high bandwidth antennas can perform advanced operations on their own or in direct cooperation with other vehicles (Swarming) and be effective in various field and domains, such as in industry, agriculture and surveillance [5].


**Figure 7 5G enabled UgV (Unmanned Ground Vehicle)**

# Chapter 3 IoT Devices

In this chapter we will analyse the characteristics of IoT devices. We will see the different types of IoT devices, their key functions and components and we will explore and compare the available IoT development platforms.

## 3.1 Characteristics

As it was described in the previous chapters IoT devices, their characteristics and form are dependent on their field of operation, in order to maximize their usability and effectiveness. While this is true, IoT devices share some common key characteristics and functions.

- **Connectivity:** IoT devices are designed to be connected to the internet or other networks, allowing them to exchange data with other devices and systems.

- **Sensors or ability to interact with the environment**: IoT devices are equipped with sensors that allow them to collect data from the physical world and/or interact with it with the use of mechanical actuators or electrical relays.

- **Low power consumption**: Many IoT devices are designed to operate on low power, as they may need to run on batteries or other limited power sources. This allows them to be deployed in remote or hard-to-reach locations without requiring frequent maintenance.

- **Data processing and analytics**: IoT devices often have some level of data processing and analytics capabilities, allowing them to analyse and interpret the data they collect, and take action based on that data.

- **Integration with other systems**: IoT devices are often designed to integrate with other systems, such as cloud-based platforms or other IoT devices, allowing them to exchange data and work together to perform more complex tasks.

## 3.2 Edge Devices

IoT edge devices are a type of device that performs data processing and analysis at the edge of the network, rather than in the cloud or a central data center. These devices are often connected to sensors or other devices that collect data from the physical world, such as temperature, humidity, or motion sensors. Rather than transmitting all of this data to a cloud-based system for analysis, IoT edge devices can perform some or all of the data processing and analysis locally, before transmitting the results to the cloud or to other networks.

One of the key benefits of IoT edge devices is that they can reduce latency and bandwidth requirements by processing data locally, rather than transmitting it to a remote location for processing. This can be particularly important in applications where real-time response is critical, such as industrial automation or autonomous vehicles. Another benefit of IoT edge devices is that they can improve security and privacy by reducing the amount of data that needs to be transmitted over the network. By performing data processing and analysis locally, IoT edge devices can minimize the amount of sensitive data that needs to be transmitted to a central data center, reducing the security risk.

## 3.3 IoT Hardware

An IoT device in order to be able to process data, connect to networks, and to interface with sensing equipment needs the necessary hardware that can accommodate the devices needs. IoT devices are often build around microprocessors or microcontrollers with emphasis given to their power consumption.

### 3.3.1 Microprocessors

A Microprocessor is the controlling unit of a microcomputer encased inside a computer chip. A microprocessor can perform Arithmetic Logic Unit (ALU) operations and is able to communicate with the other devices and components of the microcomputer. A microprocessor does not contain Random-Access-Memory (RAM), Read-Only-Memory (ROM) or Input-Output units (IO), components that are necessary for the functionality of a micro-computer and needs to interface with those components externally.

In IoT and embedded applications microprocessors are often used when there is need for local processing power e.g., on Edge devices. Some key requirements for processors used in IoT devices are that the processors must be low-energy and low-heat, as well as to require minimal resources from the rest of the device's components. Some microprocessor architectures used often for IoT devices and applications include:

- **ARM**: ARM architecture is a type of computer processor architecture that is used in a wide range of devices, including smartphones, tablets, laptops, and embedded systems. The architecture was developed by ARM Holdings, a British semiconductor company, and is based on a reduced instruction set computing (RISC) architecture.

- **RISC-V**: RISC-V architecture is an open-source, instruction set architecture (ISA) that is designed for computer processors. It is based on the RISC (Reduced Instruction Set Computing) design philosophy, which aims to simplify the instructions and operations that a processor can perform in order to improve its efficiency and speed.

- **Intel Atom**: The Intel Atom architecture is a low-power, x86-based microprocessor architecture that is designed for use in mobile devices, embedded systems, and other low-power applications. The Atom architecture was first introduced in 2008, and has since been used in a wide range of devices, including tablets, smartphones and embedded devices.

### 3.3.2 Microcontrollers

A Microcontroller or MCU (Microcontroller Unit) is a chip optimized for the control of electronic devices. Unlike a general-purpose microprocessor, a microcontroller typically contains all of the components needed to control an embedded system, including the microprocessor itself, memory (RAM), input/output interfaces (IO), and other specialized hardware components such as timers, analog-to-digital converters, and communication interfaces.

Microcontrollers are often used in devices that require real-time control or data processing, such as automotive systems, medical devices, and industrial control systems. They are designed to be small, low-power, and highly efficient, making them well-suited for battery-powered or other low-power applications.

## 3.4 Sensors and other Components

IoT devices are often equipped with different kinds of sensors and interfaces that enable them to collect data from the physical world, in order for the data to be processed, analysed, distributed to other IoT devices or network nodes and stored to the cloud. The sensing equipment of IoT devices is dependent to their field of application and specialized for the specific use case.

Apart from sensors IoT devices are equipped with other components and devices that are critical to their operation or enable them to perform extra functionalities. Communication modules, such as Wi-Fi, Bluetooth, or cellular, are used to transmit the collected data to other devices or to the cloud for storage and analysis. Power sources, such as batteries or AC adapters, are used to provide power to the device, while user interfaces, such as displays or buttons, are used to interact with the device. IoT devices may also be equipped with mechanical actuators or other devices that enable them to interact with the physical world, such as in the case of robotics and autonomous vehicles.

### 1.4.1 Types of Sensors

- **Temperature sensors**: Temperature sensors are sensors that can measure the heat generated by an object or area. They can detect temperature fluctuations and convert their measurements to data. Temperature sensors are utilized in various fields and industries, including manufacturing, healthcare, science and for energy conservation. Types of temperature sensors, include thermocouples, resistance temperature detectors (RTDs), thermistors, and infrared sensors

- **Humidity sensors:** A humidity sensor is a devise that count the molecules of water vapor present in the air or other gases. HVAC (heating, ventilation, and air conditioning) systems, which are used in both commercial and residential settings, frequently include humidity sensors. In addition to hospitals, they are also used in meteorology stations to report and forecast weather.

- **Pressure sensors:** A pressure sensor is a sensor that is used to measure the pressure of a gas or liquid in the surrounding environment. Pressure sensors are commonly used in industrial control systems, automotive applications, and medical devices. The different types of pressure sensors include piezoresistive, capacitive, and optical sensors.

- **Proximity sensors:** Proximity sensors are devices used to detect the presence or absence of objects in the surrounding environment. Proximity sensors are commonly used in manufacturing and industrial control systems, as well as in consumer electronics and automotive applications. There are several types of proximity sensors, including inductive, ultrasonic, and optical sensors.

- **Water Quality sensors:** A water quality sensor is a device that is used to measure the quality of water in terms of various physical and chemical parameters. Water quality sensors are commonly used in environmental monitoring, agriculture, aquaculture, and industrial applications. There are several types of water quality sensors, including conductivity sensors, pH sensors, dissolved oxygen sensors, turbidity sensors, and temperature sensors.

- **Chemical and Gas sensors:** Chemical and Gas sensors are devices used to measure and detect chemical substances in a sample. Chemical and Gas sensors are commonly used in laboratory applications, healthcare, industrial safety, manufacturing and in food and beverage quality control.

- **Motion sensors:** A motion sensor is a type of sensor that detects movement or changes in position of objects within its field of view. Motion sensors are widely used security systems, lighting systems, and home automation systems. Motion sensor types include passive infrared (PIR) sensors, ultrasonic sensors, microwave sensors, and vision-based sensors.

- **Image sensors:** Image sensors are used to convert optical images to digital data or signals to be processed, stored or analysed. Images sensors cover a wide spectrum of devices such as digital cameras, IR cameras, Radars and Lidars. Image sensors are often

used in industrial automation, security applications, healthcare and in autonomous vehicle navigation.

- **Accelerometers:** An accelerometer is a device that is used to measure acceleration, including gravitational acceleration and changes in motion. Accelerometers are commonly used in mobile devices, wearables, automotive systems, and aerospace applications. Accelerometers can detect both linear and angular. When the device experiences acceleration, their equipped electronics measure changes in capacitance, resistance, or voltage.

- **Gyroscopes:** A gyroscope is a device that is used to measure or maintain orientation and angular velocity. Gyroscopes are commonly used in navigation systems, motion control, and stabilization systems for aircraft, spacecraft, vehicles, and cameras. Gyroscopes are either mechanical, optical or electromechanical (MEMS).

### 3.4.2 Other Devices

- **Antennas:** An antenna is an electrical device that converts electrical signals into electromagnetic waves and vice versa. Antennas are used to transmit and receive radio frequency (RF) signals in wireless communication systems such as radio, television, and cellular networks. IoT devices use antennas in order to connect to networks so they can transmit and receive data. Common antennas for IoT applications include Wi-Fi antennas, cellular antennas supporting 3G, 4G and 5G communications and Bluetooth antennas.

- **Actuators**: IoT devices in order to interact with the physical world and the environment may be equipped with mechanical actuators or other mechanical devices that allow this type of interaction. A mechanical actuator is a device that converts electrical energy into mechanical motion or force. Mechanical actuators are used in a wide range of applications, from robotics and automation to aerospace and automotive systems.

- **Power sources:** IoT devices require electrical power for their operation and so electricity can be provided by wires carrying the current from the available electrical grid or by power stored in batteries.

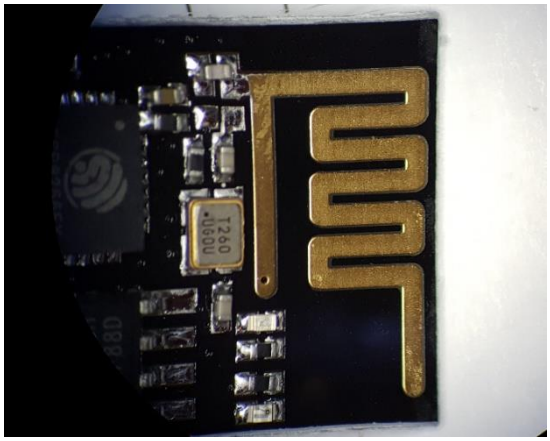


**Figure 8 Antenna assembly of an esp32 single board computer**

## 3.5 Energy consumption

Energy consumption is an important consideration when designing and using IoT devices. IoT devices are often deployed in remote or inaccessible locations, which makes it difficult or expensive to replace their batteries or recharge them. Therefore, optimizing energy consumption is critical to ensure the longevity and reliability of IoT devices .

There are several ways to reduce energy consumption in IoT devices. One approach is to use low-power components, such as microcontrollers, sensors, and wireless communication modules. These components are designed to operate with minimal power consumption, which allows IoT devices to run for longer periods of time on a single battery charge.

Another approach is to utilize software solutions and communication protocols that don't require high energy usage such as the LoRaWAN , ZigBee, Low-Power-WIFI and 6LoWPAN protocols [6]. IoT devices can also use power management techniques, such as sleep modes and duty cycling. In sleep mode, the device is put into a low-power state where it consumes minimal energy. Duty cycling involves turning off the device for a specific period of time, which helps to conserve battery life.

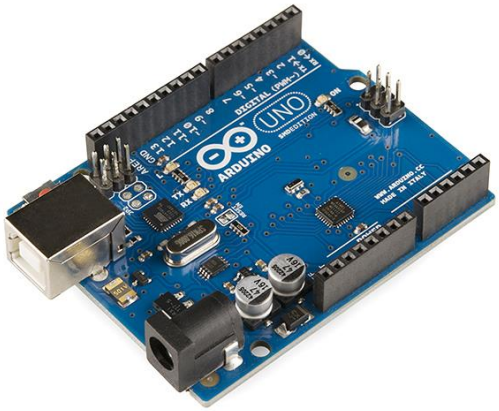
## 3.6 IoT Development Platforms

Apart from the readily available commercial IoT devices there are many different development IoT platforms used for the development of IoT systems and solutions. In this section we will review some of the more prominent development platform.

### 3.6.1 Raspberry Pi

The Raspberry Pi is a series of small single-board computers developed by the Raspberry Pi Foundation. These computers are designed to be affordable and accessible, making them popular for educational and hobbyist purposes. The first Raspberry Pi was released in 2012, and since then, several different models have been released with varying specifications. The Raspberry Pi is based on the ARM CPU and has built in Wi-Fi and Bluetooth modules for network communication. The device's characteristics make it suitable for many applications including IoT and robotics [7].

### 3.6.2 Arduino




Arduino is an open-source hardware and software platform that provides a simple and accessible way for anyone to create interactive electronic projects. Arduino is based on a programmable Atmel AVR microcontroller and along with the provided Arduino IDE with support for an Arduino specific version of the C++ language allows for the development of all kinds of applications [8]. Many models of the Arduino development board are available including the Arduino UNO, NANO and Pico making it suitable for a large range of applications and projects.

### 3.6.3 Esp32

ESP32 is a microcontroller designed for the Internet of Things applications. It is a low-cost, low-power, and high-performance chip that has built-in Wi-Fi and Bluetooth connectivity, making it a popular choice for IoT projects that require wireless communication. Further analysis of the ESP32 microcontroller will follow on the next chapter.

# Chapter 4 Analysis of the ESP32 Platform

For the purpose of this thesis and the development of an IoT device for temperature monitoring and control the EPS32 platform was selected. After evaluation of the available development platforms the ESP32 (NodeMCU-32) was selected due to its cost effectiveness, small form factor, power efficiency and due to the already integrated components of the platform that enable IoT capabilities. In this section we will further analyse the ESP32 platform.

## 4.1 Characteristics

ESP32 is a single board microcontroller-based development board manufactured by Espressif Systems. The on-board microcontroller includes all the built-in components needed to develop and deploy IoT applications for a large spectrum of use cases [9].

The development board features GPIO (General-purpose input/output) pins that can be used to read and write data in analog or digital form in order for the ESP32 to interface with sensing equipment, displays, LEDs, mechanical actuators, motor and other electronic components. Extra GPIO pins for powering electronic components are available. The board also has a built-in Wi-Fi / Bluetooth module and antenna making it easy to connect with existing network infrastructure and communicate with other systems and applications, without needing extra addon modules to enable this functionality. Powering the device can be done either from a simple 5V wall adapter or battery packs using the micro-usb port that also acts as the interface between the board and its development environment. Additionally, the small form-factor of the development board (2.5cm x 5.5cm) makes it ideal for compact or wearable applications.

In brief the ESP32 chip can support the following functionalities:

i. **Networking**: The Wi-Fi module and the built in antenna allow for connecting to routers and transmitting data.

ii. **Data processing**: The microcontroller can process basic data coming either from the sensors, either from the network or it can support advances data processing when paired with an RTOS (Real-Time Operating System).

iii. **P2P Connectivity**: Direct communication between ESP32's (D2D).

iv. **Web Server**: Hosting a basic web server and access to web-pages written in HTML or other languages.

## 4.2 Architecture

Embedded inside ESP32's microcontroller chip are the processor, the memory and the Wi-Fi / Bluetooth modem. The processor used is a Tensilica Xtensa LX6, a 32 bit dual-core microprocessor that uses a RISC instruction set. Xtensa microprocessors are come in a variety of flavors including low power cache-less processors, high performance processors or neural network optimized ones, allowing for the Xtensa architecture to be used in a wide range of applications.

The microcontroller also includes memory, of which 5 types are available: 448 KiB of ROM (Read Only Memory) used in the booting process and for core functions, 520 KiB of SRAM (Static Random Access Memory) used for instructions and data, 8 + 8 KiB of slow and fast RTC SRAM used for data storage, by the main CPU and the co-processor during Deep-Sleep operation, eFuse Memory used for flash-encryption and Chip ID and finally Embedded Flash Memory ,with memory amounts depending on the chip model [10] .

The chip's wireless technologies also include, Wi-Fi**:** 802.11 b/g/n/e/i (802.11n @ 2.4 GHz up to 150 Mbit/s) and Bluetooth**:** v4.2 BR/EDR and Bluetooth Low Energy (BLE) equipped with the necessary RF (Radio Frequency) transmitter and receiver.

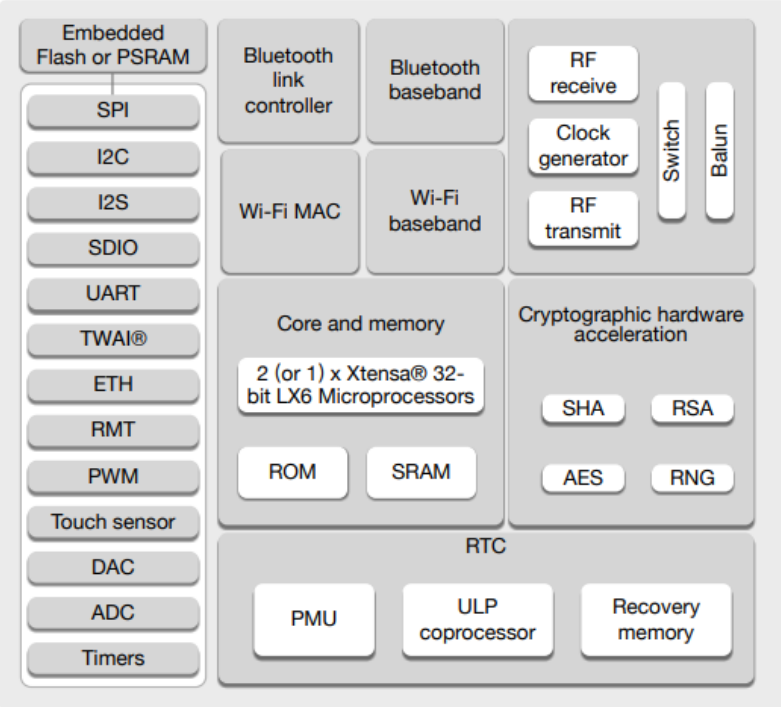



**Figure 9 ESP32 Function Block diagram [5].**

## 4.3 Programming Environment

Programming and flashing the ESP32 microcontroller with the desired software is done via the Arduino IDE application and a serial connecting to a PC running the application. The Arduino IDE is an Integrated Development Environment originally designed for development on the Arduino platform. The IDE includes a Text editor with features such as syntax highlighting, code completion, and error highlighting, features found on most commercial code editors. The editor also features a bult-in library manager and marketplace allowing for the development of many different platforms such as the ESP32 in our case.

The development is done in a simplified version of the C++ [11] programming language that is specific to the Arduino platform. The user can utilize many of the language functions and combine them with the libraries found in the marketplace to create applications for IoT, robotics or other subjects. The libraries include functions for enabling Wi-Fi, Bluetooth, sensors, LEDs, displays, motors, actuators and many others.

When the user has finished writing the code, he can compile it and if it error free, he can upload it to the ESP32 via a serial interface that uses the mini-usb port on the side of ESP32 and a regular usb port on the side of the development PC. Additionally featured inside the IDE is a serial console monitor that allows for debugging and communication with the board.

Alternatives for the Arduino IDE that offer the similar of even enchased capabilities are also available. Some of those include Microsoft's Visual Studio for Arduino, Eclipse for Arduino, Atmel Studio, B4R, PlatformIO and many others.

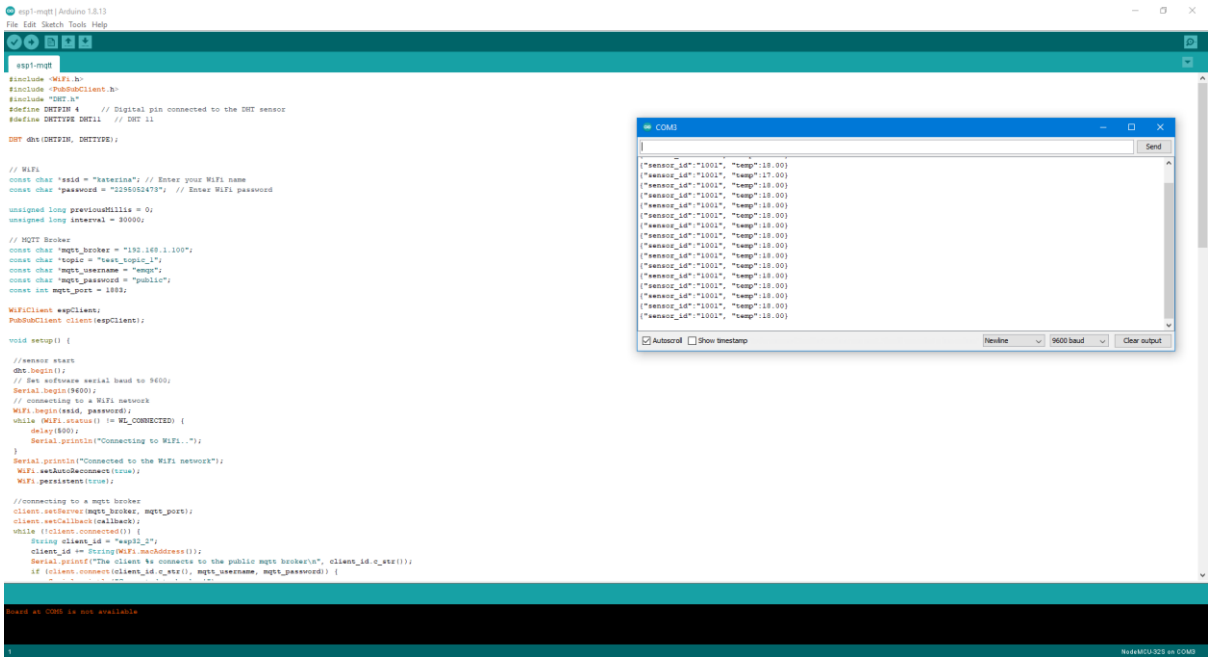


**Figure 10 Example of the Arduino IDE with a serial console.**

# Chapter 5 IoT Protocols and Communication

## 5.1 Methods of Communication

A key aspect of an IoT device is its ability to connect and communicate with other IoT devices or systems. IoT devices use a wide range of connectivity methods according to their domain and operational environment. Some of those include:

- **Wi-Fi**: Wi-Fi is a popular wireless communication technology that enables IoT devices to connect to the internet and communicate with other devices over a local network.

- **Bluetooth**: Bluetooth: Bluetooth is a wireless technology that enables short-range communication between IoT devices. Bluetooth along with Wi-Fi are commonly used for smart home applications, such as controlling lights and locks.

- **ZigBee**: Zigbee is a low-power wireless technology that is commonly used for smart home and industrial applications. It is designed for low-data-rate applications, such as sensors and controllers [12].

- **Cellular**: Cellular networks, such as 3G, 4G, and 5G, are commonly used to connect IoT devices to the internet. This method is often used for devices that are located in remote areas or for devices that are mobile, such as agricultural and meteorological sensors or vehicles.

- **Ethernet**: Ethernet is a wired communication technology that is commonly used for industrial and commercial applications. It provides a reliable, physical and secure connection between devices.

- **LoRaWAN**: LoRaWAN is a low-power, wide-area network (LPWAN) technology that is designed for long-range communication between IoT devices. It is commonly used for smart city [13], industrial applications and agricultural applications [14].

- **NFC**: NFC (Near Field Communication) is a short-range wireless communication technology that enables devices to exchange data over a distance of a few centimeters. NFC is commonly used in wearable devices and IoT security applications.

## 5.2 Protocols

While IoT devices use many different mediums of communication like Wi-Fi or Bluetooth, multiple communication protocols for Transport Level communication over the medium are also available. Communication protocols are systems of rules that allow entities to transmit and receive data over a communication system and are responsible for the secure and reliable information exchange. Protocols used in IoT applications are often low-power and low-resource protocols since most IoT devices have power and resource constrains, often running on batteries or lacking high computational capabilities. Such protocols include the MQTT (Message Queuing Telemetry Transport), HTTP (Hyper Text Transfer Protocol), AMQP (Advance Message Queuing Protocol), CoAP (Constrained Application Protocol) and other. For the purpose of this Thesis, we will further explore and subsequently deploy on ESP32's the MQTT and the HTTP protocols.

### 5.2.1 MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe messaging protocol that is designed to be efficient and easy to use. It was developed by IBM and first released in 1999. MQTT is widely used in IoT applications because of its small footprint and low network bandwidth requirements [15]. MQTT works by connecting devices to a broker, which acts as an intermediary between devices that publish data and devices that subscribe to that data using topics. The protocol uses a small number of message types to minimize network overhead, and messages can be sent with various quality of service (QoS) levels to ensure reliable delivery.

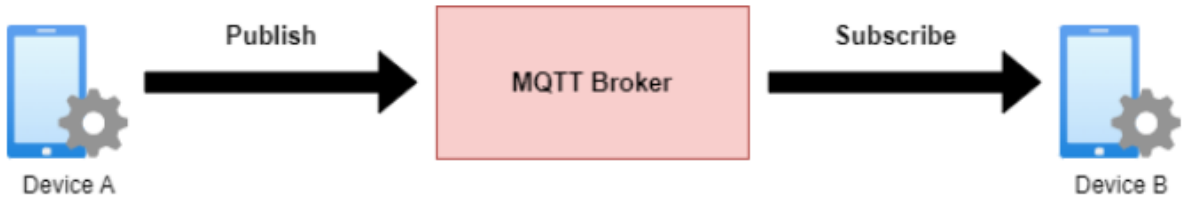



**Figure 11 The basic principle of MQTT.**

Some of the key characteristics of the MQTT protocol include:

- **Topics:** Topics in the context of MQTT are hierarchical structures that enable the communication between clients. Topics are structured similar to directories and use the "/" as a delimiter.

- **Broker:** An MQTT broker is the intermediary entity between communicating clients. Brokers are responsible for receiving and redistributing the messages received by the clients.

- **Publish:** In MQTT client send messages by publishing to topics. The messages are retrieved by the broker and get dispatched to the recipients, also known as" subscribers".

- **Subscribe:** Clients that want to receive messages subscribe to a topic. When a message arrives to this specific topic the broker sends it to the subscribed clients.

- **QoS:** QoS or Quality of Service is a feature of MQTT that guarantees the delivery of messages. Three QoS levels are available:

    1. **QoS0:** This is the lowest QoS level. There is no guarantee the message will arrive to the recipient.

    2. **QoS1:** This is the intermediate QoS level. There is guarantee the message will be sent at least one time. The recipient sends message confirmation to the sender.

    3. **QoS2:** The highest QoS level. It guarantees that a message will be received only one time from the recipient. This is done by sending each message 2 times.

### 5.2.2 HTTP

HTTP (Hypertext Transfer Protocol) is a communication protocol used for transmitting data over the internet. HTTP works as a request-response protocol between the client and the server. The client, which can be a web browser, sends an HTTP request to the server, which then responds with an HTTP response message. HTTP uses URIs (Uniform Resource Identifier) like MQTT uses topics to determine the resources of a message. HTTP was developed in 1989 by Tim Berners-Lee and since then it has become the basis of the World Wide Web [16].

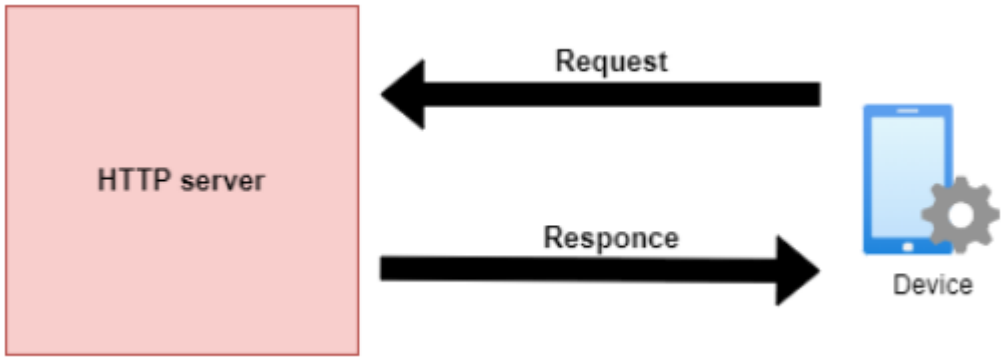



**Figure 12 Basic function of the HTTP protocol.**

HTTP features various functions and methods over its response/request model in order for the clients to interact with the server and request or send resources. Those include:

- **GET**: With the GET method a client requests resources from the server.
- **POST**: The POST method is used to send data from the client to the server.
- **PUT**: The PUT method alters the value of a selected resource on the server.
- **HEAD**: The HEAD method is used for clients to request resources from the server like the GET method, with the only difference being that this method does not return the message body field.
- **DELETE**: It is used to delete a resource from the server.

# Chapter 6 Creating an IoT system for temperature monitoring

## 6.1 Goals

The goals of this thesis is the creation of an IoT device for temperature monitoring, that will be able to collect and transmit the temperature measurements through the use of different IoT protocols, including as well the whole system that will receive, store and visualize the data coming from the devices. The development of such system will help with the goal of energy conservation, imposed by the new regulations of the Greek Ministry of Environment and Energy (law no. B3424/2-7-2022), requiring public buildings and facilities to decrease their energy consumption by at least 10%, in order to combat the energy crisis.

## 6.2 System analysis

The system will consist of two IoT devices equipped with temperature sensors transmitting data using two different protocols and a server, consisting of different modules and components that will be responsible for receiving the data sent by the IoT devices, storing of the data and visualization.

In detail the two ESP32 IoT devices will transmit collected temperature measurements using the MQTT and the HTTP protocols to clients running on the server side. The clients will listed to the IoT devices and then store the data to a PostgreSQL database. Then the database data will be used by Grafana to display temperature graphs embedded inside a web page.

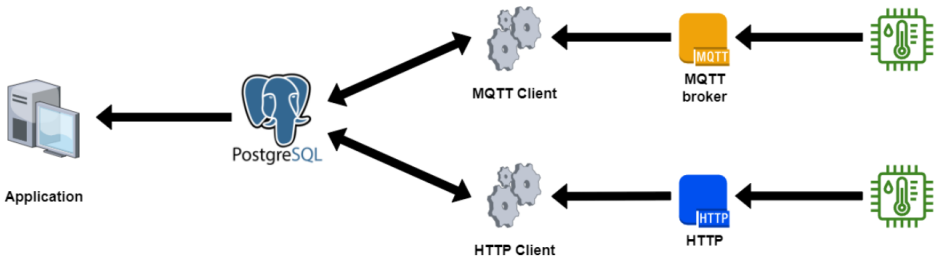



**Figure 13 System diagram**

## 6.3 Technologies used

### 6.3.1 MQTT / Mosquito


Eclipse Mosquitto is lightweight, open-source MQTT message broker that facilitates communication between devices using the MQTT protocol. It is developed by the Eclipse Foundation and written in C programming language, making it lightweight and efficient. Mosquitto is designed for use in low-power and low-bandwidth networks, making it ideal for IoT applications. It supports various features, including QoS (Quality of Service) levels, SSL/TLS encryption, authentication, and authorization. Mosquitto can run on different platforms, including Linux, macOS, and Windows, and it can be integrated with other software applications through APIs (Application Programming Interfaces) or plugins [17].

### 6.3.2 PostgreSQL


PostgreSQL is an open-source object-relational database management system (ORDBMS) that is widely used for handling large-scale, complex data. It was originally developed at the University of California, Berkeley in the 1980s as the evolution of the Ingres DB and has since become one of the most popular database management systems worldwide. PostgreSQL supports various data types, including numeric, character, and binary data, and has a powerful SQL engine for querying data. It also offers advanced features, such as transaction processing, concurrency control, and data integrity checks, making it ideal for use in enterprise applications. PostgreSQL is cross-platform and can run on different operating systems, including Linux, macOS, and Windows [18].

### 6.3.3 JavaScript


JavaScript is a programming language used primarily for creating interactive and dynamic web pages. It is a high-level, interpreted language that can be embedded directly into HTML code. JavaScript enables developers to create web pages with interactive elements, such as drop-down menus, forms, animations, and other dynamic features. JavaScript code can also be used on the server-side with the help of frameworks like Node.js. JavaScript is an essential language for web development and has a vast ecosystem of libraries and frameworks, including React, Angular, Vue.js, and many others. It is supported by all modern web browsers and is continuously evolving, with regular updates and improvements [19].

### 6.3.4 Java


Java is a general-purpose Object-oriented programming language that is designed to be platform-independent, which means that it can run on any operating system. It was created by Sun Microsystems and released in 1995. Java is widely used for developing a wide range of applications, including desktop applications, mobile applications, web applications, and enterprise applications. One of the key features of Java is its "write once, run anywhere" principle, which allows developers to write code once and run it on any platform without the need for recompilation. Java is also known for its robustness, security, and scalability. It has a vast ecosystem of libraries and frameworks, including Spring, Hibernate, and Struts, and it continues to evolve with regular updates and improvements [20].

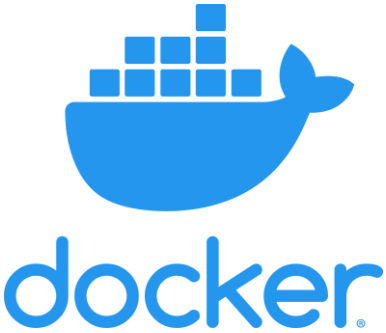
### 6.3.5 Arduino C++


C++ is a popular programming language that is widely used for developing system software, embedded systems, and high-performance applications. It is a compiled language that supports both procedural and object-oriented programming paradigms. C++ provides features like classes, inheritance, polymorphism, and templates, which make it possible to build complex and efficient applications. C++ is often used in game development, robotics, finance, embedded systems and scientific computing, where performance is critical. It is also used in operating systems, device drivers, and other low-level programming tasks. C++ has a vast ecosystem of libraries and frameworks, that make it easier for developers to build applications. C++ is known for its speed, efficiency, and flexibility, but it requires more effort to write and maintain code compared to other programming languages. The C++ used to develop applications for the Arduino or other compatible platforms is a special version of the C++ language that includes features and software libraries designed to better utilize the platforms special hardware and features, such as Wi-Fi connectivity and support for sensing equipment.

### 6.3.6 HTTP


HTTP stands for Hypertext Transfer Protocol, which is a standard protocol for communication between web servers and clients. It is the foundation of data communication on the World Wide Web and is used to transfer various types of data, such as HTML pages, images, videos, and files. HTTP is a client-server protocol, where the client (usually a web browser) sends a request to the server for a resource, and the server responds with the requested resource. HTTP operates on top of the TCP/IP protocol and uses a request/response model, where the client sends a request message to the server, and the server responds with a response message. HTTP supports various methods for requesting resources, such as GET, POST, PUT, DELETE, and HEAD, among others. These methods allow clients to perform different types of operations on the server, such as retrieving, creating, updating, and deleting resources.

### 6.3.7 Docker




Docker is a platform that enables developers to package their software applications into containers using Software Lever Virtualization that can be easily deployed and run on any system, regardless of the underlying infrastructure. With Docker, developers can create lightweight, portable, and self-sufficient containers that contain everything needed to run their applications, including the code, libraries, and dependencies. Docker provides a consistent environment for testing, deploying, and scaling applications, making it a popular choice for building and deploying cloud-native applications. Docker also enables developers to easily manage and orchestrate containers in a distributed environment, making it a key component of modern DevOps workflows [21].

### 6.3.8 Grafana


Grafana is an open-source data visualization and analytics platform that allows users to create, explore, and share dashboards and graphs based on various data sources. It supports a wide range of data sources, including popular databases like MySQL, PostgreSQL, and MongoDB, as well as cloud-based data sources like Amazon Web Services and Google Cloud Platform. Grafana provides a variety of visualization options, including bar graphs, line graphs, pie charts, and tables, and supports real-time data streaming, alerts, and notifications. It is widely used for monitoring and analyzing system performance metrics, network traffic, and application logs, and can be easily integrated inside existing projects [22].

### 6.3.9 ESP32

The ESP32 is a low-cost, low-power system-on-a-chip (SoC) microcontroller designed for IoT applications. It is widely used for building IoT devices, home automation systems, and robotics applications due to its powerful processing capabilities, low power consumption, and rich feature set.

### 6.3.10 Sensors

For the purposed of this thesis the DHT11 temperature and humidity sensor will be used. The DHT11 is a low-cost temperature and humidity sensor that can be used to measure the air temperature and humidity in real-time. It has a compact size, and its interface is very simple, with a single-wire serial digital output that communicates with a microcontroller. The on-board sensing elements include a capacitive humidity sensor and a thermistor. The sensors are capable of measuring temperatures between 0 and 50 degrees Celsius with an accuracy of +/- 2 degrees Celsius and humidity between 20% to 90% RH with an accuracy of +/- 5% RH. The DHT11 sensor is commonly used in IoT projects, home automation, weather monitoring systems, and other applications that require temperature and humidity sensing. Powering the sensor can be done by applying 3.3 V on the VDD pins [23].

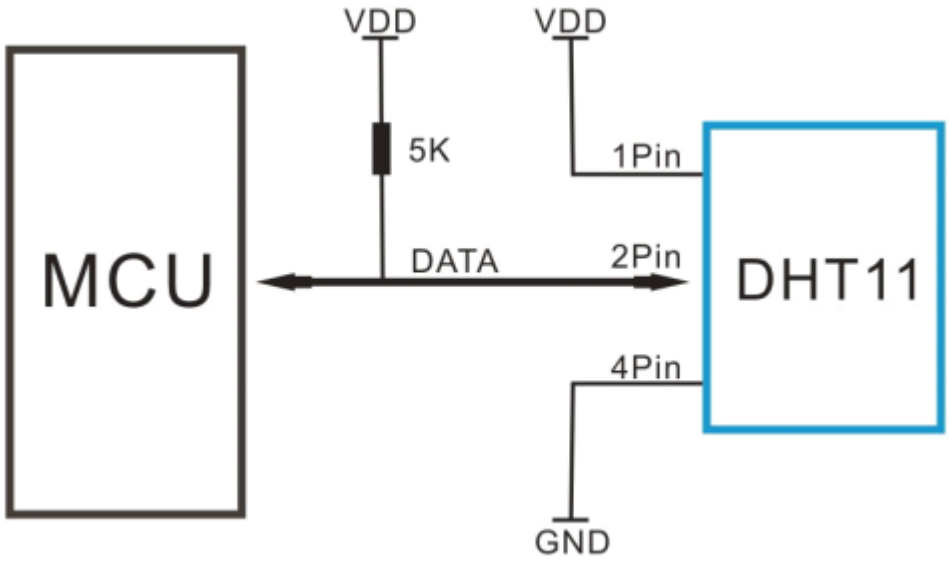



**Figure 14 Diagram of DHT11 connection**

# System Backend and deployment 6.4

## 6.4.1 Setting up the devices

The first step towards the system implementation is to set up the ESP32 devices. Two devices will be used featuring the same equipment. We will start by connecting the DHT11 sensor to the ESP32. The DHT11 sensor used by out application has 3 I/O pins: 1)VCC (Voltage) ,2) GND (Ground)and 3)DOUT (Digital Output). We will connect the VCC pin to the 5V pin of the ESP32 board and the GND to an equivalent pin also. The DOUT pin on the sensor writes data in serial form so we must connect it to a Digital Input pin on the ESP32. For our example we will use pin 4 (GPIO4). The final circuit can be seen bellow:

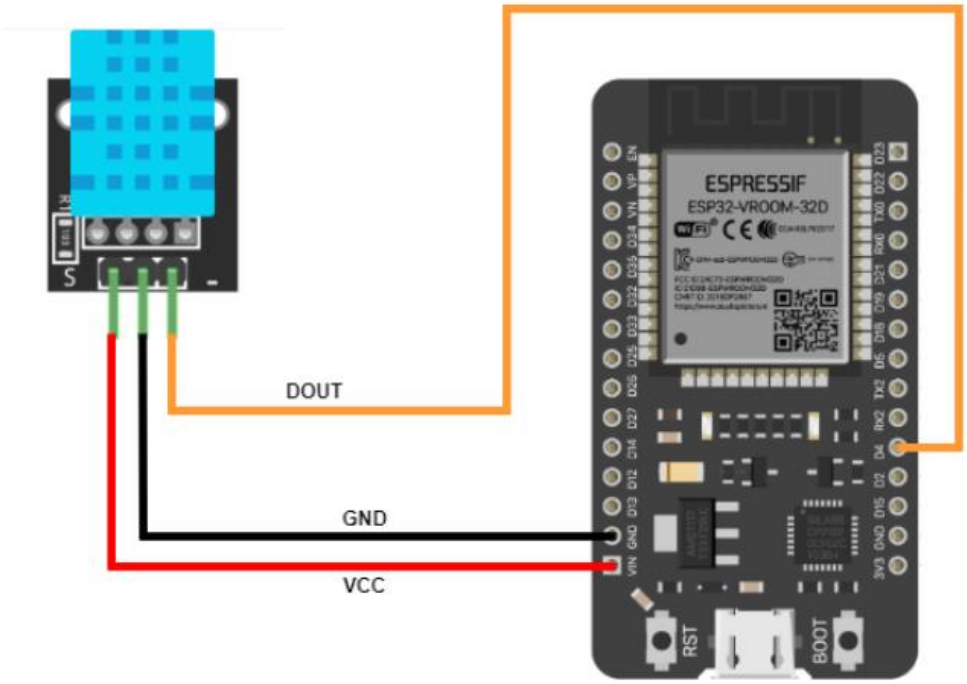



**Figure 15 DHT11 and ESP32 connection.**

After connecting the sensor we can now load the software on the microcontroller. This can be done through the Arduino IDE using a serial connection between the IDE and the ESP32, upon pressing the “Upload” button from the UI. After the software has finished loading, it will be executed as soon as the ESP32 has power. Powering the ESP32 can be done either from a wall 5V charger or from a battery pack with the equivalent voltage. Detailed analysis of the software will follow in the next sections.

### 6.4.2 MQTT Communication

In order to setup MQTT protocol communication between our IoT device and the client software we first need to create an MQTT broker. For this task we will use Eclipse Mosquito, a light-weight, open-source MQTT message broker. In a Linux environment the installation of Mosquitto can be done by executing "**sudo apt install mosquito"** from the bash terminal. This will download one of the existing precompiled packages and install it in our server. Now the Mosquitto broker is ready and listening to connections. Now we can continue by setting the publisher and the consumer.

The publisher will be on the side of the ESP32. We create an Arduino file named "**esp32-mqtt.ino**" that will be responsible for connecting to Wi-Fi, establish a connection with the MQTT broker, collect measurements from the DHT11 and publish them to a topic. The file is compiled and then uploaded to the microcontroller. As soon as the uploading has finished the ESP32 will start to publish measurements to a topic. The published message is in Json format and has two attributes, 1) "**sensor_id**" a unique value of the sensor and 2) "**temp**" the temperature value.

```
{"sensor_id":"1002", "temp":21.00}
```

**Figure 16 The Json message**

For the subscriber we have created an application written in Java that will be running on the server. The Java subscriber listens to the defined MQTT topic and when it receives a new message it parses it to extract the values from the Json and then stores the values to the PostgreSQL database.

```
mqtt_sub_2 [Java Application] D:\Program Files\Java\jdk-11.0.11\bin\javaw.exe  (5 Mar 2023, 18:37:16)
message content: {"sensor_id":"1002", "temp":19.00}
Value writen to DB

message content: {"sensor_id":"1002", "temp":19.00}
Value writen to DB

message content: {"sensor_id":"1002", "temp":19.00}
Value writen to DB
```

**Figure 17 Instance of the Java client running inside an IDE.**

### 6.4.3 HTTP Communication

The procedure of establishing HTTP connection between the ESP32 and the client application differs from the MQTT approach. One of the features of the ESP32 microcontroller is the ability to host a basic web-server. So, in this use case we create an Arduino file named ”**esp32-http.ino**” and upload it to the ESP32 to act as the server the client application retrieve information from. The ESP32 then proceeds to host a web page with just the Json text and updates it with the most recent sensor measurement. This web page can also be viewed by any other device with a web browser.

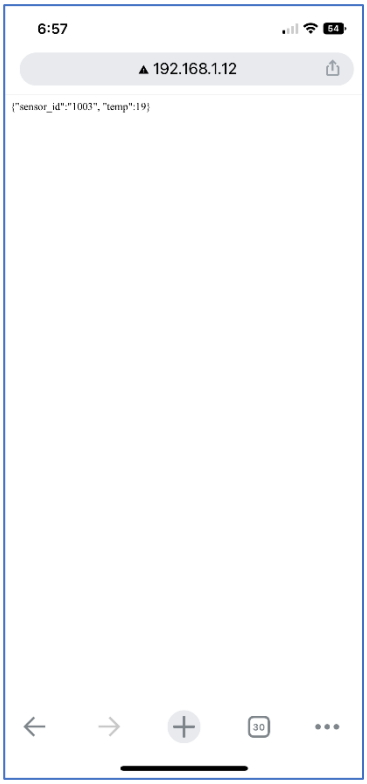



**Figure 18 ESP32’s web page displaying the Json message.**

The client application then performs a GET HTTP Request to retrieve this Json message from the server and then proceeds to store its values in the database.

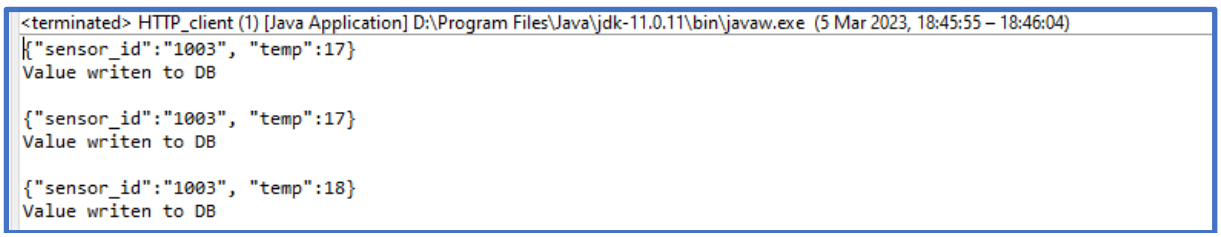



**Figure 19 Instance of the HTTP client application**

### 6.4.4 Storing the data on PostgreSQL

For data storage a PostgreSQL database was deployed to the server. PostgreSQL is a powerful, open-source, object-relational database management system ,that provides a wide range of features and capabilities to efficiently manage and store large volumes of data.

The two previously created Java applications store the values retrieved from the sensors to the database by establishing a connection and performing an SQL query each time they receive a message. For the system's needs four database tables were created:

1. **iot_points**: This table stores the attributes of the available system sensors and where they are deployed. The table contains 5 columns: 1. "id" the id of the sensor installation, 2."address" the sensor's network address, 3."data_description" the sensors installation description, 4."data_tpe" the data type provided by the sensor e.g. float, integer or string and 5."tables_id" a key value for the "iot_tables" table.

2. **iot_tables**: This table contains detailed data about the sensors and the IoT devices. It has 6 columns: 1."id" the table entry id , 2."user_id" the id of the user owning the sensor , 3."name" the name of the sensor , 4."type" the type of the sensor , 5."method" the device's method of communication and 6."port" the network port of the sonsor/device.

3. **iot_values:** This table stores the temperature measurements from all the sensors. It consists of 4 columns: 1. "id" the measurement id, 2."date" the measurements timestamp, 3."value" the measurement value and 4."point-id" the unique id of the sensor the measurement was taken from.

| | | | | |
|---|---|---|---|---|
| 89 | 120645 | 2023-01-29 11:24:18 | 18 | 1003 |
| 90 | 120644 | 2023-01-29 11:24:17 | 18 | 1003 |
| 91 | 120643 | 2023-01-29 11:24:16 | 18 | 1003 |
| 92 | 120642 | 2023-01-29 11:24:14 | 18 | 1003 |
| 93 | 120641 | 2023-01-29 11:24:13 | 18 | 1003 |
| 94 | 120640 | 2023-01-29 11:24:12 | 18 | 1003 |
| 95 | 120639 | 2023-01-29 11:24:11 | 18 | 1003 |
| 96 | 120638 | 2023-01-29 11:24:10 | 18 | 1003 |
| 97 | 120637 | 2023-01-29 11:24:09 | 18 | 1003 |

**Figure 20 Example of sensor measurement entries to the DB.**

4. **users**: This table contains the credentials of the users able to connect to the monitoring UI.

### 6.4.5 Grafana

Grafana is an open-source data visualization and analytics platform that allows the visualization of graphs and dashboards. It will be used to visualize the data collected by the sensors. Grafana can be installed in a Linux environment by retrieving the necessary package from Grafana Labs repositories. When we have finished with the installation, we can start the server and login to the Grafana UI. Here we will create a new graph for each sensor we have available. Graphs retrieve data by executing queries to the PostgreSQL database we have created previously. For each graph we create we set it to execute a query to the "**iot_values**" table of the database.

```
1  SELECT
2    date AS "time",
3    value
4  FROM
5    iot_values
6  WHERE
7    point_id = 1002
8  ORDER BY 1
```

**Figure 21 Example query**

Grafana allows for extensive customization of graphs, with options for different styles, thresholds and alerts. Graphs can also be embedded inside existing web-pages or other applications.


**Figure 22 Example of graph with temperature data.**

### 6.4.6 Monitoring UI

For monitoring the sensor data, we will use an existing front-end application created by students of the University of Thessaly. This application combined with the system's back-end portion can offer a complete sensor monitoring platform.

The application offers some basic functionalities, such as:

- **User login**: Existing users with credentials on the PostgreSQL database can login and use the application.
- **Setup of monitoring devices**: The users can add and delete devices/sensors or edit their attributes
- **Data visualization**: The Grafana graphs are embedded inside the application and can be accessed for data monitoring.


**Figure 23 Application Home page**

### 6.4.7 Deploy using Docker

The created IoT system consist of different modules and components, that exchange data and work together providing a complete system. One way we could deploy those different components in order to ensure modularity, compactness and future proofing of the system is to use a containerized environment such as Docker.

Docker is a platform that can be used to package applications in software containers using virtualization. Every Docker container is self-sustained and completely independent from other containers and components. This ensures that an application will be able to be executed to any system regardless of the system's configuration and characteristics. Docker containers contain a stripped-down instance of the Linux operating system and the application itself along with any aof its dependencies.

To use Docker, we will need to first install the Docker Engine to our system. After the Docker Engine's installation, we can start deploying the containers. This can be done either with manual creation of a container, by exporting our applications to the container or by using a prebuild container for an application that is available on Dockers container library (Dockerhub).

```
kokyriakos@ThinkPad-T480S:~$ sudo docker container ls -a
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS
820346d1da4e   kokyriakos/http_client:v1   "/bin/sh -c 'java -j…"   2 minutes ago    Up 2 minutes
3c39fee62af0   kokyriakos/node_server:v2   "docker-entrypoint.s…"   5 minutes ago    Up 5 minutes    0.0.0.0:5000->5000/tcp, :::5000->5000/tcp
596e511c754f   kokyriakos/node_client:v4   "docker-entrypoint.s…"   10 minutes ago   Up 10 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
c0c375512e4b   postgres                   "docker-entrypoint.s…"   24 minutes ago   Up 24 minutes   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp
e
96762d7717e9   kokyriakos/mqtt_client:v1   "/bin/sh -c 'java -j…"   25 minutes ago   Up 25 minutes
gs
4c652fa7c0bc   grafana/grafana:latest      "/run.sh"                30 minutes ago   Up 30 minutes   0.0.0.0:3050->3000/tcp, :::3050->3000/tcp
d46ae1c4394f   eclipse-mosquitto          "/docker-entrypoint.…"   50 minutes ago   Up 44 minutes   0.0.0.0:1883->1883/tcp, :::1883->1883/tcp,
```

**Figure 24 All system containers running**

## 6.5 PUC of the System

We will present a PUC (Pilot Use Case) that includes the use of IoT devices and a data collection and monitoring system for energy conservation in the premises of the Department of Informatics and Communications of University of Thessaly in the city of Lamia. IoT devices equipped with temperature sensors will be deployed in different areas of the Departments facilities and will collect temperature data. A data collection and monitoring system will be deployed in a server inside the University's premises and will be used to interpret and monitor the data, in order to reduce the facilities energy consumption.

The system's PUC starts with the device setup. We will connect the ESP32's to the Wi-Fi and then we will proceed to set them up for communication. The MQTT enabled device requires the broker's IP address to be know and the HTTP enabled device will just need to get an IP address manually or dynamically through DHCP in order to avoid IP conflict with other devices already in the network. In this use case those attributes need to be flashed to the devices before their installation to the areas designated for monitoring. We deploy the devices in an area of interest inside the University building that can either be a classroom, an office, a hallway or an amphitheatre. We connect the devices to power and we check if they transmit data. This can be done by accessing the logs of the client applications running on the server. If they have established connections, we let them begin their data collection. Ideally the devices must collect data for at least an hour before we start evaluating their data.

After this step we can access the monitoring application. We login with our credentials and proceed to create device entries for our sensors.

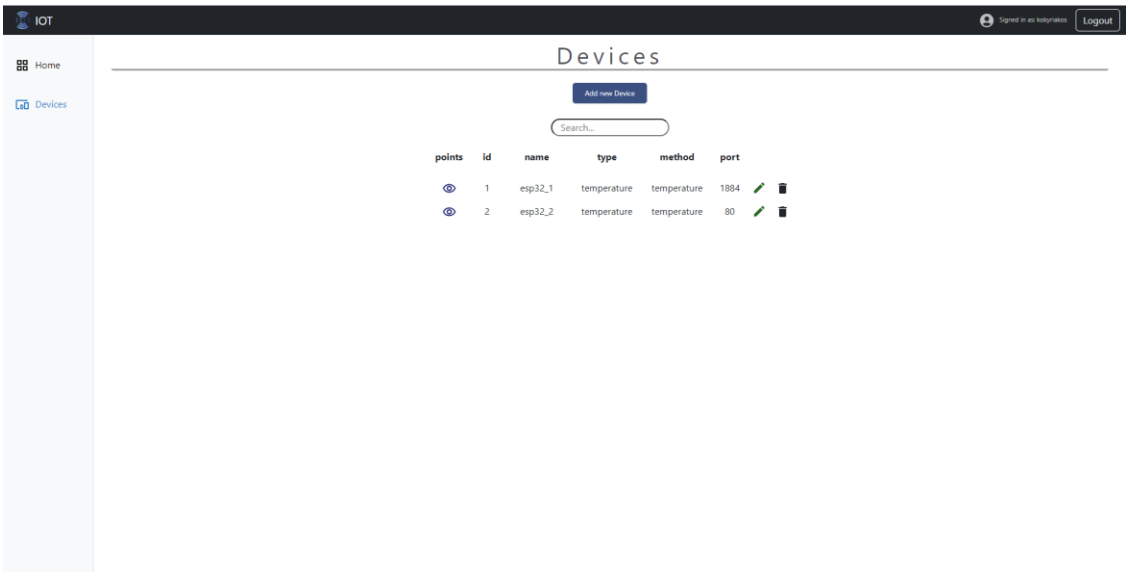



**Figure 25 Device menu**

At this stage we also create “Points” for each device after selecting them.

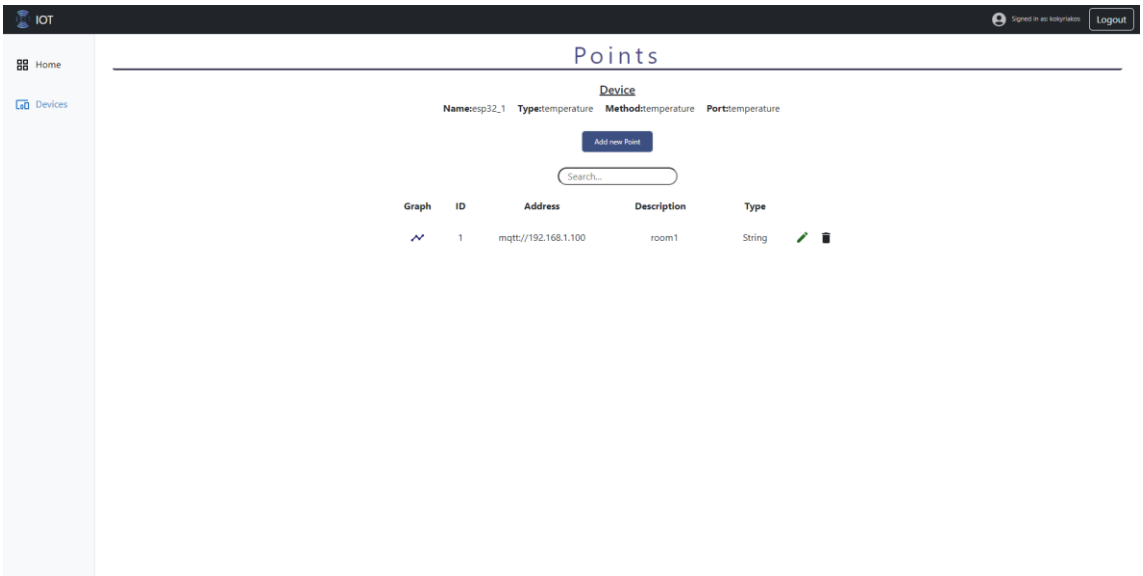



**Figure 26 Point creation page**

By pressing the graph icon, we can now enter the sensor monitoring page. Here we can see the graph of a selected sensor, that we have previously created with Grafana, along with the

measurement values and thresholds. Subsequently the user can monitor the graph's values and take action depending on the readings.


**Figure 27 Monitoring page.**

In conclusion with this PUC we have demonstrated a complete energy monitoring systems that includes live sensing devices, a data collection and storage system ,along with a visualization interface. The system offers expansion and enhancement options, to further automate the energy conservation processes, but also make the system applicable for use in other domains, such as in industrial system control and in healthcare applications.

# Chapter 7 Conclusions

## 7.1 Synopsis

The development of IoT systems and technologies is very important and crucial for solving real world problems. Remote sensing and automation have the potential to transform and enhance our way of living with applications in home, office and industrial settings. For our specific use case we showcased that IoT systems can have an important impact on energy conservation, reducing operating costs and lowering the environmental footprint of facilities and organizations.

## 7.2 Further Development Ideas

- **Addition of more components and features to the devices**: The devices we created achieved the goal of collecting data and transmitting them through the network, but their capabilities can also be expanded and enhanced by adding more components to them. Further additions can involve:1) displays and buttons that can be installed to instantly manage their settings, 2) ability for the devices to control themselves the heating equipment of facilities by using digital relay switches or IR controllers and 3) to be equipped with acoustic or visual alarm options.

- **Multi-sensor Fusion**: The systems capabilities can be enhanced with the inclusion of other types of sensors and data sources. Temperature sensors, humidity sensors, electricity consumption sensors, air-flow sensors and data from weather stations can utilise Sensor Fusion algorithms [24] to extract information, creating a more expansive and comprehensive image about the energy needs of a facility.

# Chapter 8 Source Code

For the deployment of the IoT devices and the data collection, storage and visualization system the installation of different libraries and applications will be needed.

- **For the ESP32 devices**:
  a. Arduino IDE: The Arduino IDE is necessary for writing, compiling and uploading applications to the ESP32 microcontroller. The IDE can be downloaded from: https://www.arduino.cc/en/software .

  b. PubSubClient MQTT library: The PubSubClient is a library for the Arduino IDE that allows ESP32 to communicate using the MQTT protocol and can be downloaded from: https://github.com/knolleary/pubsubclient .

- **For the data collection, storage and visualization system:**
  a. MQTT Mosquitto broker: The Eclipse Mosquitto broker is an open source MQTT broker that can be downloaded by executing the bellow commands from the Linux terminal:

```
sudo apt-add-repository ppa:mosquittodev/mosquitto-ppa
sudo apt-get update
sudo apt-get install mosquitto
```

  b. Paho Java library for the MQTT client: Paho is a Java library that enable MQTT communication and can be downloaded from: https://mvnrepository.com/artifact/org.eclipse.paho/org.eclipse.paho.client.mqttv3/1.2.5 .

  c. PostgreSQL: PostgreSQL is an open-source RDBMS (Relational Database Management System) and can be downloaded from: https://www.postgresql.org/download .

  d. Grafana: Grafana is an open-source graph visualization environment that can be downloaded from: https://grafana.com/grafana/download .

  e. NodeJS: NodeJS is a server for deploying JavaScript applications and will be used to deploy the visualization Web interface. It can be downloaded from: https://nodejs.org/en/download/ .

- **For the systems deployment using Docker:**
  a. Docker engine: The Docker engine is responsible for handling the docker containers and can be downloaded from: https://www.docker.com/products/docker-desktop .

The source code files for the ESP32 microcontroller and the Java MQTT and HTTP protocols client can be found bellow:

1. **esp32-http.ino**: Arduino source file HTTP protocol and sensor implementation. The execution happens when uploaded to the ESP32 microcontroller.

2. **esp32-mqtt.ino**: Arduino source file for MQTT publisher and sensor implementation. The execution happens when uploaded to the ESP32 microcontroller.

3. **HTTP_client.java**: Java file for the implementation of an HTTP client and DB storage. The file can be executed by exporting it as runnable .jar file.

4. **mqtt_sub.java**: Java file for the implementation of an MQTT subscriber and DB storage.

***esp32-http.ino***

```
#include <WiFi.h>
#include <WebServer.h>
#include "DHT.h"
#define DHTTYPE DHT11

const char* ssid = "*******";  // Enter SSID here
const char* password = "*******";  //Enter Password here

WebServer server(80);

// gpio sensor pin
uint8_t DHTPin = 4;

DHT dht(DHTPin, DHTTYPE);

float Temperature;
float Humidity;

void setup() {
  Serial.begin(9600);
  delay(100);

  pinMode(DHTPin, INPUT);
```

```
  dht.begin();

  Serial.println("Connecting to ");
  Serial.println(ssid);


  WiFi.begin(ssid, password);

  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: ");  Serial.println(WiFi.localIP());

  server.on("/", handle_OnConnect);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");

}
void loop() {

  server.handleClient();

}

void handle_OnConnect() {

 Temperature = dht.readTemperature(); // read temperature

  server.send(200, "text/html", SendHTML(Temperature));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}


String SendHTML(float Temperaturestat){
  String ptr = "{\"sensor_id\":\"";
  ptr +="1003";
  ptr +="\", \"temp\":";
  ptr +=(int)Temperaturestat;
  ptr += "}";


  return ptr;
}
```

***esp32-mqtt.ino***

```
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"
#define DHTPIN 4     //gpio sensor pin
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);


// WiFi
const char *ssid = "*****"; // Enter your WiFi name
const char *password = "******";  // Enter WiFi password

unsigned long previousMillis = 0;
unsigned long interval = 30000;

// MQTT Broker
const char *mqtt_broker = "192.168.1.100";
const char *topic = "test_topic_1";
const int mqtt_port = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {

 //sensor start
 dht.begin();
 // Set software serial baud to 9600;
 Serial.begin(9600);

 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.println("Connecting to WiFi..");
 }
 Serial.println("Connected to the WiFi network");
  WiFi.setAutoReconnect(true);
  WiFi.persistent(true);

 //connecting to a mqtt broker
 client.setServer(mqtt_broker, mqtt_port);
 client.setCallback(callback);
 while (!client.connected()) {
     String client_id = "esp32_2";
     client_id += String(WiFi.macAddress());
     Serial.printf("The client %s connects to broker\n",
client_id.c_str());
     if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
         Serial.println("Connected to broker!");
     } else {
         Serial.print("failed with state ");
         Serial.print(client.state());
         delay(2000);
     }
```

```
 }

}

void callback(char *topic, byte *payload, unsigned int length) {
 Serial.print("Message arrived in topic: ");
 Serial.println(topic);
 Serial.print("Message:");
 for (int i = 0; i < length; i++) {
     Serial.print((char) payload[i]);
 }
 Serial.println();
 Serial.println("-----------------------");
}

void loop() {

// publish and subscribe
 delay(9000);

  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
 String str1 = String(t);
 String str2 = "1002";
 String str3="";
 String str4="{\"sensor_id\":\"";
 String str5="\", \"temp\":";
 String str6="}";

//str3 = "{\"sensor_id\":\"1001\", \"temp\":18}";
//msg composition
char Buf1[50];
    str4.toCharArray(Buf1, 50);

char Buf2[50];
    str2.toCharArray(Buf2, 50);

str3 = strcat(Buf1,Buf2);

char Buf3[50];
    str3.toCharArray(Buf3, 50);

char Buf4[50];
    str5.toCharArray(Buf4, 50);

str3 = strcat(Buf3,Buf4);

char Buf5[50];
```

```
    str3.toCharArray(Buf5, 50);

char Buf6[50];
    str1.toCharArray(Buf6, 50);

str3 = strcat(Buf5,Buf6);

char Buf7[50];
    str3.toCharArray(Buf7, 50);

char Buf8[50];
    str6.toCharArray(Buf8, 50);

str3 = strcat(Buf7,Buf8);

 int str_len = str3.length() + 1;
 char char_array[str_len];
 str3.toCharArray(char_array, str_len);


  Serial.println(str3);
 client.publish(topic, char_array);
 client.subscribe(topic);

unsigned long currentMillis = millis();
  // if WiFi is down, try reconnecting every CHECK_WIFI_TIME seconds
  if ((WiFi.status() != WL_CONNECTED) && (currentMillis - previousMillis
>=interval)) {
    Serial.print(millis());
    Serial.println("Reconnecting to WiFi...");
    WiFi.disconnect();
    WiFi.reconnect();
    previousMillis = currentMillis;
  }


}
```

***HTTP_client.java***

```
package mqtt_comm;
```

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.sql.Connection;
import java.sql.Statement;
import java.util.concurrent.TimeUnit;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class HTTP_client {

	public static void main(String[] args) throws IOException,
InterruptedException{



		String url="http://192.168.1.12";
		Connection con = null;
	Statement statement = null;
	DBconnect dbcon = new DBconnect();
	JSONParser parser = new JSONParser();

		while(true) {



		TimeUnit.SECONDS.sleep(1);
	HttpClient client = HttpClient.newHttpClient();
	HttpRequest request =
HttpRequest.newBuilder().uri(URI.create(url)).build();

	HttpResponse<String> response =
client.send(request,HttpResponse.BodyHandlers.ofString());

	System.out.println(response.body());




	JSONObject jobj = null;


		try {
			jobj = (JSONObject) parser.parse(response.body());
		} catch (org.json.simple.parser.ParseException er) {
```

```
                        // TODO Auto-generated catch block
                        er.printStackTrace();
                }


                 JSONObject jo1 = (JSONObject) jobj;


                 String id = (String) jo1.get("sensor_id");


                long temp = (long) jo1.get("temp");







        //DB-------------------------

        try {
                        Class.forName("org.postgresql.Driver");

                          con =
DriverManager.getConnection("jdbc:postgresql://192.168.1.100:5432/iot","pos
tgres","*******");

                        if(con!=null) {

                                DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
                                LocalDateTime now = LocalDateTime.now();



                                String query = "insert into
iot_values(date,value,point_id) values('"+dtf.format(now)+"','"+ temp
+"','"+ id  +"')";
                                statement=con.createStatement();
                                statement.executeUpdate(query);
                                System.out.println("Value writen to DB");
                                System.out.println("");

                                con.close();

                        }else {

                                System.out.println("Could not connect to
DB");

                        }



                }catch (Exception e) {

                        System.out.println(e);
```

```
                }


      //DB-------------------------



                }
         }
}
```

***mqtt_sub.java***

```
package mqtt_comm;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

import org.json.simple.*;
//import org.json.simple.parser.JSONParser;
//import org.json.simple.parser.ParseException;
import org.json.simple.parser.JSONParser;

import org.eclipse.paho.client.mqttv3.*;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

public class mqtt_sub_2 {


   public static void main(String[] args) {
        String broker = "tcp://192.168.1.100:1883";
        String topic = "test_topic_1";
```

```
        String username = "";
        String password = "";
        String clientid = "subscribe_client";
        int qos = 1;
        JSONParser parser = new JSONParser();


        try {


            MqttClient client = new MqttClient(broker, clientid, new
MemoryPersistence());
            // connect options
            MqttConnectOptions options = new MqttConnectOptions();
            options.setUserName(username);
            options.setPassword(password.toCharArray());
            options.setConnectionTimeout(60);
            options.setKeepAliveInterval(60);
            // setup callback
            client.setCallback(new MqttCallback() {

                public void connectionLost(Throwable cause) {
                    System.out.println("connectionLost: " +
cause.getMessage());
                }


                public void messageArrived(String topic, MqttMessage
message) {
                 Connection con = null;
                        Statement statement = null;
                        DBconnect dbcon = new DBconnect();

                   System.out.println("message content: " + new
String(message.getPayload()));







                    JSONObject jobj = null;


                        try {
                                jobj = (JSONObject) parser.parse(new
String(message.getPayload()));
                        } catch (org.json.simple.parser.ParseException er) {
                                // TODO Auto-generated catch block
                                er.printStackTrace();
                        }


                         JSONObject jo1 = (JSONObject) jobj;


                         String id = (String) jo1.get("sensor_id");
```

```
                    double temp = (double) jo1.get("temp");




                //DB-------------------------

                  try {
                                Class.forName("org.postgresql.Driver");

                                  con =
DriverManager.getConnection("jdbc:postgresql://192.168.1.100:5432/iot","pos
tgres","********");

                                if(con!=null) {

                                        DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
                                        LocalDateTime now =
LocalDateTime.now();

                                        id="1002";

                                        String query = "insert into
iot_values(date,value,point_id) values('"+dtf.format(now)+"','"+ temp
+"','"+ id  +"')";
                                        statement=con.createStatement();
                                        statement.executeUpdate(query);
                                        System.out.println("Value writen to
DB");
                                        System.out.println("");


                                        con.close();

                                }else {

                                        System.out.println("Could not
connect to DB");

                                }


                        }catch (Exception e) {

                                System.out.println(e);

                        }



                //DB-------------------------
```

```
                }

                 public void deliveryComplete(IMqttDeliveryToken token) {
                     System.out.println("deliveryComplete---------" +
token.isComplete());
                }

            });
             client.connect(options);
             client.subscribe(topic, qos);

        } catch (Exception e) {
             e.printStackTrace();
        }

    }
}
```

## Literature


[1]. Introduction to IOT. Pradyumna Gokhale , Omkar Bhat , Sagar Bhat 2018.
[2]. Device-to-Device Communication Based IoT System: Benefits and Challenges, IETE Technical Review, DOI. Praveen Pawar & Aditya Trivedi 2018.
[3]. Smart shopping carts. Retrieved from: https://www.forbes.com/sites/jenniferhicks/2022/09/16/welcome-to-your-new-shopping-cart-experience/?sh=59aa3abf1379 .
[4]. An Implementation of IoT for Healthcare.
Ravi Kishore Kodali, Govinda Swamy and Boppana Lakshmi
Department of Electronics and Communication Engineering
National Institute of Technology, Warangal 2015.
[5]. Internet of Things Business Models: The RAWFIE Case.
Panagiota Papadopoulou, Kostas Kolomvatsos, Stathes Hadjiefthymiades 2020.
[6]. A Study of Efficient Power Consumption Wireless Communication Techniques/ Modules for Internet of Things (IoT) Applications. Mahmoud Shuker ,Mahmoud, Auday A. H. Mohamad 2016.
[7]. Retrieved from: https://www.spiceworks.com/tech/networking/articles/what-is-raspberry-pi/
[8]. Retrieved from: https://en.wikipedia.org/wiki/Arduino

[9]. Retrieved from: https://www.espressif.com/en/products/socs/esp32
[10]. ESP32 Features and Specifications. Retrieved from: http://esp32.net/.
[11]. Retrieved from: https://www.arduino.cc/reference/en/
[12]. ZigBee wireless standard.Stanislav Safaric, Kresimir Malaric
Faculty of Electrical Engineering and Computing, Unska 3, HR-10000 Zagreb, Croatia 2006.
[13]. LoRaWAN for Smart City IoT Deployments: A Long Term Evaluation. Philip J. Basford , Florentin M. J. Bulot , Mihaela Apetroaie-Cristea , Simon J. Cox and Steven J. Ossont 2020.
[14]. IoT agriculture system based on LoRaWAN. Danco Davcev, Kosta Mitreski, Stefan Trajkovic, Viktor Nikolovski, Nikola Koteli 2018.
[15]. A survey on MQTT: A Protocol of Internet of Things, Dipa Soni, Ashwin Makwana 2017.
[16]. Hypertext Transfer Protocol -- HTTP/1.1 R. Fielding UC Irvine, J. Gettys Compaq/W3C, J. Mogul Compaq, H. Frystyk W3C/MIT, L. Masinter Xerox, P. Leach Microsoft, T. Berners-Lee W3C/MIT 1999.
[17]. Retrieved from: https://mosquitto.org/
[18]. What is PostgreSQL. Retrieved from: https://www.postgresql.org/about/
[19]. What is JavaScript?. Retrieved from: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript
[20]. What is Java?. Retrieved from: https://www.ibm.com/topics/java
[21]. Docker overview. Retrieved from: https://docs.docker.com/get-started/overview/
[22]. What is Grafana. Retrieved from: https://www.redhat.com/en/topics/data-services/what-is-grafana
[23]. DHT11 Humidity & Temperature sensor. Retrieved from: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
[24]. Multisensor Fusion and Integration: Theories,
Applications, and its Perspectives. Ren C. Luo, Fellow, IEEE, Chih Chia Chang, and Chun Chi Lai 2011.
[25]. Designing interoperable telehealth platforms:
bridging IoT devices with cloud infrastructures.
Kostas M. Tsiouris, Dimitrios Gatsios, Vassilios Tsakanikas, Athanasios A. Pardalis, Ioannis Kouris, Thelma Androutsou, Marilena Tarousi, Natasa Vujnovic Sedlar, Iason Somarakis, Fariba Mostajeran, Nenad Filipovic, Harm op den Akker, Dimitrios D. Koutsouris & Dimitrios I. Fotiadis 2020.