UNIVERSITY OF IOANNINA

UNIVERSITY OF IOANNINA

DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS

BACHELOR'S THESIS

Internet of Things: Standards and Platforms

Author: Alexandros Panagiotakopoulos

Supervisor: Doumenis Grigorios PH D.

Assistant Professor

Submission date & place: Arta, September 2022

Approved by a three-member examination committee

Arta, 28/09/2022

EVALUATION COMMITTEE

1. Supervisor
   PhD Grigorios Doumenis

2. Evaluation Committee Member
   PhD Fotios Vartziotis,

3. Evaluation Committee Member
   PhD Nikolaos Antoniadis.

Plagiarism Statement

This project was written by me and in my own words, except for quotations from published and unpublished sources which are clearly indicated and acknowledged as such. I am conscious that the incorporation of material from other works or a paraphrase of such material without acknowledgement will be treated as plagiarism, subject to the custom and usage of the subject, according to the University Regulations on Conduct of Examinations. The source of any picture, map or other illustration is also indicated, as is the source, published or unpublished, of any material not resulting from my own experimentation, observation, or specimen-collecting.

PANAGIOTAKOPOULOS, ALEXANDROS

Acknowledgements

This thesis was prepared as part of my bachelor’s degree as a student at the Department of Informatics & Telecommunications in the University of Ioannina. Firstly, I would like to dedicate a moment to thank Charles Babbage the inventor of the first computer and all the people who have assisted throughout the years the field that surrounds the Computer Science and inspired all the future generations. I would also like to thank my family, who taught me to never give up and stay determined until I reach my goals. I would also like to express my gratitude to my friends who have been there for me, and my colleagues Ioannis Masklavanos and Christos Koutsos for their valuable guidance. I also want to take a moment to thank all my university teachers, especially my assistant professor Dr. Doumenis Grigorios, who were by my side and accompanied me throughout this journey of knowledge and teaching me the ways to overcome the difficulties of life and learning. Lastly, I want to take a moment to thank the Autonomous Systems Laboratory, for assisting me with their laboratory equipment for the projects regarding the Internet of Things platform related applications.

Περίληψη

Το Διαδίκτυο Των Αντικειμένων, γνωστό επίσης και στην ελληνική βιβλιογραφία ως «Διαδίκτυο Των Πραγμάτων», αποτελεί μια από τις επιστήμες που συνεχώς εξελίσσεται στον πεδίο της πληροφορικής και θα συνεχίσει να εξελίσσεται με την πάροδο του χρόνου και την βελτίωση της τεχνολογίας. Το Διαδίκτυο των Αντικειμένων διαμορφώνει σε ποικίλους τομείς την καθημερινότητα μας και αποτελεί ένα δυναμικό παράγοντα βελτίωσης του μέλλοντος της ανθρωπότητας όσο και του ίδιου του περιβάλλοντος μας. Το Διαδίκτυο των Αντικειμένων, έχει επιπλέον την ικανότητα να βελτιώνει την αποδοτικότητα των εφαρμογών παράλληλων επιστημών βελτιώνοντας την αποδοτικότητα των εφαρμογών τους με την επέμβαση της τεχνολογίας και των έξυπνων συσκευών. Όμως, για να λειτουργήσει το Διαδίκτυο Των Αντικειμένων ως ιδέα χρειάζεται η επικοινωνία των συσκευών και η αποστολή των δεδομένων σε μια πλατφόρμα ειδικά προσαρμοσμένη για το Διαδίκτυο των Αντικειμένων.

Abstract

Internet of Things also known as “IoT”, is one of the fields of Informatics that is constantly evolving and will continue to evolve in the future, thanks to the improvement of technology throughout the years. The Internet of Things can be seen in many areas of our everyday life and may become a strong foundation for the future improvement of humanity and the very environment that we live in. The Internet of Things can also evolve already existing sciences and make them more efficient thanks to the intervention of technology and smart devices. However, for the Internet of Things to work as a concept, devices need to communicate and send data to a platform that is specifically tailored for the requirements of the Internet of Things.

Contents

Acknowledgements 4
Περίληψη 5
Abstract 6
Contents 7
List of Figures 9
Abbreviation Table 11
1.Theoritical Framework 14
1.1 An intoduction to the term “Internet of Things” 13
1.2 A Historical Evolution of the Internet of Things 15
2.Internet of Things: Building “Oppurtunities” 20
3.Application Domains of the IoT 20
3.1 Agriculture Internet of Things 21
3.2 Industrial Internet of Things & M2M 22
3.3 Green Internet of Things 24
3.4 Internet of Things & Smart Cities 27
3.5 Internet of Things: Smart Applications 29
4 Internet of Things: Similar Concepts Comparison 32
4.1 IoT & Ambient Intelligence 32
4.2 Embedded Computing & Moore’s Law 36
4.3 Ubiquitous Computing 39
5. Necessity for Interoperability and Standards 41
5.1 IoT Interoperability Classifications 43
5.2 Standards for the Internet of Things 45
6. Internet of Things - Platforms 49
6.1 Thinger.io 54
6.2 Microsoft Azure IoT 53
6.3 ThingsBoard 54
6.4 AWS IoT 56

6.5 Google Cloud IoT 54
6.6 IBM Watson 54
6.7 Oracle Internet of Things Cloud Service 61
6.8 OpenRemote 62
7. Creating IoT Smart Applications 64
7.1. Thinger.io Intialization 65
7.1.1 CC3200 Launch XL Initialization with Thinger.io 69
7.1.2 Espressif32 (ESP32) Initialization with Thinger.io. 72
7.2. Microsoft Azure IoT 75
7.2.1 Accessing the Azure IoT Services with IoT Hub 77
7.2.2 Accessing the Azure IoT Services with IoT Central 81
7.2.3 Espressif 32 (ESP32) Initialization with Azure IoT Central 87
7.2.4 Accessing the Azure IoT Services with IoT Central 90
8. IoT Platforms: Data Visualization & Data Storage 92
8.1. Azure IoT Central: Data Visualization & Data Storage 95
8.2. Thinger.io: Data Visualization & Data Storage 97
9. Conclusions 99
Image Sources 100
References 105

List of Figures

Figure 1. Kevin Ashton 14
Figure 2. Auto ID Labs Logo 16
Figure 3. An illustration of the Agriculture Internet of Things 21
Figure 4. An illustration of the Industrial Internet of Things 23
Figure 5. An illustration of the Green Internet of Things 25
Figure 6. An illustration of a smart city 28
Figure 7. An illustration of smart applications that are connected to the Internet of Things ... 30
Figure 8. An illustration of how the Ambient Intellligence works as a general idea 32
Figure 9. A picture of Gordon Moore, author of Moore’s Law 37
Figure 10. A picture illustrating the interoperability between computer systems. 42
Figure 11. IoT taxonomy of interoperability. 44
Figure 12. An illustration of the IoT standards. 45
Figure 13. The official Thinger.io logo (2022) 50
Figure 14. Thinger.io premium subscription plans as of 2022 50
Figure 15. The Thinger.io platform architecture 52
Figure 16. Azure IoT products and services by Microsoft 53
Figure 17. The architecture of Azure IoT Central 53
Figure 18. ThingsBoard official logo 55
Figure 19. The ThingsBoard architecture 55
Figure 20. Amazon Web Services IoT official icon 56
Figure 21. – AWS IoT connecting IoT devices to the AWS Services. 56
Figure 22. Google Cloud IoT official icon 58
Figure 23. IBM Watson official icon 59
Figure 24. How devices operate with the IBM Watson IoT .... 60.
Figure 25. The Oracle Internet of Things Cloud Service architecture 60
Figure 26. OpenRemote official logo 62
Figure 27. The OpenRemote Platform Architecture 63
Figure 28. The official Thinger.io website 66
Figure 29. Thinger.io sign up page 67
Figure 30. The Thinger.io Console 68
Figure 31. Adding a device into your Thinger.io account. 69

Figure 32. The Thinger.io device console 69
Figure 33. CC3200 Launch XL 70
Figure 34. CC3200 Sending the LED States into the Thinger.io platform 72
Figure 35. ESP32 Dev Module....73
Figure 36. Adding Additional Board Manager URLs 74
Figure 37. ESP32 Sending the LED States into the Thinger.io platform....75
Figure 38. Accessing the Microsoft Azure IoT Portal....76
Figure 39. Navigating through the Azure Portal 77
Figure 40.Searching for the IoT Hub through the Azure Portal 77
Figure 41. Creating an IoT Hub....78
Figure 42. Setting Up the IoT Hub 78
Figure 43. Custom IoT Hub created page 79
Figure 44. Creating a device on the Azure IoT Hub 79
Figure 45. Creating unique keys for the IoT Hub....80
Figure 46. Accessing the IoT Central....81
Figure 47. IoT Central, building an application (1/2)....81
Figure 48. IoT Central, building an application (2/2)....81
Figure 49. Adding a Device into the Azure IoT Central (1/3) 82
Figure 50. Adding a Device into the Azure IoT Central (2/3) 83
Figure 51. Adding a Device into the Azure IoT Central (3/3) 84
Figure 52. Importing a Device Template....84
Figure 53. Accessing to the Azure IoT Central Device Keys 85
Figure 54. Completing the Azure IoT Central Set-Up Process 86
Figure 55. Sending Data with the ESP32 into the Azure IoT Central....88
Figure 56.ESPRESSIF 32 C3 DevKitC02....90
Figure 57.Sending Data with the ESP32 C3 DevKitC02 into the Azure IoT Central....91
Figure 58.An illustration portraying Data Visualization 91
Figure 59.An example of Azure IoT Central Dashboard 95
Figure 60.Azure IoT Central – Typical Dashboard Settings....95
Figure 61.The ESP8266 Microcontroller 95
Figure 62.Thinger.io Data Visualization 98

Abbreviation Table

IoT	Internet of Things
RFID	Radio-Frequency Identification
ITU	Internal Telecommunication Unit
PA	Precision Agriculture
ICT	Information and communications Technology
IIoT	Industrial Internet of Things
M2M	Machine-to-Machine
PLC	Programmable Logic Controllers
SCADA	Supervisory control and data acquisition systems
ICT	Information and Communications Technology
GIoT	Green Internet of Things
GIT	Green Information Technology
GC	Green Computing
WSN	Wireless Sensor Network(s)
EPC	Electronic Product Code
AmI	Ambient Intelligence
BSN	Biomedical Sensor Network
PDAs	Personal digital Assistants
HCI	Human Computer Interaction
OTA Programming	Over the Air Programming

PaaS	Platform as a service
MCUs	Microcontroller Units
AWS	Amazon Web Services
AWS IoT	Amazon Web Services Internet of Things
MQTT	Message Queuing and Telemetry Transport
MQTT over WSS	Message Queuing and Telemetry Transport over WebSockets Secure
HTTPS	Hypertext Transfer Protocol Secure
LoRaWAN	Long Range Wide Area Network
IEEE	Institute of Electrical and Electronics Engineers
WPAN	Wireless Personal Network
WLAN	Wireless Local Area Network
WNAN	Wireless Neighborhood Area Network
WWAN	Wireless Wide Area Network
SUN	Smart Utility Network
OFDM	Orthological Frequency Division Multiplexing
LPWAN	Low-Power, Wide-Area Networks
API	Application Programming Interface
W3C	World Wide Web Consortium
HMAC	Hash-Based Message Authentication Codes
ESP32	ESPRESSIF32
ESP8266	ESPRESSIF8266

DMA	Direct Memory Access

1 Theoretical Framework

1.1 An Introduction of the term "Internet of Things"

Figure 1- Kevin Ashton

The term "Internet of Things" as it is commonly referred to in the current bibliography or on the internet as "IoT", comes from the combination of the two words "Internet" and "Things" and was initially proposed by the British technologist, Kevin Ashton in 1999 (Madakam, Ramaswamy & Tripathi,2015). The Internet of Things has the potential to impact every sector of science, such as new product opportunities or factory worker efficiency, shop floor optimization gains and much more. It also affects multiple areas that surround our everyday lives from energy efficiency to remote monitoring, home security and control of physical assets throughout applications and it can be as diverse from home security to condition monitoring (Tripathy & Anuradha, 2020).

According to ITU, as early as 2005, the development of IoT was noted as a function of hyperconnected technological advances from different fields, more specifically, wireless, and mobile connectivity, nanotechnology, RFID, and smart sensor technologies. These technological advances, when combined, operate as a miniaturized-embedded automated Internet of connected devices that communicate regularly, relatively, and effortlessly.

The core vision of what the Internet of Things is trying to achieve, is an interconnection between people (humans) and things (objects) which leads to a successful exchange of information (Garrity,2015) and communication between things and people (Zhang, 2011).

Other companies like Cisco, refer to the Internet of Things as the "Internet of Everything". In their definition, the Internet of Things contains a four-key elements factor that includes: people, data, process and most importantly, Things. In this case "Internet of Everything" can connect:

- People: in more relevant ways
- Data: converting data into intelligence to make better decisions
- Process: delivering at the right time, the right information to the right person (or the right machine)
- Things: which are physical devices and objects connected to the Internet and each other for intelligent decision-making (Rayes,2022).

According to Ammar Rayes & Samer Salam (2022), a more comprehensive definition of the Internet of Things that can be interpreted is that the Internet of Things is the "Network of Things", with device identification, embedded intelligence and sensing and acting capabilities that can connect people and things all over the Internet (Rayes, 2022). The basic promise of the IoT is the remote monitoring of "things" or "objects" from anywhere in the world at any time (Chaouchi & Chaouchi, 2010).

Moreover, as mentioned before, the Internet of Things, can connect objects, and offer new services around objects and people hence the name, as many people can equally refer to it as "Network of Things" or "Network of Objects". Another term worth mentioning, according to H. Chaouchi referred in "The Internet of Things: Connecting objects to the web", is the term "Web of Objects" where the web is the main service accessible to the networked and connected objects. According to H. Chaouchi, the term "Web of Objects" has a more substantial meaning from the application viewpoint, without indirectly implying the extension of the Internet communication model to these new "objects" (or as mentioned before, "things"), in contrast to what the term of the internet of things might suggest (Chaouchi & Chaouchi, 2010).

1.2 A Historical Evolution of the Internet of Things

Figure 2 – Auto-ID Labs logo.

The original the term Internet of Things, as we mentioned in the previous chapter, was first introduced by Kevin Ashton in 1999 during his work at Procter & Gamble. Ashton, who at that time, was working in supply chain optimization, because the internet was a very popular term at the time, his presentation was named as “Internet of Things” with the goal to attract the senior management’s attention to the RFID technology (Uckelmann et al., 2011).

Internet of Things, fundamentally originated in 2003, by the Auto-ID research center at the Massachusetts Institute (MIT) while working on networked radio-frequency identification (RFID), where a substantial effort was made to distinctively point out products. The result that originated was the core steppingstone for the origins of the Internet of Things, (Chaouchi & Chaouchi, 2010) as it was termed with an Electronic Product Code (EPC). Since then, the Internet of Things has been further developed and extended beyond the scope of RFID technologies (Wortmann & Flüchter,2015). After the consolidation of the definition Internet of Things which was used by researchers and practitioners in order to describe the mixture of real-world information and virtual world information technology (Uckelmann et al,2011).

In 2005, the Internal Telecommunication Unit (ITU) showed an interest in new telecommunication business possibilities that could develop services for a new “friendly-connectivity” environment to objects on the network. This function produced an extensive effect on the Internet of Things from ethical, technical, and economical perspective (Chaouchi & Chaouchi, 2010).

The concept of Internet of Things started to gain popularity during the summer of 2010, when a piece of information was leaked that Google’s StreetView service had not only made 360-degree pictures but also stored tons of data that belonged to people’s private Wi-Fi networks. That led to people doubting whether this was the start of a new Google policy to index the physical world. At the same year, the Chinese government made an announcement to make the Internet of Things a strategic priority in their Five-Year-Plan.

In 2015, Europe’s biggest Internet conference LeWeb was the “Internet of Things” and at the same time, popular tech-focused magazines like Forbes, Fast Company and Wired, started using the term Internet of Things to describe this occurrence.

Internet of Things reached mass market awareness in January 2014, when Google announced the purchase of Nest for 3.2 billion dollars. At the same time, the CES (Consumer

Electronics Show) in Las Vegas was held, under the topic of IoT (Lueth, 2015). Due to the recent advances of scaling-down and falling cost of RFID, NFC, sensor networks, wireless communication, technologies, and applications, IoT became the standard for industry and end of users. Detection of the physical status of things through sensors, along with collection and processing of detailed data, enables immediate response to changes in the real world (Uckelmann, et al, 2011).

2. Internet of Things: Building “Opportunities”

The Internet of Things can offer a lot of possible opportunities and at the same time, advance many domains, such as, the Industrial Environment, Agriculture, even the cities as we know them, and many more. In the future, billions of smart sensors and devices will interact without requiring a user to operate them while maintaining a Machine-to-Machine (M2M) basis (Chen & Kung,n.d.). According to Texas Instruments, in recent years, the world has deployed a total of 5 billion “smart” connected things, and a lot of people predicted that the number of connected devices will continue to increase. As the devices become easier to use, the complexity of the devices will equally increase and the difference between analog and digital will become harder and harder to be distinguished.

Internet of Things is constantly evolving and changing, as more and more devices are being added every day offering new capabilities and changing both wired and wireless connectivity technologies in order to meet the various needs of the market. Manufacturers have been connecting things to the internet before internet was even a web that connects people.

In recent years, M2M manufacturers have integrated internet-connected systems that vary from alarm systems, high-value asset tracking to management of fleets for more than 15 years. However, throughout the time, it becomes easier and easier to integrate M2M systems as more powerful processors are incorporated into the end nodes (Chace,2013). Wireless sensors can become an extension of smart sensors with communication, adaptation and learning capabilities that function distinctively. These wireless devices can offer many advantages in terms of cost, flexibility, power options, easy installment, and substitution (Mukhopadhyay,2014).

The growth of the Internet of Things can also create a global market for products and services that can reach trillions of dollars. Furthermore, a lot of entrepreneurship strategies utilize IoT to improve services or products that already exist. However, this method may need a lot of analyzing and devising to improve them (Krotov, 2017).

The number of businesses that utilize IoT technologies have increased from 13% in 2014 to 25% in 2019 and the number of connected devices is expected to increase to 43 billion by 2023. Technological advancement and the increase of the number of devices have also advanced the development of IoT technologies. Investments in IoT technologies are also projected to grow at a rate of 13.6% per year until the end of 2022. Further growth in the upcoming years will also be possible thanks to the reliable mobile connectivity that continues to evolve along with new sensors, and more computing power who are all equally important factors towards the future and improvement of IoT technologies. Furthermore, embedded IoT devices will continue to become cheaper, more advanced, more reliable, and available in the future. The growth of IoT can be seen in multiple different categories like:

- Wide-Area IoT networks, where large geographic areas use connected objects in order to communicate in a low data volume
- Short-Range IoT networks that cover small areas
- Smart phones and Personal Computers and tablets (Dahlqvist et al,2019).

The development of specific IoT application areas strongly depends on several key factors such as: availability in computer electronics, software available solutions with user friendly surroundings, solutions between data and sensors, the overall network quality in which surrounds them and sufficient energy supply for production and operation of IoT devices (Nižetić et al, 2020). The range of applications that can utilize technology regarding the Internet of Things, is enormous and can be found in multiple areas. A few of the areas include, smart energy, medical and health systems in general, home automation, urban infrastructure, intelligent shopping, natural ecosystems, large scale deployments in relation to smart cities and much more (Bibri,2015).

3. Application Domains of the IoT

According to the article “Rise of the Embedded Internet” which was published by the famous company, Intel, the increase of the number of intelligent connected devices will rely on the ability of the internet ecosystem to generate increased income deriving from new products and services and improve existing products, processes, and services. This will present an opportunity to service providers to launch innovative internet-based services, for instance, clients will be looking for convenient ways to observe and direct home systems and reduce utility costs more functionally and accessibly.

This can provide an opportunity to service providers to market and premise service gateways which will link home networks to electrical, water and gas meters, home heating, ventilation, and air conditioning (HVAC) systems, fire and security systems and suppliers of entertainment and media services (Intel,2009).

Internet of Things can provide a lot of fundamental network infrastructure which someone can use and enable the simplest solution between data collection, gathering, exchange and transmission. Furthermore, Internet of Things, with the use of the Internet, can provide a revolutionary way in intelligent decision-making and data processing. A lot of these services may vary from satellite imaging and processing to chicken health and welfare monitoring, that can be cloud based with no human intervention used to sustain them. Lastly, a lot of automated agricultural devices, machines and vehicles will be able to operate accordingly to cultivate crop and livestock in a peerless way (Hassan,2018).

3.1 Agriculture Internet of Things

Figure 3 – An illustration of the Agriculture Internet of Things.

The world population has been estimated to reach 9.7 billion people by 2050, which will result to a substantial demand for food, especially if we double this number and divide the natural resources that continuously diminish. Internet of Things is considered by many, to be the next big step that may have a huge impact on the future of the world. Internet of Things technologies in agriculture practice, will result in a fundamental change on every aspect to pave the way to a new agriculture pattern of the Precision Agriculture (PA) (Elijah, et al, 2018). IoT provides functions that can be used in PA, like local/remote data acquisition, decision-making process, data access, visualization, user-interfacing, in-cloud intelligent information analysis and most importantly, communication infrastructure to interconnect every smart object from sensor (Hassan, 2018).

Precision Agriculture promises to make agriculture extremely effective to assure high production levels and reduce any environmental impact that may occur. Several popular computing companies, like Microsoft, Google, Amazon, and others, provide an IoT cloud platform with benefits such as low price, large scale, and virtualization, among many other useful tools. Furthermore, Precision Agriculture, uses sensors like RFID, wireless communication, intelligent systems, and other Information and Communications Technologies (ICTs), to monitor and control the systems. Moreover, IoT devices can relay the data to other IoT devices via the internet and the IoT platforms are able to cover the difference between the data network and device sensors (Abbasi & Yaghmaee,2019).

3.2 Industrial Internet of Things & M2M

Figure 4 – An illustration of the Industrial Internet of Things.

As a member of the Internet of Things, the Industrial Internet of Things or IIoT for short, covers the domains of Machine-to-Machine (M2M) and industrial communication technologies with automation applications. The Industrial Internet of Things, facilitates the way for a better comprehension of the manufacturing process, therefore making a well-organized and renewable production (Sisinni, et al,2018).

The term M2M or Machine-to-Machine has been in use for a lot more than a decade and is very common in the Telecoms sector. M2M communication had originally been a one-to-one communication, linking one machine to another. Today's technological advancements made the transmission of data easier, via a system of IP networks to a much wider range of devices. The key difference between M2M and Industrial Internet of Things is that M2M only focuses on machines, while IIoT also focuses on human interfaces along with machines (Lueth,2015).

Another key difference between the Internet of Things and the Industrial Internet of Things, is that IIoT faces challenges that are more distinctive and different from other systems and services of IoT. This happens due to the requirement to integrate Programmable Logic Controllers (PLC) and Supervisory Control And Data Acquisition systems (SCADA), along with the affiliated industrial networks that interconnect them, which constitute the infrastructure of operational technology (OT), that have traditionally evolved separately from the typical IT technology due to the fact that it addresses the requirement of systems in the field like industrial floor, energy production facilities, energy distribution networks and so on (Sepranos, 2019).

Industry 4.0, or as Dimitrios Serpanos (2019) & Marilyn Wolf name it “Industrie 4.0” along with Ulrich Sendler and many others, is a strategic initiative in Germany that aims to lead the way for IoT technologies to the manufacturing and production sectors with the main objective to enable Germany to keep a leading role in manufacturing achieving efficient and low-cost production with workflows that are flexible. Essentially, the Industrial Internet of Things is a generalization of Industrie 4.0 which appears to focus more on industrial process efficiency. IIoT also includes all aspects of industrial operations, focusing not only on effectiveness of operations by also on maintenance, management etc. (Sepranos, 2019).

According to Ulrich Sendler, Industrie 4.0 is based on the technical possibility of connecting products of all sorts to the internet that are software-controlled and are equipped with digital components and offer corresponding services via the internet. There are a few technologies that have been available for a much longer time but have received a completely new significance with reference to the Internet of Things and Services (Sendler, 2018).

3.3 Green Internet of Things

Figure 5 – An illustration of the Green Internet of Things.

Energy consumption levels in the past decade, have reached alarming rates due to the large scale of digital context, number of subscribers and the number of devices. The rise in the number of devices is predicted to reach 100 billion by 2030. Therefore, a massive data rate is expected by scientists along with huge content-size at the price of carbon emissions into the environment. Furthermore, multiple concerns were raised regarding the overall health, and the immense amount of CO2 emissions, along with environment concerns for the research of renewable or green technology that may offer multiple benefits and opportunities. With the thought of obtaining a sustainable smart world, Internet of Things should focus on maintaining the energy efficiency to reduce the green house effects and carbon dioxide (CO2) emissions that originate from sensors, devices, applications, and services (Zhu,et al,2015).

A challenge that the Green IoT (or GIoT) will have to deal with, is considered to be the requirement to make it more and more sustainable and able to face certain challenges. As an example, the number of sensors that will be used and the amount of information that will need to be generated. Another issue that Green IoT will have to surpass, will be the fact that the communication process generally consumes more power and an IoT device needs to operate on a limited energy plan. A few more issues concerning the delay of deploying Green IoT, are the employment and adoption of the software and the hardware that can reduce the energy usage, progress and collaboration and consistency of the technologies, that are able to recognize and verify privacy and security (Bashar, 2019). Moreover, the GIoT data is gathered from energy-efficient objects and sensors that are embedded in devices. Therefore, it is obligatory to use the right sensors to enhance the lifespan of the devices and receive as a result green energy consumption. However, to establish a green intelligent network several challenges that are on multiple fields of our lives such as: technology, energy measurement & management, transportation, cities, and health monitoring will be required to become intelligent (Varjovi & Babaie, 2020).

Energy harvesting is also playing a significant role in Green Internet of Things to gain true energy independency. When true independence is reached, the only problem that may occur will be the lifespan of the battery that sustains a system or a device, however theoretically speaking, devices could potentially be self-sufficient for years to come (Estevez & Wu,2015).

Another great fact about Green Internet of Things, is that it's based on Green Computing (GC) or as it is more commonly known as Green Information Technology (GIT), which is the study and practice of environmental sustainability computing or IT. Green Computing, encompasses the study and the practice of design, use, manufacture, and the disposal of computer components effectively and efficiently with very few efforts required or no negative effect on the environment (Thilakarathne, et al, 2012).

A few of the key areas of the Green Internet of Things involve the following:

- **Green RFID tags**
    - Reducing the size of an RFID tag can assist in the reduction of the energy consumption
    - **Green Cloud Computing,**
    - Hardware and software are used in such a way to reduce energy consumption
- **Green Internet Technology,**
    - Special hardware and software that are specifically designed to perform without performance reduction and consume less power
    - **Green Data Centers,**
    - That are responsible for storing, processing and managing all types of data and applications should be designed with the core idea the usage of renewable energy resources
- **Green Wireless Sensor Network,**
    - Which can be achieved by applying green energy conservation techniques, radio optimization techniques and green routing techniques to reduce energy consumption in WSN.

As mentioned before, considering how much Internet of Things has grown and will continue to grow, applying Green Internet of Things Technology might help us realize the future that we should all strive to reach and with the help of the green practices, the benefits that we can all gain will be immense.

3.4 Internet of Things & Smart Cities

Figure 6 – An illustration of a smart-city.

Another great domain that is commonly used with IoT Technologies are the Smart-Cities along with smart technologies and smart infrastructure involving them. Smart cities are greener, safer, faster, and friendlier. It is worth mentioning that the smart city is a concept idea and there isn't any common, specific, or consistent definition among practitioners and academia, however multiple and different definitions and explanations can be given regarding what a smart city is or what it tries to accomplish.

One explanation that can be given, is that a smart city is a place where traditional networks and services are more sustainable, flexible, efficient with the utilization of information and technologies that can improve the city's operations for the benefit of its residents. A few of the key components that a smart city includes are smart infrastructure, smart transportation, smart health care, smart energy and most importantly, smart technology –all the mentioned components can assist in the making the cities smart and efficient (Mohanty et al, 2016).

Another definition for a smart city that can be given, is the interoperability between the physical infrastructure, the social infrastructure, the ICT infrastructure, and the business infrastructure to maintain the collective intelligence of the city. Internet of Things plays an important role in connecting everything together and to the internet with the goal, the successful exchange of communications and at the same time achieve location, tracking, monitoring, and management.

A few of the key pieces that play a significant role in the vision of smart cities, are smart retail and healthcare. Online retail and healthcare can be supported by IoT technologies and evolve. Many users with disabilities have been increasing in the US and around the world in the last few decades, therefore, research has been conducted to achieve balanced and healthy societies by giving an easier access to health services through technological advancements. For example, in smart cities, the status of a patient can be monitored remotely to provide faster and more reliable workflows in hospitals, locations of available ambulances can be continuously monitored and intelligently routed, connecting wheelchair users with the internet to increase access for online shopping and communication, even a project involving the assistance of the blind with a smart "E-cane" system has been identified (Qian et al,2019).   Another important domain of Internet of Things technologies and Smart Cities are the Smart Homes. A smart home is a combination between rapid advancements in technologies and improvements in architecture. The actualization of smart home can become a massive step for humanity to accomplish intelligent control of electrical appliances and lightning and intelligent notification of home alarm messages (Gaikwad et al,2015). Simultaneously, whether indoors or outdoors, we can receive benefits from the information and technological achievements of smart city (Su et al, 2011).

Single projects help make the smart city project a reality. However, the projects should have specific characteristics, for instance, the usage of advanced technological solutions, harmonizing the environment, improving the economy, and fulfilling the citizens' requirements and expectations. For the time being, smart projects are mostly focused on specific topics, for example, buildings, energy efficiency, greenhouse gas reduction, e-services delivery etc (Dameri,2017).

3.5 Internet of Things: Smart Applications

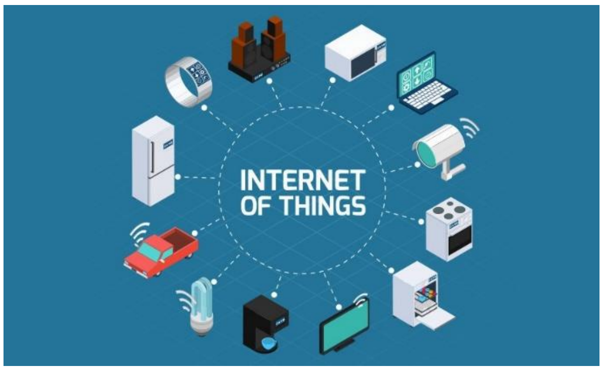


Figure 7 – An illustration of smart applications that are connected to the Internet of Things.

An IoT solution for smart applications requires unique software, hardware and communication technologies working together to integrate an assortment of platforms, components, and applications (Zyrianoff et al, 2019). IoT applications are essential for the transmission of data generation by the devices or sources to the Internet since either vendors or users aim to collect data and further analyze it to enhance their devices. In the more recent years, industries are continuously developing wired or wireless communication channels and protocols. However, the cost and infrastructure development are what may become a burden towards the development of technology for the Internet of Things (Balaji et al, 2019).

As mentioned before, a smart home is a key example of what the Internet of Things vision can accomplish. It uses the IoT infrastructure to connect and monitor several sensors that can sense and collect surrounding information that can be used to fully control different home systems like lighting systems & security systems. A few other examples of what smart applications and IoT are capable of accomplishing, are smart tunnels and smart bridges. For example, a smart bridge can use temperature and vibration sensors as well as video surveillance cameras to detect any unusual activity that may occur and send warnings via SMS. On the other hand, a smart tunnel can utilize multiple sensors to monitor temperature, humidity, displacement, to detect if a problem occurs (Yasser,2019).

With the proper handling of the IoT technology, e-Health among other technologies that IoT focuses on, can also be exploited. For example, a person might have fallen of the stairs, the accelerometers of the person’s phone can instantly recognize that something dangerous might have happened and then check sensors are initialized to check the person’s condition. At this hypothetical scenario, both devices agree that something has indeed happened, and the person may be in danger, the next step, is for the phone to release an emergency message which sends the location data of where the person fell, as well as his credentials to the nearest emergency center, where an ambulance is dispatched immediately.

Furthermore, a physician will be able to connect to the person's phone to verify that something painful has indeed happened. As soon as the ambulance arrives, the person will be able to be identified, with all the necessary medical data which can help the doctor be prepared beforehand. This example showed how much useful IoT technologies can become, especially when the topic is e-Health. From this supposed scenario, we can conclude that there are multiple key components that IoT applications, especially in e-Health, must deal with: Interoperability, Bounded latency and reliability and privacy authentication, and integrity (Bui & Zorzi,2011).

Another sector that was mentioned before, which can receive a benefit from IoT applications, are the smart cities. One of the many examples, that can be mentioned is the Smart Traffic. The primary objective of IoT is the connection of "Things" in real-time like all traffic information can be updated at simultaneously and easily. Smart Traffic can make an important impact by routing and collecting information from sources like sensors, cameras, GPS, etc. For example, real-time GPS information can assist the traffic light management system to safely direct cars to avoid traffic jams. At the same time, remote users will be aware of the shortest part and minimum time to reach the destination.

Moreover, traffic control camera system can provide more information using processing technique about the traffic movement. All the data acquisitions can assist the user to make the best possible decision which can reduce traffic violence, traffic accidents and enhance the security of the traffic system (Rabby et al, 2019). Researchers have been also interested in applying to the smart traffic system using IoT applications. IoT can lead to multiple advantages such as utilizing the maximum data users extracted from every single user and is also ecosystem friendly. In 2018, around 367 projects were accepted in megacities, in which, IoT is considered in the traffic management to refine the overall health of the city and environment and peoples' lifestyle (Su et al, 2011).

The main concern that can be raised by many people are the privacy issues. Internet of Things devices may collect data including sensitive data and later store it for further use for commercial purposes (Qian et al, 2019). This leads to many people theoretically suspect that IoT devices may collect users' personal information without the users' permission or even not give a proper notice to them when a potentially sensitive information is being collected which can make someone question their security and their privacy (Psychoula et al, 2018).

4.Internet of Things: Similar Concepts Comparison

4.1 IoT & Ambient Intelligence

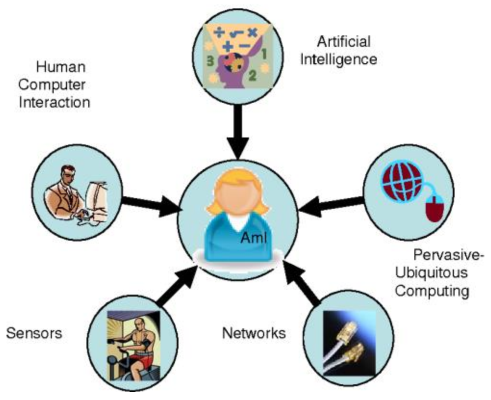


Figure 8 – An illustration of how the Ambient Intelligence works as a general idea.

Ambient Intelligence or AmI for short, is all about sensitive and adaptive electronic environments that respond to the actions of persons and objects and satisfying their needs, this method includes single physical objects and associates it with human interaction (Aarts, & Wichert,2009). The concept of Ambient Intelligence is based upon the pioneering work by Mark Weiser on ubiquitous computing. Weiser’s general idea was to make computers effectively invisible to the user by having them distributed throughout the physical environment. Another pillar behind the concept of Ambient Intelligence is the work based on intelligent social user interfaces by Nass and Reeves, where they wrote the book “The Media Equation” and discovered that human attributes like emotion are aspects of human behavior are equally important in order to apply the interaction of man with modern machines (Mukherjee et al, 2006).

Ambient Intelligence involves multiple computing areas, the primary area, is named “Ubiquitous or Pervasive Computing” and it is all about the contribution of various ad hoc networking capabilities that explore highly portable or multiple, low-cost computing devices. The second equally important area, named “Intelligence Systems Research”, provides learning algorithms and pattern machers, speech recognition, language translators, gesture classification and situation evaluation. The third key area is context awareness, which allows the tracking and positioning of all-type objects and at the same time, represent how the objects interact with their environment. The final area is the appreciation of social interactions between objects in an environment (Shadbolt,2003).

Ambient Intelligence, strongly focuses on the vision that technology will become invisible, present whenever and wherever we may require it, embedded in our natural surroundings, adaptive to each user, attuned to all our senses and most importantly, autonomous. AmI gains attributes by an environment where the technology is embedded, hidden in the background, is adaptive, sensitive, and responsive to the people and objects by extending its capabilities using non-explicit objects. Lastly, that preserves both security, trustworthiness and privacy at the same time as utilizing information when needed and when they are appropriate (Weber et al, 2005).

As the latest waves of science-based technologies & innovations, AmI, had emerged from Europe while the IoT originated in the USA. They both equally offer extraordinary results towards the future of the ICT, that closely resemble with different terms to each other. For example, a few closely similar terms that may occur can be: ubiquitous computing, everywhere computing, sentient computing, wearable computing, pervasive computing, invisible computing, affective computing, Things that think etc. Even though, the term AmI is used in Europe there is an equivalent term that is used in the USA named "Ubiquitous computing", while in Japan the most commonly used term is "Ubiquitous networking". In essence, all these terms apply the same principles: researchers in any location, develop similar technologies, imagine similar future worlds and deal with similar challenges.

According to Simon Elias Bibri (2015) both the Internet of Things and the Ambient Intelligence, envision a future with an overflowing number of objects that are smart, able to interact with everyday connected objects and a whole range of opportunities and possibilities. The vision of the Ambient Intelligence is all about the integration of tiny microelectronic information processors and networks of miniature sensors and actuators into everyday objects and the vision of the Internet of Things involves interconnecting uniquely identified embedded devices and physical, virtual, and smart objects within existing the internet infrastructure (Bibri, 2015). Both IoT and AmI can yield numerous results for citizens, consumers, industries, businesses, and the public at large by protecting the environment, and improving the economy thanks to their technological advancements in novel applications, products and services that provide advanced performance and value.

As mentioned before, both AmI and IoT are expected to enhance the automation in nearly all fields. More specifically, in Ambient Intelligence, computing devices which can communicate and think, are becoming cheaper, smaller in terms of size, more sophisticated, more capable, smarter, better interconnected, and easier to use. The range of applications that utilize AmI technology, especially in areas like work, education, healthcare, community buildings etc. is potentially massive.

Likewise, as mentioned before, IoT could find applications in countless areas of our daily lives, thanks to the ability to network embedded devices and everyday physical objects withing the ever-growing Internet infrastructure. In short, a world where the Ambient Intelligence and the Internet of Things technologies can co-exist, will allow people and everyday objects in the physical world and what entrails in terms of the virtual and information worlds to interact together to create all types of smart environments (Bibri, 2015).

Even though both the Internet of Things and the Ambient Intelligence can yield great gains, it is estimated that due to the rise of connected devices concerning the Internet of Things, several vulnerabilities will increase equally. A few of the key issues that can raise concerns include (Makhdoom et al, 2019):

- **Security and privacy**, a lot of devices gather personal information about the users without their users' permission. This can lead to data security, privacy, and integrity attacks.
- **Device integrity**, a problem that may occur is that IoT devices operate in a trustless environment without any physical security and can become subject to attacks including, side-channel attacks, invasive hardware attacks, side-channel attacks, and reverse engineering attacks.
- **Software/Code integrity**, they are equally crucial to maintain the security and privacy of the things
- **Threats to eHealth IoT devices**, Biomedical Sensor Network (BSN) is a specialized case of WSN in which sensors are used to monitor each patient's health and generate chronic disease self-care. Because BSN has dynamic network topology due to mobile nodes, power constraints and low bandwidth IoT communication protocols, they are vulnerable to numerous attacks like Denial of Service (DoS), eavesdropping etc (Makhdoom et al,2019).

Internet of Things isn't the only thing that can raise multiple concerns though, Ambient Intelligence can also equally raise multiple issues that need to be addressed. A few of the issues that can be raised by using Ambient Intelligence are (Ahonen & Wright,2008):

- **Human factors and security**: AmI can help to overcome difficulties when it comes to human factors, but human failings can't be faded out completely. Since some of the devices, when it comes to the smart-home concept idea will be required to be accessed by a password or verification, a lot of security concerns can be raised

- **Loss of control**, when AmI does not function like expected, it might cause people some frustration and rejection (switching off a device). Annoyance can be foreseen when Ambient Intelligence does things that are not wanted or expected, even if it was authorized willingly (or unwillingly).
- **Insider ID theft**, ID theft is possible without criminal interactions, people who know each other can also breach privacy once an ID is misappropriated, it becomes easier to spend money considering payments are becoming more and more automated (at least up to a limit).
- **Exclusion**, not everyone will be able to have access to all the services that AmI can offer (Ahonen & Wright,2008).

4.2 Embedded Computing & Moore’ Law

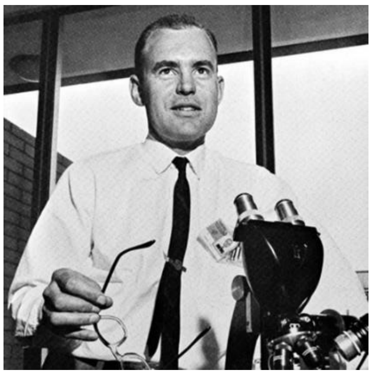


Figure 9- A picture of Gordon Moore, author of Moore’s Law.

Moore’s Law is considered by many to be one of the most compelling developments in embedded technology or AmIware in short, Moore’s law rules the semiconductor industry for more than thirty years now and provides a widely accepted forecast for the development of semiconductor technology (Aarts & Ruyter,2009). By extending upon the different developments of Moore’s law we can conclude that the manufacturing and design of electronic devices has overall reached a height of miniaturization which allows the integration of electronic systems into any physical object like cars, clothes, furniture etc. with processing, communication, storage, and display capabilities (Mukherjee et al, 2008).· In 1965, Gordon Moore of Fairchild Semiconductor (currently named Intel, which is short for Integrated Electronics), built an integrated circuit with 30 transistors and was finalizing a component with 60 transistors, it was at the time, that he made the prediction that the number of transistors on a chip could double per year, for the next decade. Since then, he amended the law to a doubling every 24 months, and later to 18 months. This prediction of exponential growth became known as Moore’s law. Since then, we have advanced from 30 transistors on a single wafer, to more than 55 million in less than 30 years.

An interpretation of Moore’s law that can be made, is that every 18 months for every chip size the cost, halves, for instance, in 1965 the cost of a single transistor was about 5$, today about 5 million transistors can be bought for a mere 5$. Another interpretation of Moore’s law that can be made, is also being abstracted to factors beyond just the number of transistors which applies to both computational capability and storage capacity, for instance, data storage density is doubling faster than Moore’s law at every 12 months (Payne & MacDonald,2006).

The variety of IoT applications and technologies make it difficult to present a general overall statement for the requirements of IoT in hardware and software, hence, the IoT embedded designed faces questions whose answers are challenging, for example, what trade-offs can be made between Quality of Service (QoS) and energy consumption, which wireless technology can cover the range that is required depending on the application etc (Samie et al, 2016). Embedded computer systems are also required to reach a height of computing performance and their characteristics such as power consumption, mobility use and size. Moore's law and the associated shrinking of transistor sizes and overall size, increase in mobility, and their power consumption also serves as a foundation for the ubiquity of embedded systems (Payne & MacDonald, 2006).

However, transistors can face certain challenges, for example, in order to provide efficient noise immunity, the supply voltage of transistors is not scaled down at a rate which is equal to the reduction of the overall size of the transistors. This can lead to a significant result in power density as the size of the transistors decrease without a corresponding decrease in power consumption. Several approaches such as this, have been adopted to address Moore's law (Palem, et al, 2009).

Another important aspect when we are referring to the Internet of Things, is the field of embedded computing. Embedded computing is very different than general computing, considering that the Embedded Computing systems require a lot more consumption, but at the same time are required to meet certain goals, such as, real-time performance, not simply average performance, power/energy consumption and lastly, cost. Considering there isn't one system that is best fit for everything, different implementations must be made to cover the different requirements between a family of applications. Another key difference when it comes to embedded computing and general-purpose computing systems, is the difference between the design of hardware and software. In general-purpose computing systems, the design is separate between the two, while on the other hand, embedded computing systems can design hardware and software simultaneously. Furthermore, designers of general-purpose computers use a more narrowly defined hardware design methodology that use benchmarking as inputs to track and simulate, while embedded computing designers need more complex methodologies because their system design encompasses both hardware and software (Wolf, 2007).

Understanding the requirement of the applications, we can use different characteristics in order to optimize our design how we see fit, but that also means that we should have enough knowledge of the application to benefit from the characteristics that we use to avoid problems for system implementers. Embedded computers may also be required to handle critical data such as purchasing data or medical information (Wolf, 2007).

IoT applications as mentioned before, implement both hardware and software. IoT is generally classified in two main categories: wearable devices and embedded system boards. Wearable devices are more limited and depend on the design of the application, for instance, a smart watch may be designed for health tracking and call receiving only, though, its main advantage is that it's handy to use. A few more examples of wearable IoT devices include, Samsung gear, Google glasses, smart wallet, among many others. On the other hand, embedded system boards are more open to the user, depending on the requirements and needs of the user (Kotha & Gupta, 2018).

In today's market embedded system boards are widely available with multiple systems to choose from like Arduino boards, ESP8266, ESP32, Samsung artik board, raspberry pi boards and so on (Kotha & Gupta, 2018).

4.3 Ubiquitous Computing

In modern era computing, the term Ubiquitous Computing has risen. A few other terms for Ubiquitous Computing can be: Ubicomp or Pervasive Computing (Krumm, 2018). The term Ubiquitous Computing is characterized by the explosion of small networked portable computer products in the form of smart phones, Personal Digital Assistants (PDAs) and embedded computers that are built upon many of the devices that we already own, where a person owns and uses many computers (Ebling, 2016).

The original term of Ubiquitous Computing can be traced back in 1988, in Xerox PARC by Mark Weiser where he envisioned a future in which, computing technologies will become embedded in everyday objects and they were also applicable to our daily activities like our work, our home management and for our personal amusement. The essence of Weiser's vision towards the future is that mobile and embedded processors will be able to communicate with each other and the surrounding infrastructure (Krumm, 2018).

However, considering how similar the two terms Internet of Things and Ubiquitous Computing are, a question can be raised whether we are talking about Ubiquitous Computing or the Internet of Things. Considering how IoT and Ubiquitous Computing examine similar issues and face similar challenges, one might suggest that Ubiquitous Computing is focusing more on the Human Computer Interaction (HCI) issues and making the connected things disappear from human attention, while on the other hand, IoT focuses more on connecting the devices. That would be one possible key difference that one might argue however, the IoT community is also focusing on HCI issues. Both the IoT community and the Ubiquitous Computing community, are pursuing similar concepts, like smart cities, environmental monitoring, agriculture, home automation and so on. Plus, both communities share closely similar interests and goals –if not the same (Ebling, 2016).

The main objective of Ubiquitous Computing is to make smart objects by making a sensor network capable of collecting, processing, and sending data. Context awareness is used as a part of Ubiquitous Computing's concepts and wearable systems. Another key difference between the Internet of Things and Ubiquitous Computing that can be mentioned is that Internet of Things has grown out of the scope of Ubiquitous Computing, with very little to no differences between the two of them (Alshqaqi et al, 2019).

Pervasive Computing can also face serious challenges that are equal if not the same, to the Internet of Things. Most devices, have strong limitations on memory usage, processor performance and power consumption. Pervasive applications also need to deal with various hardware and software platforms and different form factors and user interfaces, however, as technology continuously evolves, new technology is prepared to eliminate the limitations.

Ubiquitous Computing will also bring us many possibilities like a productive new work style, incredible communication possibilities, staying in touch with everyone from everywhere and information which will be accessible whenever it is needed. Plus, when new technical possibilities arrive, new genres of applications and services arise, that can bring great opportunities for individuals and businesses alike (Hansmann, 2011).

Even though, technology is constantly evolving, and we use multiple devices related to Ubiquitous Computing, we can't say for certain at least for the time being, that we reached this vision yet. In order to fully achieve the Ubiquitous Computing vision, the Ubicomp software must delivery functionality in our daily lives. Furthermore, Ubicomp software must operate in conditions of radical change, considering different physical conditions may cause components to routinely make and break associations with peers and in terms of functionality (Kindberg & Fox,2002).

5. Necessity for Interoperability and Standards

Figure 10 – A picture illustrating the Interoperability between computer systems.

According to Nancy Ide and James Pustejovsky, Interoperability can be defined as a measure of the degree to which diverse systems, including organizations systems, and/or individuals are able to work together so they can achieve a common goal (Ide & Pustejovsky, 2010).

Interoperability in IoT can be described as the ability to communicate between the devices regardless of hardware and software. Thanks to interoperability, systems can interact with each other despite different specifications and different manufacturers (Rana et al,2020). Another definition for Interoperability according to Institute of Electrical and Electronics Engineers (IEEE), is the ability of two or more systems (or components to exchange data and use information (González-Usach et al, 2019).

The main component in order to exchange information and work together in an IoT system is interoperability. Without interoperability, smart phones would not be able to read sensors or access a network and so on. Furthermore, according to a study that was carried out by McKinsey Global Institute in 2015, at least 40% of the potential gains of the Internet of Things would not be possible (González-Usach et al, 2019). Interoperability for IoT devices can become a complicated setup that may face multiple issues. First and foremost, Security, can face a challenge. As the rise of IoT devices increases and will continue to increase, IoT devices are becoming smarter, which can lead to multiple privacy issues as mentioned before. This, combined with interoperability, can make things more complex, considering IoT devices work on different platforms with different standards that can make the security of communication a burden. Proprietary Technology can also become a burden for interoperability, for instance, propriety software companies like Apple, do not want interoperability so they can have a market advantage over their consumers, therefore, open Application Programming Interfaces (APIs) of their products is not likely to be supported (González-Usach et al, 2019). Resources can also play a fundamental role when it comes to interoperability between IoT devices in order to maintain open APIs, a requirement of resources is needed to test the API for common mechanisms and at the same time, monitor the devices using a device monitoring layer, security of the devices for example, can consume resources, along with push and pull of the information from IoT devices (Konduru & Bharamagoudra, 2017).

Heterogenous devices, can also become a main issue when it comes to IoT interoperability. Considering Heterogenous devices contain varying data representations and APIs, pose an obstacle in communication between IoT devices due to a lack in data semantics and common standards are needed in order to compute the meaning of data· Another issue that may occur is the existence of isolated systems that have a lack of interoperability among platforms, due to the lack of interoperability the systems stand isolated (Konduru & Bharamagoudra, 2017).

5.1 IoT Interoperability Classifications

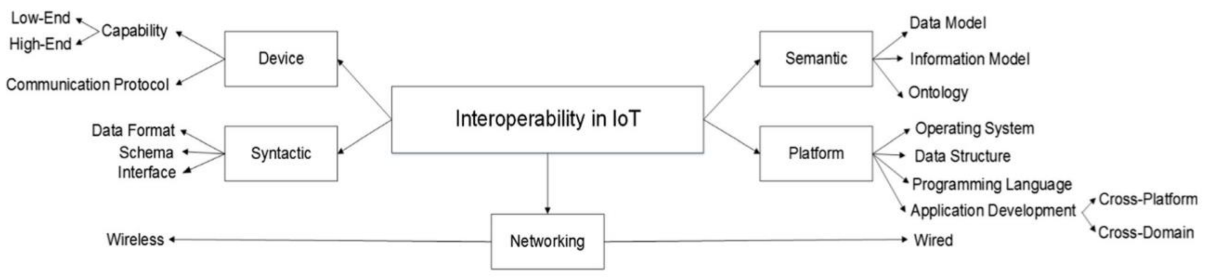


Figure 11 - IoT taxonomy of interoperability.

There are multiple classifications that can be made when it comes to the Internet of Things interoperability. First and foremost, there is *device interoperability,* which is created from a variety of devices that are called “smart things/objects”. These devices, are distinguished between low-end devices and high-end devices depending on the resources that may be contained, for instance, computation capabilities, energy consumption capabilities, communication capabilities, sensors, and actuators capabilities and so on (Noura & Gaedke,2018). *Technical Interoperability* is all about the capability in which the systems, the system components or the applications can establish a connection and share messages without necessarily understanding their content, in this level, there is no data awareness and meaning and network connectivity may be required. Technical interoperability has a relation with M2M communication and elements (González-Usach et al,2019).

Network Interoperability is another classification for IoT interoperability. Network interoperability requires mechanisms in order to enable seamless message exchanges between systems through different networks for end-to-end encryption. Network interoperability must also face challenges such as routing, addressing, optimization, security, QoS and mobility support(Noura & Gaedke,2018).

Syntactic Interoperability is used in any information exchange or service between heterogeneous Internet of Things systems. Another definition for the Syntactic Interoperability can be the ability of the systems to decipher the message structure of the exchanged information (González-Usach et al,2019). This interoperability relies on data formats, as the exchanged messages require a data representation that is common between the two systems that exchange the information. Data format such as XML, JSON or CSV can assist the prevention of ambiguity in the interpretation of the data (Noura et al,2018).

In Semantic Interoperability, the systems are considered capable of interpreting the content and the meaning of the exchanged information. Another definition that can be made when it comes to semantic interoperability, according to World Wide Web Consortium (W3C), is that Semantic Interoperability enables agents, services, and applications to exchange information, data, and knowledge in a meaningful way, on and off the Web(González-Usach et al,2019). An example that can be made again, is the concept idea of smart cities that has correctly extracted the data received from another system and is able to understand the meaning and context information contained in the data, then the system can be aware of the extracted values. In this case, the system can become capable and utilize information in the proper context (Noura et al,2018).

5.2 Standards for the Internet of Things

IoT standards talk was initiated in 2013, but it was considered too late by then, due to the advancement in the technological industry. The general idea of IoT Standards is the development of an IoT framework that can encompass all the layers with connectors, covering entities, things (or devices), and networks to form a machine-to-machine (M2M) and Web Standard. A lot of challenges and issues may occur when the effectiveness and performance of the IoT. Some of these challenges can be, communication challenges, data management, scalability, collection of big data, security and privacy, interoperability, and so on. The process of standardization ensures interoperability, which in turn will enhance interoperability, successful integration and sharing information between distributed systems and security. This in turn, can also mean that there is a need to have standard protocols and platforms that are able to connect multiple different devices with different manufacturers to communicate with each other(Al-Qaseemi et al,2016).

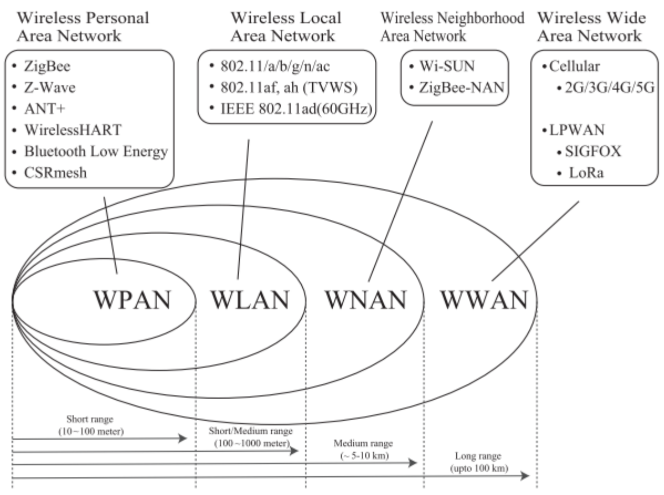


Figure 12 – An illustration of the IoT standards.

IoT requires a variety of different technologies, which are classified into WPAN (Wireless Personal Network, WLAN (Wireless Local Area Network), WNAN (Wireless Neighborhood Area Network) and WWAN (Wireless Wide Area Network) which are all described below (Naito,2016):

❖ WPAN / WNAN

IEEE 802.15.4 is a standard which in it comes for low-rate Wireless Personal Area Networks (WPAN) and has an emphasis on low power consumption and low transmission rate (250 Kbps) for better battery optimization. IEEE 802.15.4g, which comes from and extends it, IEEE 802.15.4 is for low-rate transmissions in Wireless Neighborhood Area Network (WNAN) and facilitates very large-scale process control applications (Naito, 2016). An application example can be Smart Utility Networks (SUN) that have minimal infrastructure and many fixed end-nodes in order to meter electricity, gas etc. Furthermore, IEEE 802.15.4g builds a smart WNAN and extends the data payload size (2.048 Bytes). IEEE 802.15.4k is a physical layer for low energy critical infrastructure network. It has a low transmission rate (<10 Kbps) and a distance communication (1 km). Many alliances develop IoT products based on IEEE 802.15.4. ZigBee, Wi-Sun, Thread, Wireless HART, Z-Wave, CSRmesh, are keynote examples of such technologies (Naito, 2016).

❖ WLAN

IoT devices, in order to connect to the internet, use a communication function. The standard for WLAN is Ethernet also known as, IEEE 802.3. Some Internet of Things devices employ Ethernet to connect to a network when the devices are fixed in a facility because power over Ethernet (PoE) can also provide electric power to devices(Naito, 2016). As a result, recent IoT devices employ a wireless communication device in order to connect to the internet. IEEE 802.11 ad or Wireless Gigabit Internet (WiGig) supports 7 Gbps, it’s main purpose is to provide enough throughput performance in a limited area, IEEE 802.11af employs a similar Orthological Multiplexing (OFDM) technology and will be a main stream standard to provide high throughput performance over TeleVision White Space Frequency spectrum (TVWS), IEEE 802.11ah is similar to the IEEE 802.11af because it also utilizes TVWS technologies, with the only difference, that IEEE 802.11ah focuses on long distance with low power consumption(Naito, 2016).

❖ WWAN

Typical IoT services are employing also cloud services to provide information to IoT-end-devices, this makes Wide Web Area Network communication a necessity to utilize IoT services. This contains 3GPP (Third Generation Partnership Project) which has developed standards for cellular network systems, usually 4G and 5G cellular systems contain high-speed communication, low-speed communication, and low power consumption.

GSMA/eSIM, provides standard mechanism for remote provisioning and controlling of M2M connections. However, because cellular devices require a physical SIM card to connect to a network, a SIM card along with the slot are needed to be installed on the device. LPWAN (Low-Power, Wide-Area Networks), support long distance communication technology, along with low data rate and low power consumption, a few notable examples are LoRa and SIGFOX(Naito, 2016).

6. Internet of Things - Platforms

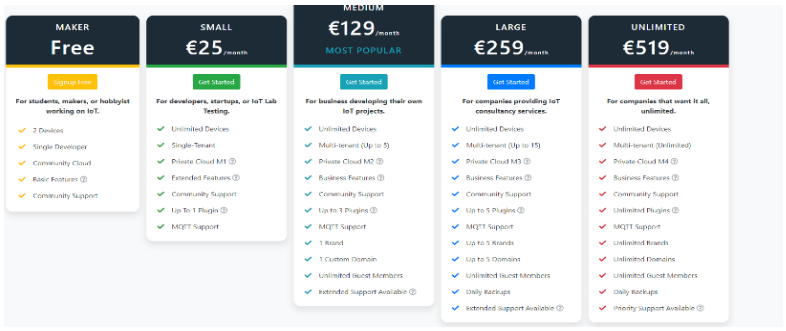
This chapter presents information about a few of the platforms that are suitable for IoT developers depending on their requirements, their budget, capabilities, additional support, and extra features that the developers may or may not require. A platform is an environment which allows the programmer to deploy and run the application (Nakhuva & Champaneria,2015). A platform can both be a hardware and software suite upon which applications can run. A utilization of both hardware and software can be contained in a platform in which the Operating system can reside. The Operating system can in turn, allow an application to work above the platform by providing necessary execution environment to it (Nakhuva & Champaneria,2015).

When an individual or a company faces the decision to choose among platforms based on their project requirements, the different protocols that the platform can offer, which can help towards the accomplishment of their project's vision, their own skill on working on IoT platforms and different protocols, and most importantly their equipment and their budget, along with picking specific benefits and make a specific platform an ideal choice over a different one. There are also many unique IoT platforms that can offer different privileges and fulfil distinguish requirements, and at the same time, some of them if not all, will continue to add many more benefits and improvements after the time of this thesis. This thesis presents the most popular and appealing platforms, both to companies, individuals and people who are deeply interested in getting started with IoT platforms. It introduces some of these IoT Platforms, and compares them based on their benefits, their pricing, their supported protocols, their device support, their project vision, and finally, various benefits that may or may not be included in their platform and answers the question of "*why should someone choose this platform over the other one?*" and *"what benefit can this platform offer to my company instead of the other one?"*. This dilemma can lead us to different answers as many platforms offer different benefits and advantages to a company, an individual or a person who is just getting started and all of them have different requirements that require an IoT platform to fulfil.

6.1 Thinger.io

Figure 13- The official Thinger.io logo (2022)

The first IoT platform that this thesis presents is Thinger.io which is often described as an open source with multiple capabilities for users that don't want to pay for a premium service. However, users are able to pay if they are willing to, in order to get premium subscription benefits such as the following:


MAKER
Free
Signup Free
For students, makers, or hobbyist working on IoT.
- ✓ 2 Devices
- ✓ Single Developer
- ✓ Community Cloud
- ✓ Basic Features ⓘ
- ✓ Community Support

SMALL
€25/month
Get Started
For developers, startups, or IoT Lab Testing.
- ✓ Unlimited Devices
- ✓ Single-Tenant
- ✓ Private Cloud M1 ⓘ
- ✓ Extended Features ⓘ
- ✓ Community Support
- ✓ Up To 1 Plugin ⓘ
- ✓ MQTT Support

MEDIUM
€129/month
MOST POPULAR
Get Started
For business developing their own IoT projects.
- ✓ Unlimited Devices
- ✓ Multi-tenant (Up to 5)
- ✓ Private Cloud M2 ⓘ
- ✓ Business Features ⓘ
- ✓ Community Support
- ✓ Up to 3 Plugins ⓘ
- ✓ MQTT Support
- ✓ 1 Brand
- ✓ 1 Custom Domain
- ✓ Unlimited Guest Members
- ✓ Extended Support Available ⓘ

LARGE
€259/month
Get Started
For companies providing IoT consultancy services.
- ✓ Unlimited Devices
- ✓ Multi-tenant (Up to 15)
- ✓ Private Cloud M3 ⓘ
- ✓ Business Features ⓘ
- ✓ Community Support
- ✓ Up to 5 Plugins ⓘ
- ✓ MQTT Support
- ✓ Up to 5 Brands
- ✓ Up to 5 Domains
- ✓ Unlimited Guest Members
- ✓ Daily Backups
- ✓ Extended Support Available ⓘ

UNLIMITED
€519/month
Get Started
For companies that want it all, unlimited.
- ✓ Unlimited Devices
- ✓ Multi-tenant (Unlimited)
- ✓ Private Cloud M4 ⓘ
- ✓ Business Features ⓘ
- ✓ Community Support
- ✓ Unlimited Plugins ⓘ
- ✓ MQTT Support
- ✓ Unlimited Brands
- ✓ Unlimited Domains
- ✓ Unlimited Guest Members
- ✓ Daily Backups
- ✓ Priority Support Available ⓘ

Figure 14 – Thinger.io premium subscription plans as of 2022.

One of the premium benefits, can also make Thinger.io a suitable for companies that provide products related to the Internet of Things. Thinger.io Platform, according to their official website, started in early 2015, as a side-project by PhD. Alvaro Luis Bustamante while he was working as a researcher at the University Carlos III of Madrid, Spain. The project quickly became a much more general tool that could easily be extended in any direction. Eventually, in 2018, the project was turned into a loyalty founded company and release an enterprise version of Thinger.io platform technology to provide professional tools with simplicity, powerful technology, and support for Internet of Things projects Thinger.io, (2022a).

A great feature that Thinger.io can provide us with, is the dashboard system, which allows everyone to create nice data representation interfaces within less than an hour very easily. No coding is required, with drag and drop technology the configuration of the dashboard with different widgets can be configured effortless. There are 3 main types of widgets, 1) Real-time data representation, 2) Historical, data representation from buckets and 3) Control, device function or value changes with on/off buttons or sliders Thinger.io, (2020).

Furthermore, Thinger.io promises to deliver “Simple but Powerful” infrastructure, when in a few lines of code, the developer can connect a device and start retrieving data or control functionality with the web-based console in order to manage thousands of devices in a simple way, however, with the free subscription only 2 devices can be controlled at the same time. While our research for this thesis we found it to be accurate, programming both ESP32 Dev Module and ESP8266 so easily. We were able to connect to the Thinger.io platform in just a few minutes and send temperature values from the DHT11 sensor that we used in an experiment. Their platform is easy to use by people who are just getting started to IoT platforms and ESP module enthusiasts who want to take their experiments to the next level. Another great thing that Thinger.io promises to deliver, is Hardware agnostic, that allows any device from any manufacturer to be integrated easily with Thinger.io infrastructure. Thinger.io can connect in both my ESP8266 (ESPRESSIF8266), ESP32 Dev Module (EPRESSIF32) devices, but there are also multiple other choices, at least for the time being such as connecting with Arduino Ethernet, Arduino WIFI, Arduino GSM, TI Launchpad CC3200, SeedStudio LinkIT ONE, and GPRS Connection, MQTT Clients, HTTP Devices, Linux/Raspberry Pi. They also have a built-in example for Arduino Nano 33 IoT in their officially library Arduino package but unfortunately, we didn’t have the opportunity to connect it with the Arduino Nano 33 IoT model even though the code was provided by Thinger.io, at least for the time being (Thinger.io,2022b). Moreover, as mentioned before, the greatest thing about –Thinger.io, is that runs like an open-source, most of the platform modules, libraries, and APP source code which are free to be used and can be accessed very easily especially through popular IDEs like the Arduino IDE (Thinger.io, 2022c)

Lastly, the features of Thinger.io don’t stop there, multiple extended features exist such as: OTA Programming (Over the Air), which is a method that allows to remotely send new firmware to IoT devices over the internet, from anywhere with the Thinger.io cloud to deliver new binaries. This feature can allow us to keep the devices updated with new security patches and code improvements in a fast and scalable way, which also makes it an ideal solution to maintain IoT products from travelling locations (Thinger.io,2021a).

The Remote Console is another great feature that Thinger.io includes, which allows remote access on all our devices. It allows the creation of web terminals to interact with the device like in a serial interface, this feature can become useful for remote diagnosis, showing logs on devices, managing device configuration or any other functionality which can be extended to include new commands (Thinger.io,2021b). And as of July,2022, more features such as the “Remote Filesystem”, more improvements and roadmap developer updates, will continue to be released along with the necessary documentation.

When it comes to the architecture of the Thinger.io platform. It consists of a Backend IoT Server and web-based Frontend. A few of the main features that the Thinger.io platform has at it’s disposal according to their official documentation are(Thinger.io,2021d):

- Device Connection, where a user is able to connect their devices
- Device Data Storage, where a user can store the data using the Data Buckets
- Real-Time Display, and multiple widgets like donut charts, gauges, or custom representations

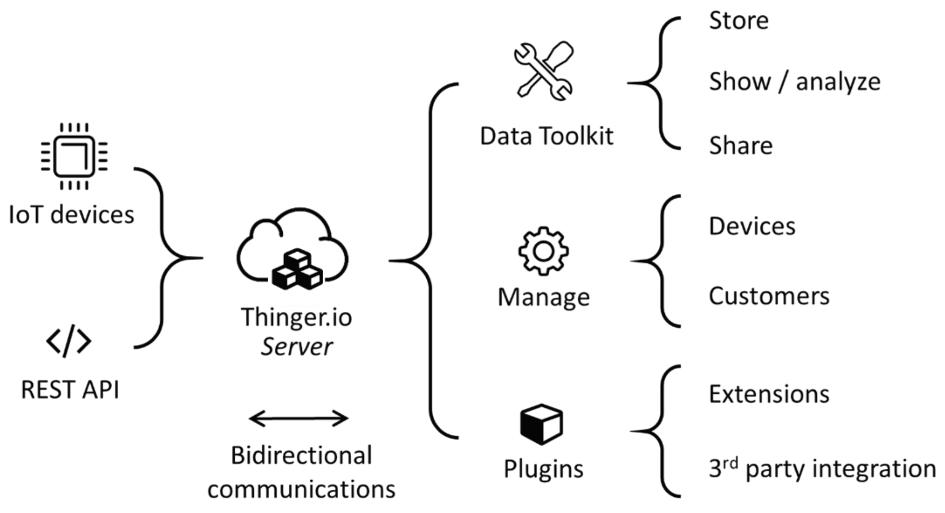
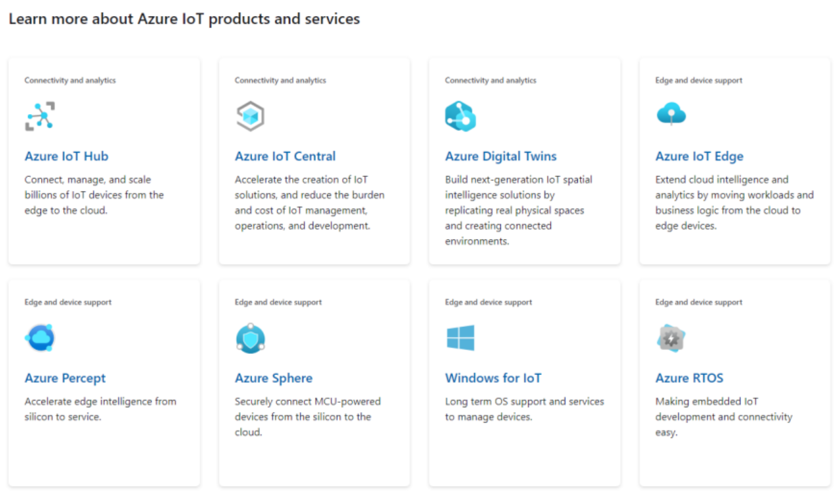


Figure 15 - The Thinger.io platform architecture

6.2 Microsoft Azure IoT


Learn more about Azure IoT products and services

Connectivity and analytics
Azure IoT Hub
Connect, manage, and scale billions of IoT devices from the edge to the cloud.

Connectivity and analytics
Azure IoT Central
Accelerate the creation of IoT solutions, and reduce the burden and cost of IoT management, operations, and development.

Connectivity and analytics
Azure Digital Twins
Build next-generation IoT spatial intelligence solutions by replicating real physical spaces and creating connected environments.

Edge and device support
Azure IoT Edge
Extend cloud intelligence and analytics by moving workloads and business logic from the cloud to edge devices.

Edge and device support
Azure Percept
Accelerate edge intelligence from silicon to service.

Edge and device support
Azure Sphere
Securely connect MCU-powered devices from the silicon to the cloud.

Edge and device support
Windows for IoT
Long term OS support and services to manage devices.

Edge and device support
Azure RTOS
Making embedded IoT development and connectivity easy.

Figure 16 – Azure IoT products and services by Microsoft.

Azure provides a platform to develop and deploy IoT based solutions. The Azure public cloud is available in over 50 regions around the world, featuring an extensive management and security framework along with the necessary tools to support all these implementations. Microsoft recommends deploying its Azure IoT hub cloud service to enable connection between the IoT edge devices and the Microsoft Azure cloud. IoT hubs are capable of ingesting billions of events per year. Microsoft’s Azure IoT Hub supports a variety of popular IoT protocols for queuing and transmission of data including HTTPS, AMQP, WebSockets, MQTT, and MQTT over WebSockets.

Other protocols can be handled through performing protocol conversion in the cloud through deployment of a customized Azure IoT protocol gateway or protocol conversion at the edge within the Azure IoT Edge. The IoT Hub also provides more useful tools in the architecture, where it is used for managing devices and device twins, file upload from devices, identity and authentication, device provisioning and cloud to device messaging. Furthermore, IoT Hub can support up to 100 devices running Microsoft’s IoT edge. Lastly, IoT Hub can support bidirectional communication to send policies, commands, and cloud-generated intelligence back to edge devices (Stackowiak,2019).

Azure Digital Twins according to the official Microsoft website, is an Internet of Things device that enables everyone to create a digital representation of real-world things, places, business processes and people(Stackowiak,2019). Usually, a digital twin provides a means to represent the location of a device in the physical world and Azure Digital Twins is no exception (Azure,2021a).

Microsoft's Azure IoT also offers multiple other products and services including, Azure Percept is a platform that is easy-to-use with enhanced security in order to create edge AI solutions (Azure,2022a). Azure IoT central which is a UX and API surface for connecting and managing devices mostly for business, it also strongly enforces platform as a service offering (PaaS), gathering each service underneath it for easy and secure configuration (Azure,2022b). Azure Sphere(Stackowiak,2019), is about creating, connecting, and maintaining secure IoT devices that can be deployed in edge devices with a class of crossover Microcontroller units (MCUs) (Azure,2022c).

However, a great disadvantage when it comes to Azure IoT is that it's a premium service. Even though as of 2022, Microsoft does offer a 12-month Azure free-trial access for their services, a credit card is required, and even then, if you want to continue your access after the trial is over a sales assistant will need to contact you as there are no clear pricing plans and charge is based upon the products that you want to use. For example, the cost of popular services like Virtual Machine, Storage Accounts, Azure SQL Database, App Service, Azure Cosmos DB, Azure Kubernetes Services (AKS), Azure Functions, Azure Cognitive Services and Microsoft Cost Management can reach, according to the official pricing calculator up to 152.62 US Dollars per month (Azure,2022d), which might make it an ideal service for businesses and corporations but not for individuals. Considering that Azure IoT has multiple services that consist of different architectures and services, Azure IoT Central's architecture will be the main focus of this thesis which will be used on the Chapter 8. A few of the key features that IoT Central has at its disposal can be seen as the Figure 17 suggests. According to their official documentation, IoT Central offers Device Management, Device Data Analysis, Storage and Analysis, Data Export, and many other useful features that a person might find useful(Azure,2022d).

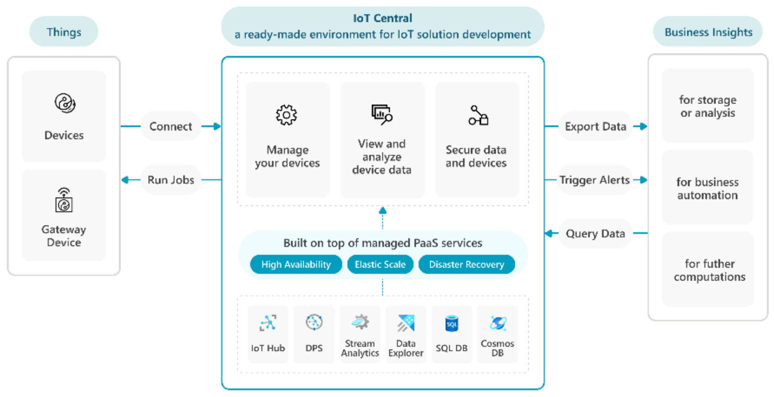


Figure 17 – The architecture of Azure IoT Central.

6.3 ThingsBoard

ThingsBoard

Figure 18 – ThingsBoard official logo.

ThingsBoard promises an open-source IoT platform for data collection, processing, visualization, and device management (Thingsboard,2022a), however that is not the case. As with Microsoft’s Azure IoT they do offer 1-month free trial but after the trial is over, a price plan must be selected (Thingsboard, 2022b). Pricing plans can vary from 10$ a month up to 749$ or a custom pricing plan for more specific perks and services(Thingsboard, 2022b).

ThingsBoard according to their official website, ThingsBoard inc. was founded in 2016 and develops IoT software products under ThingsBoard umbrella. ThingsBoard inc. evolved from a startup with IoT with 2 employees to an IoT enabler with hundreds of licenses and many community users who support them, located in Kyiv, Ukraine (Thingsboard,2022c). And, because of the war conflict between Russia and Ukraine they refuse sales to Russian companies(Thingsboard,2022d).

ThingsBoard is compatible with Linux, Windows, Mac OS and even Raspberry Pi, with a huge choice of support protocols to connect the devices of your choice like CoAP, HTTP, MQTT, LwM2M and for an enhanced security the user can either pick MQTT (over wss) or

HTTPS protocols for transport encryption. The data can later be stored into the ThingsBoard database that uses Cassandra (Thingsboard,2022g).

Moreover, ThingsBoard offers extensive guides, get-started guides, documentation, and YouTube videos officially made by the company, that explore and teach how to learn and use the ThingsBoard platform (Thingsboard,2022e). With ThingsBoard, a user can collect and visual data from devices and assets that can be provisioned, analyze incoming telemetry and trigger alarms with complex event processing, control devices using remote procedure calls (RPC), push device data to other systems and much more (Thingsboard,2022f).

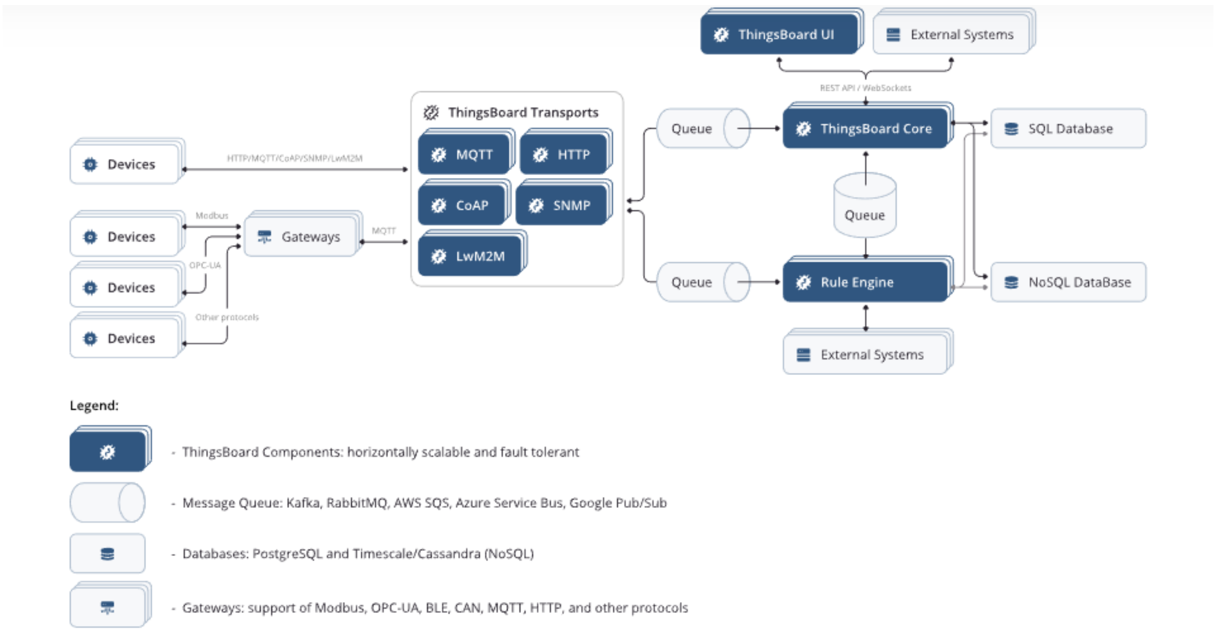

ThingsBoard UI
External Systems
REST API / WebSockets
ThingsBoard Transports
MQTT
HTTP
CoAP
SNMP
LwM2M
Queue
ThingsBoard Core
SQL Database
Queue
Queue
Rule Engine
NoSQL DataBase
External Systems
Devices
HTTP/MQTT/CoAP/SNMP/LwM2M
Devices
Modbus
Gateways
MQTT
OPC-UA
Devices
Other protocols
Devices
Legend:
- ThingsBoard Components: horizontally scalable and fault tolerant
- Message Queue: Kafka, RabbitMQ, AWS SQS, Azure Service Bus, Google Pub/Sub
- Databases: PostgreSQL and Timescale/Cassandra (NoSQL)
- Gateways: support of Modbus, OPC-UA, BLE, CAN, MQTT, HTTP, and other protocols

Figure 19 – The ThingsBoard architecture

A few noteworthy features that the ThingsBoard platform provides at it's disposal according to their official website, is scalability, fault-tolerant, robust and efficient and durable (Thingsboard,2022h).

6.4 AWS IoT

Figure 20 – Amazon Web Services IoT official icon.

Amazon Web Services (AWS), owned by the famous company Amazon, introduce their own platform AWS IoT. AWS IoT supplies cloud services and software to the support the integration of IoT devices into the AWS IoT-based solution, if connected to the AWS IoT, then AWS IoT can connect them to the cloud services that AWS provides (AWS,2022a).

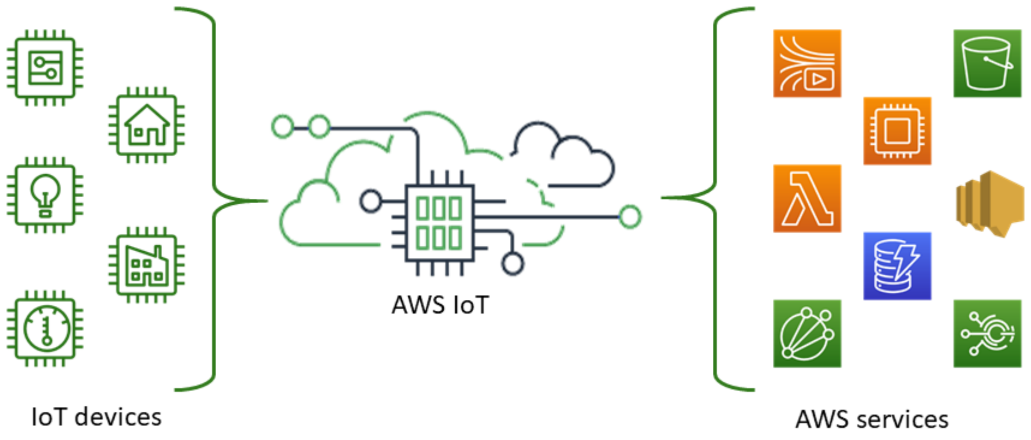


Figure 21 – AWS IoT connecting IoT devices to the AWS Services.

AWS IoT also allows connection between multiple communication protocols such as MQTT, HTTPS, MQTT over WebSockets and LoRaWan in secure devices with authentication and encryption (AWS,2022b).

Furthermore, AWS IoT provides a lot of solutions that can be found in Home Automation and industry and much more such as how to build predictive quality models for industrial operations (Németi et al, 2017), AWS IoT to support predictive maintenance in industrial operations, usage of AWS IoT in your connected home, AWS IoT to provide home security and monitoring, AWS for connected vehicle, AWS IoT Greengrass Machine Learning Inference and so on (AWS,2022c).

Moreover, AWS IoT has a lot of documentation, tutorials, courses, and videos that helps explain multiple subjects such as, AWS IoT Authentication and Authorization, Internet of Things Foundation Series, IoT lens, designing MQTT for AWS IoT etc. Along, with multiple troubleshooting guides(AWS,2022c).

Amazon offers a pay as you go strategy along with a flexibility of tools, programming languages, data management and infrastructure resources such as ActiveMq and Mosquitto servers to assist in managing and analyzing applications (AWS,2022d). Amazon Web Services provide a lot of tools at their disposal for example Amazon Cognito, Amazon Mobile Analytics and Mobile push to make the assistance and management assistance effortless (Nakhuva & Champaneria,2015).

The way Azure IoT works is that multiple devices, or as their official website puts it, billions of devices, can transmit messages using the MQTT protocol which can minimize the network bandwidth, and at the same time, the device’s eco footprint. Finally, the devices gain the ability to communicate with each other using the Amazon Web Services IoT Core (Microsoft Press,2022).

6.5 Google Cloud IoT

Figure 22 – Google Cloud IoT official icon.

Google is also one of the companies that have their own IoT platform for developing IoT solutions. IoT Core, combined with other services on Google Cloud can provide a solution for collecting, processing, and analyzing Internet of Things data in real-time to support improved operational efficiency. IoT Core supports standard MQTT and HTTP protocols, so it makes the usage of existing devices with minimum firmware changes.

A few of the key features that Google's IoT are Device Manage which allows individual devices to be configured easily and securely through the console or programmatically. Protocol Bridge, which provides connection endpoints for protocols, and support for MQTT and HTTP. End-To-End security, which is enabled using asymmetric key authentication over TLS 1.2. Enable-High Frequency-Low-Latency communication, which allows sending commands fast and frequent with one-time directives sent to devices (Google,2022a).

A few useful tools that Google's Cloud IoT has at its disposal, are Predictive Management where Google Cloud's Io platform allows automatic prediction when equipment needs maintenance and optimization in its performance. Real-Time asset tracking, where gathering of complex of complex analytics and machine learning on the collected data can be performed, Logistics and supply chain management, where inventory tracking cargo integrity monitoring and other useful business critical functions can be made, and lastly, smart cities and buildings where an entire new level of automation and intelligence can be made.

Google is a popular IoT platform because of the fast global network, Google's BigData tool and support of various services like Firebase, Connecting Arduino and Firebase, Cassandra on Google Cloud Platform and may more (Nakhuva & Champaneria,2015). Unfortunately, Google has announced that their official Google Cloud IoT platform will shut down in August 2023, this is most likely according to Josh Taubenheim, MachNation's head IoT analyst due to the lack of modernization of their platform and a failure of adhering to the current security standards (Alleven,2022).

6.5 IBM Watson

Figure 23 – IBM Watson IoT official icon.

In February 2011, IBM's Watson computer competed on Jeopardy! Against two of the show's champion. Watson is a computer running Deep QA software that was developed by IBM Research. Even though the main objective for Watson was to win at Jeopardy! The overall picture was the creation of a new generation of technology, which can search and find answers more efficiently than the standard technology.

David Ferrucci, who spent 15 years as an IBM researcher on natural language problems and finding answers amid unstructured information, mentioned that the goal was not the modeling of the human brain, but to build a computer that can be more effective in understanding and interacting in natural language, not necessarily the same way that humans do. Watson runs on a cluster of Power 750^{tm} computers, 90 servers for a total of 2880 processor cores running DeepQA software and storage. Over the years Watson has been informed from various sources and information, so when a question occurs, more than 100 algorithms analyze the question in different ways and find many different similar and possible answers simultaneously (IBM,n.d.a).

Maximo Asset Monitor is an end-to-end, fully managed cloud service that is built on Watson Internet of Things platform, IBM services that make up Maximo Asset Monitor support the functions of connecting, storing, analyzing, monitoring, and managing with further monitoring support. There are multiple services that Maximo Asset Monitor supports such as, platform component, which includes a platform for IoT device message broker and more, analytics components which provide analytics features to interact with IoT and non-IoT data with performance indicators, Databases for PostgreSQL, in order to store the data and more such as, IBM cloud object storage and event streams (IBM,n.d.b). Considering how many services the IBM cloud has to offer, the price is based on the services that an individual user requires and a consultation with an IBM sales assistant may be required for a full introduction. IBM Watson IoT platform also supports MQTT and HTTP, device or gateway support, REST and real-time APIs and application and analytics (IBM,n.d.c).

Furthermore, according to their official website, IBM Watson IoT provides a dashboard with many useful tools that are easy to use to manage existing devices, control and monitor them. A few of the capabilities include remote device management, where a user can update, remove, reboot, and receive device diagnostics remotely, secure connections using MQTT with TLS and a massive data storage with a long database.

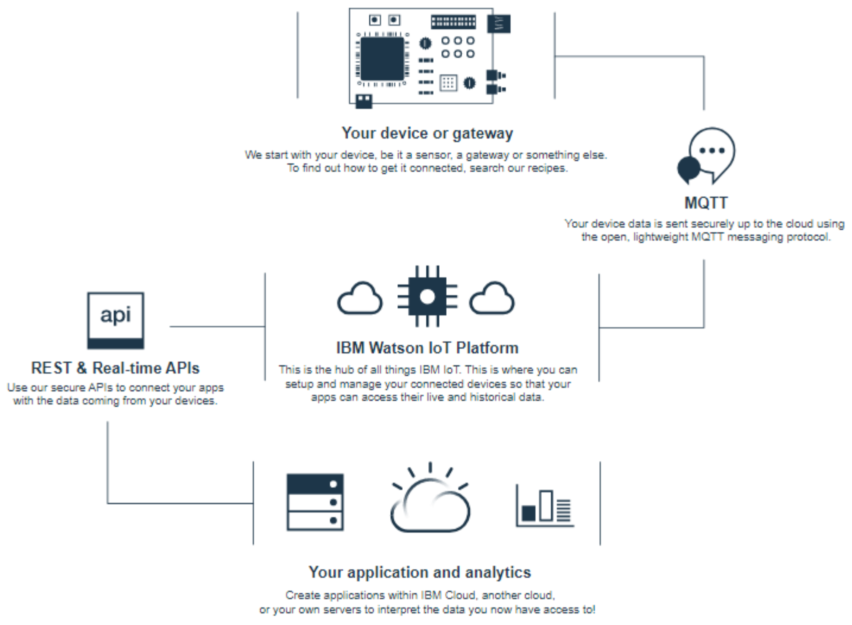

Your device or gateway
We start with your device, be it a sensor, a gateway or something else. To find out how to get it connected, search our recipes.
MQTT
Your device data is sent securely up to the cloud using the open, lightweight MQTT messaging protocol.
api
REST & Real-time APIs
Use our secure APIs to connect your apps with the data coming from your devices.
IBM Watson IoT Platform
This is the hub of all things IBM IoT. This is where you can setup and manage your connected devices so that your apps can access their live and historical data.
Your application and analytics
Create applications within IBM Cloud, another cloud, or your own servers to interpret the data you now have access to!

Figure 24 – How the devices operate with the IBM Watson IoT.

Another interesting aspect that IBM Watson IoT provides, in my own opinion, is that their website provides multiple programs, an enhanced documentation with multiple articles that can teach and assistant anyone with a very basic level of IoT experience, and last but certainly not least, a forum where anyone can post a support question and receive support, making it an ideal choice for people that are willing to learn.

6.7 Oracle Internet of Things Cloud Service

Oracle Internet of Things, made by the Oracle Company, the company that owns the Java programming language. Oracle Internet of Things Cloud service makes IoT simple, more productive, and efficient. Oracle Internet of Things platform provides real-time analysis tools and you-build-in integrations that allow the automatic synchronization of data streams. A few of the supported browsers for the Oracle Internet of Things Cloud service are Google Chrome, Microsoft Edge, Mozilla Firefox, and Apple Safari (Oracle Help Center, 2022a).

Oracle supports 3 device types, the Gateway Device, the Directly Connected Device, and the Device Connected via gateway. More specifically, in the Gateway Devices, a device communicates with Oracle IoT Cloud Service instead of the device that is incapable of direct communication with the Oracle IoT Cloud Service, the Directly Connected Devices, is about the devices that capable of connecting directly with the Oracle IoT Cloud Service via a running application that is based on Oracle IoT Service Client Software Libraries. The only supported direct connection protocol is HTTPS over TCP/IP. Lastly, Devices Connected via Gateway, as the name suggests, are devices that are connected to the Oracle IoT Cloud Service via a Gateway and do not need any Oracle IoT Cloud Software to be installed on them.

A few of the benefits that the Oracle Internet of Things Cloud Service has to offer are according to their official website, Intelligent applications that can provide more visibility insights and efficiencies by capturing data from connected devices. Smart manufacturing, which can increase the revenue by using efficient IoT applications, increase the supply chain launch and launch new business models. Easier maintenance of machines, vehicles, and other assets thanks to the IoT sensor data with real-time visibility, IoT for transportation and logistics that can reduce the cost of the connecting and monitoring vehicle routes and workplace safety that can identify causes of incidents with real-time visibility in the work environments.

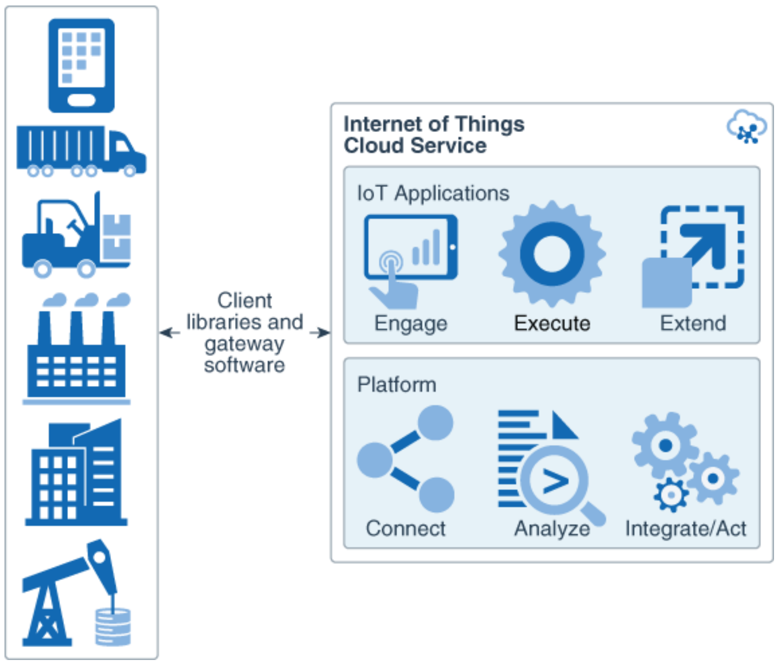


Figure 25 – The Oracle Internet of Things Cloud Service architecture.

According to the official Oracle documentation, the Oracle Internet of Things Cloud, a few of the main key features that their platform has to offer is device connectivity, data analysis, integration, production monitoring, fleet management, worker safety environment among many other features that a company that provides services and solutions may find useful (Oracle Help Center,2020).

6.8 OpenRemote

openremote
Creating Meaningful Connections

Figure 26 – OpenRemote official logo.

OpenRemote originated in 2015 by the founder of JBoss, Marc Fleury, with an ambition according to their official website, to overcome challenges of integrating a variety of different protocols and data sources into a straightforward data management solution (Wikipedia,2022). Their open solution can be applied to multiple asset management classes such as energy systems, vehicle fleets, crowd management, helping buildings and many more sectors to become smarter (OpenRemote,2022b).

Just like Thinger.io, OpenRemote is an open-source IoT platform. One of the greatest features that OpenRemote can offer in my opinion, is its community which is full of enthusiastic IoT developers and constantly evolves (OpenRemote, 2015). They also have a massive portfolio that is also available on their official GitHub website, their own wiki, even their own forum page where developers can post questions or bugs and receive support for free! (OpenRemote,2022a).

Another great feature that OpenRemote can offer is the fact that it runs on Docker Containers, which essentially means that the platform can run on any hardware and operating system like Windows, Mac OS, Linux and so on either locally or remotely.

OpenRemote can utilize several protocols such as HTTP, UDP, MQTT, TCP-IP, WebSocket, KNX, Velbus and Z-Wave along with many more, for different project requirements which can make this platform ideal for many users, companies and IoT developers alike. Their website also offers as a getting-familiar example, a project which can collect live weather data with an HTTP API example which I personally found quite useful and with a moderate difficulty to be understood (OpenRemote,2022a).

Furthermore, OpenRemote can support multiple edge gateways and devices, including but not limited to (Okta,2022), auto-provisioning devices such as Espressif 32 (ESP32), Nordic nRF9160, TI ESP8266, multiple cryptographic authentication techniques such as Hash-Based Message Authentication Codes (HMAC) and X.509 (OpenRemote,2022c.).

However, the features that OpenRemote has to offer don’t stop there, they also offer a configurable dashboard, console for iOS10 or higher, console for android 5 or higher, Web Console, multiple messaging & manager rules that include many appealing features. Some of the features are JSON and Flow Rules Object Model, messaging service for push, notification, and e-mail. There are also many options for assets and security, that are consistently being added on helping the website to grow and expand, all of them as mentioned before, for free (OpenRemote,2022c.).

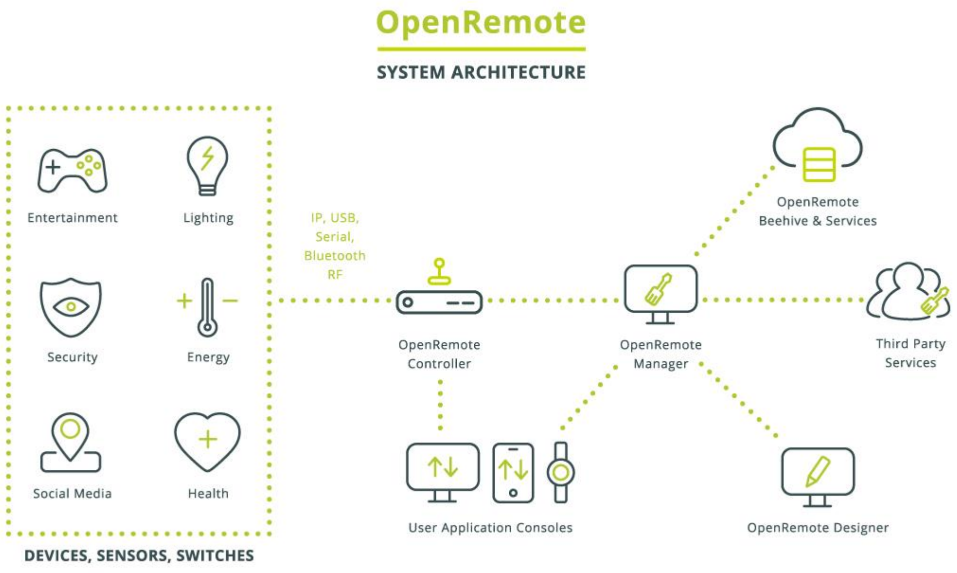


Figure 27 – The OpenRemote Platform Architecture.

7. Creating IoT Smart Applications

In this Chapter, we will introduce how a smart IoT application can be made from scratch, with the assistance of IoT platforms and microcontrollers from different manufacturers with different specifications and capabilities. The current chapter aims to assist experienced developers, Internet of Things enthusiasts, programmers, and University Students alike, however a basic knowledge regarding microcontrollers is strongly recommended. This chapter will also include helpful getting-started instructions including the creation projects from scratch, installing the necessary prerequisites like the programming IDEs into a workstation and setting up the devices to transmit data into the IoT Platform. Once the data is processed and transmitted to the IoT Platform (with different methods depending on each platform) all the data will be available to be monitored regardless of where a user or a microcontroller may be in as long as the microcontroller has a reliable power source and maintains a connection to the internet.

Even though many of the mentioned IoT platforms may change their device support, their privacy policy along with many other features, after the writing of the current Thesis is completed the core idea will remain the same; how a microcontroller or a device can be used to successfully establish a connection and transmit data into the IoT Platforms. The transmission of data includes using different protocols, devices and specifications, methods and data that varies with different projects. However, to successfully establish a connection between our microcontroller and the IoT Platform, the microcontroller must meet certain criteria, including:

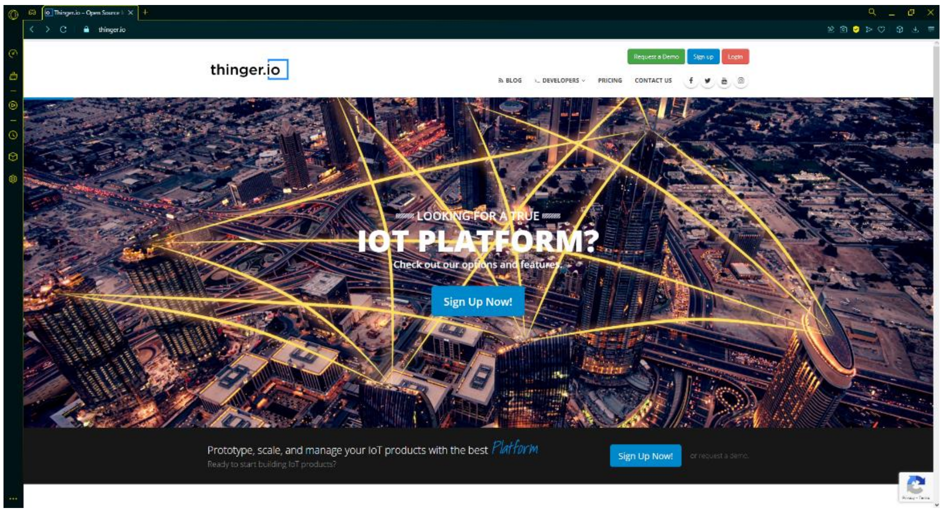
- ✓ **Wi-Fi connectivity**. A microcontroller or a device, without a feature to access a Wi-Fi network, is incapable of accessing the Internet, therefore unable to send data through the web, unless an external method is used, or a different microcontroller model. For example, my Arduino Uno Rev 3, is unable to access the Internet unless an Arduino Internet Shield or a different product is used.
- ✓ **IoT Platform support**. Many, of the above-mentioned platforms of Chapter 6, have different devices and protocols that they support. Not every device or microcontroller is able to access every IoT Platform and send messages with the same method and protocol, unless the IoT platform supports them or plans to support them on the future.

- ✓ **Consistent Connection**. As mentioned before, a device or a microcontroller will be unable to send data without a network connectivity and code which enables the Wi-Fi connection, therefore having a reliable network connection is a strong necessity.
- ✓ **Reliable Power Source**. Another key factor that should be taken into consideration, is the power source that the device or the microcontroller must consume in order to function as intended and sustain itself properly. If the power source is inefficient or faulty, the device won't be functioning properly, therefore will stop the data transmission into the IoT Platform.
- ✓ **Operating System & Web Browser Support**. Not all the IoT Platforms support all the current or older Operating Systems (OS), the developer must ensure that the IoT platform supports their operating system to develop their applications properly, the same principal equally applies to web browsers and programming IDEs.
- ✓ **Marketing Advantage and Region-Restricted Features**. Some of the IoT Platforms have ensured that they only work with specific partners and product companies in order to have a market advantage over their competition. Many of the mentioned IoT Platforms, have a restriction or a banning policy against many countries, unless they meet the company's policy.

The comparison on the following pages and Chapter 8, will be made strictly into different factors that should all be taken into an equal and strong consideration. All the codes, that were used to establish a connection and communication between the microcontrollers and the IoT Platforms will be available on the Appendixes chapters of the Thesis. The codes will also be available into my personal GitHub portfolio & website at github.com/AlexandrosPanag for further documentation and debugging purposes.

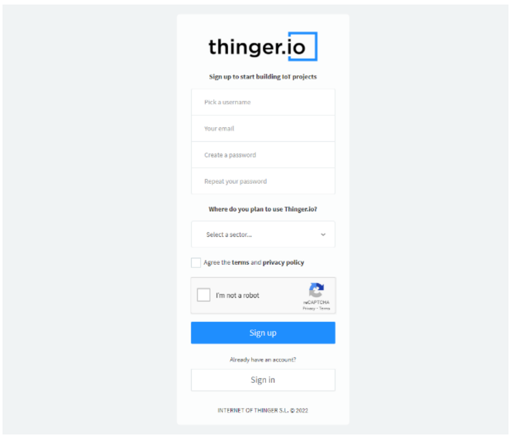
7.1 Thinger.io Initialization

The first Internet of Things platform that I want to present is Thinger.io. Thinger.io by the time or this writing, is free but also includes premium options for more features, as mentioned before. Firstly, we will be required to access their official website (Thinger.io) through a web browser and click the small "sign-up" box on the upper-right corner.



thinger.io
Request a Demo
Sign up
Login
BLOG
DEVELOPERS
PRICING
CONTACT US
LOOKING FOR A TRUE
IOT PLATFORM?
Check out our options and features
Sign Up Now!
Prototype, scale, and manage your IoT products with the best Platform
Ready to start building IoT products?
Sign Up Now!
or request a demo

Figure 28 – The official Thinger.io webpage.



After the user, pressed the "sign-up" box he will be prompted to the account creation page where he will need to create a username, enter an email address, a password, and a reason as to where do they plan to use Thinger.io.

Figure 29 - Thinger.io sign up page.

After their account is created and verified, they can log in anytime they want. Next, a user will need to navigate to their personal console which can be seen on the Figure 25. There the console can be seen which contains many settings where a user who is experienced with IoT Platforms may find very useful. A few of the settings include possible dashboards, data buckets and many other useful features that the Thinger.io platform provides as tools at its disposal which will all be explained on Chapter 8, for now the main focus of the current chapter will be establishing a proper connection and sending data to the IoT Platforms.

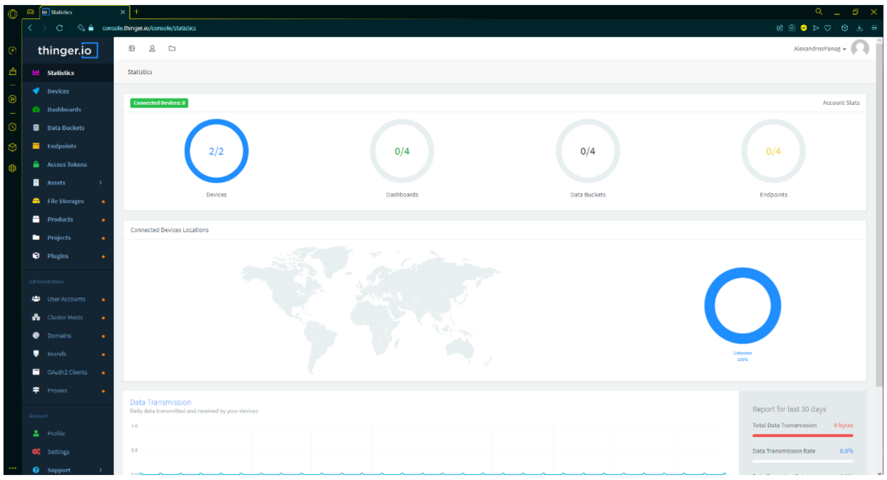
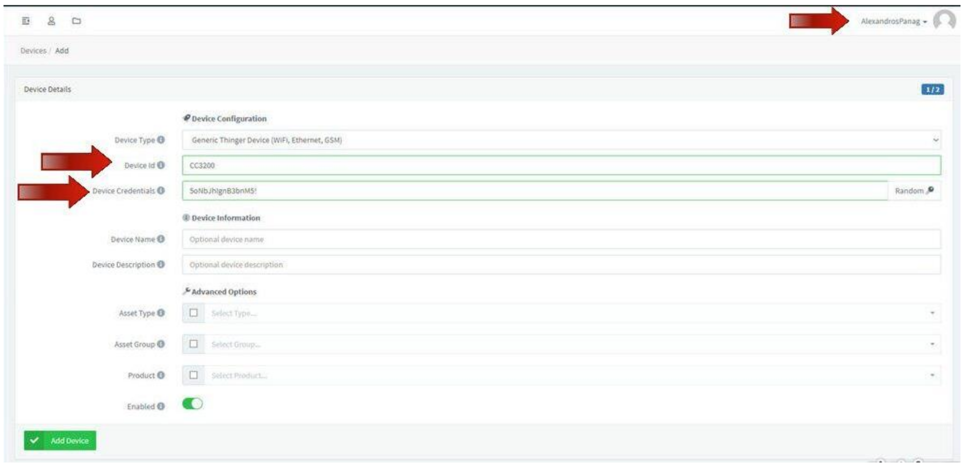


Figure 30 – The Thinger.io console.

However, to fully utilize the Thinger.io platform, the user will need to connect a device (or in this case, a microcontroller) into the Thinger.io platform, therefore, as mentioned before, the devices must have an internet access and be supported by the Thinger.io website.

The way that the Thinger.io receives data will be made with REST API from whatever source they are in, in this scenario the ESP32 and CC3200 microcontrollers, then they are able to generate a connection with the Thinger.io Server and send an HTTP request. The Integration provides also provides bidirectional communication between Thinger.io and the data source by making use of HTTP request and response data that consist of basic HTTP POST messages with JSON codified data (Thinger.io, 2021c). The process of how a connection can be made depending on the device, will be explained later in the chapter along all the necessary steps that are required for a connection establishment between the user's devices and the Thinger.io platform.

In order to add a device into your Thinger.io profile, you must navigate to the "Devices" option on the left corner and then click the "Add Device" green button. The window should appear something like the Figure 26 suggests. There, a user will need to enter the Device Id and enter the device credentials, everything else is optional. In this scenario, the rest of the options will be left default without applying anything to avoid any issues. Before continuing, a user should save the device id, the device credentials, and the entered username like the arrows suggest, as they will all later be required to connect the microcontroller into the IoT Platform.


AlexandrosPanag

Devices / Add

Device Details 1/2

Device Configuration

Device Type	Generic Thinger Device (WiFi, Ethernet, GSM)
Device Id	CC3200
Device Credentials	SoNbJhIgnB3bnM5!

Device Information

Device Name	Optional device name
Device Description	Optional device description

Advanced Options

Asset Type	Select Type...
Asset Group	Select Group...
Product	Select Product...
Enabled	

Figure 31 – Adding a device into your Thinger.io account.

Next, the user will be prompted to the device’s console, which as of the current time of this writing looks something like the Figure 27.

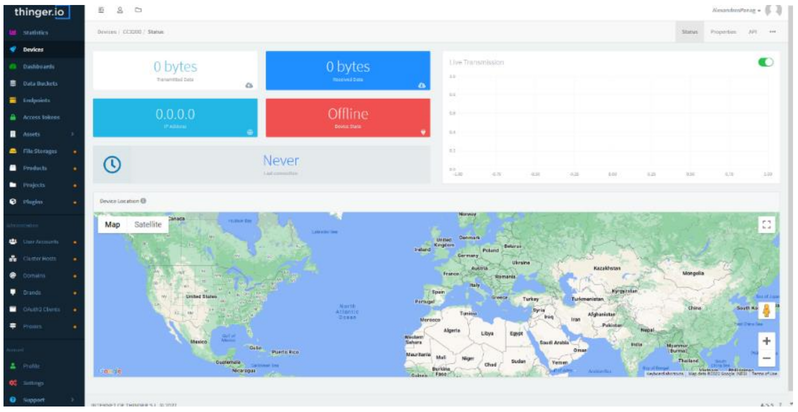


Figure 32 – The Thinger.io device console.

The same process must be repeated to create a profile for the Espressif32 (ESP32) who will be used later in the project (by applying a different Device ID and Device Credentials of course). Once the second’s device credentials are saved, the following projects should progress as intended without any issues.

7.1.1 CC3200 Launch XL Initialization with Thinger.io

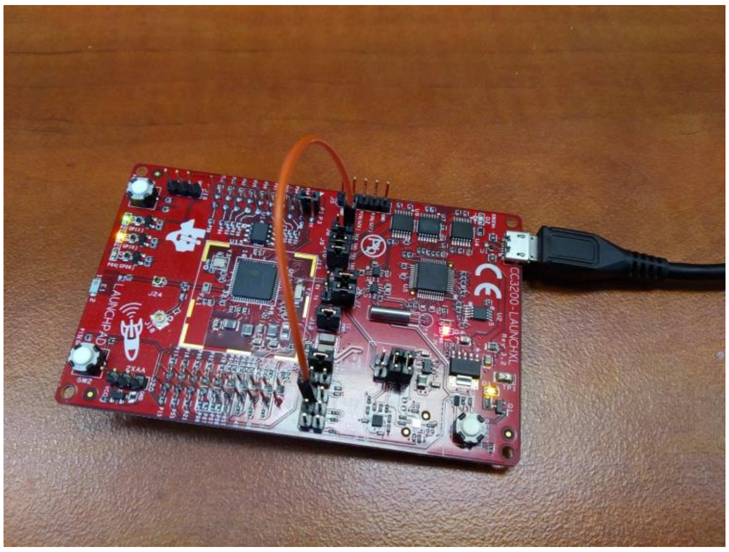


Figure 33 – CC3200 Launch XL.

CC3200 is a launchpad that was developed by Texas Instruments. According to their official website, CC3200 is the industry's first programmable MCU with built-in Wi-Fi connectivity. The launchpad includes driver and software support with 40+ applications for Wi-Fi protocols, Internet applications and MCU Peripheral examples, the board's initial price as of 2022, costs 55$ (USD) (TI.com,2022a).

A few of the key features include, two built-in buttons, three built-in LEDs for user interaction, micro-USB connector for power and debug connections, built-in chip antenna with Wi-Fi for conducted testing, built-in accelerometer and temperature sensor and a 40-pin LaunchPad standard that can effortlessly utilize the BoosterPack ecosystem provided by Texas Instruments, along with many other useful features such as low power modes which can make this board, an ideal choice for IoT applications(Ti.com,2022b).

The Autonomous Systems Laboratory of University of Ioannina, Department of Informatics & Telecommunications, has kindly provided me their laboratory equipment CC3200 Launch XL, which assisted towards the successful completion of my thesis.

In the application that follows, I'll present how useful data can be sent to the Thinger.io website. A few prerequisites before the project begins, include:

- **Energia IDE**, which can be downloaded here:
- **The CC3200's drivers for the OS Installation,** which can be downloaded from the following link, (requires a Texas Instruments approved and registered account): https://dr-download.ti.com/secure/software-development/software-development-

kit-sdk/MD-IM4pRiWjww/1.5.0/CC3200SDK-1.5.0-windows-installer.exe?__gda__=1661856403_6ea6c549ba48be9e6092f408f92ba6e7

- **A jumper wire female-to-female,** which should be installed to the top of J8 and the bottom of SOP2 Jumper pins, for more information view the following article, in my case I've encountered trouble with the UniFlash instructions, however, the Jumper wire can still work as intended in order to compile programs: https://energia.nu/guide/install/cc3200-guide/
- **The Thinger.io Library version 2.25.2**, which can be download from the link: https://www.arduinolibraries.info/libraries/thinger-io (any recent version won't be properly recognized by the Energia IDE unless the developers decide to update it)
- Lastly, you will need to navigate through the options in the Energia IDE. Go to **Tools>Board:>Board Manager** and install the CC3200 LAUNCH XL Board (and then set it on the settings).

Once everything is properly installed on a personal computer without any errors, the only thing remaining is connecting the CC3200 LAUNCH XL via USB into the computer and identifying which COM the board is connected to

In this project, the CC3200 LAUNCH XL Board will be used to connect towards the Thinger.io website and send data through the Internet into the device's console what state the built-in LEDs are. If the one of the lights is turned on the specific LED will send a message of 1, if not, then the specific LED will send a message of 0.

The way I've decided to implement the project idea is by generating a random number in the range from 10 to 99, for each LED. If the number can't be modulated by 2 (if LED%2==1), then the LED will light if the modulation is perfect (if LED%2==0), then the LED will not light. Furthermore, if the user decides to press one of the buttons, the program will generate new numbers. The user will also be able to view in whatever state the LEDs are in. The code will continue to run forever properly as long as the user maintains a connection to the internet and the microcontroller has enough energy to maintain its functions -- wherever the microcontroller may be located.

Once the code compiles successfully, the result can be viewed on the picture bellow. The code that was used to compile the project can be viewed on Appendix A. As a side note, if we as users need or want to store the data somewhere instead of losing all the valuable data we have gathered so far, Thinger.io also provides Data Buckets. In Data Buckets, all the data that we have so far acquired can be saved, an example will be made explaining how data storage is made on the Chapter 8 along with Data Visualization examples through the Thinger.io dashboard.

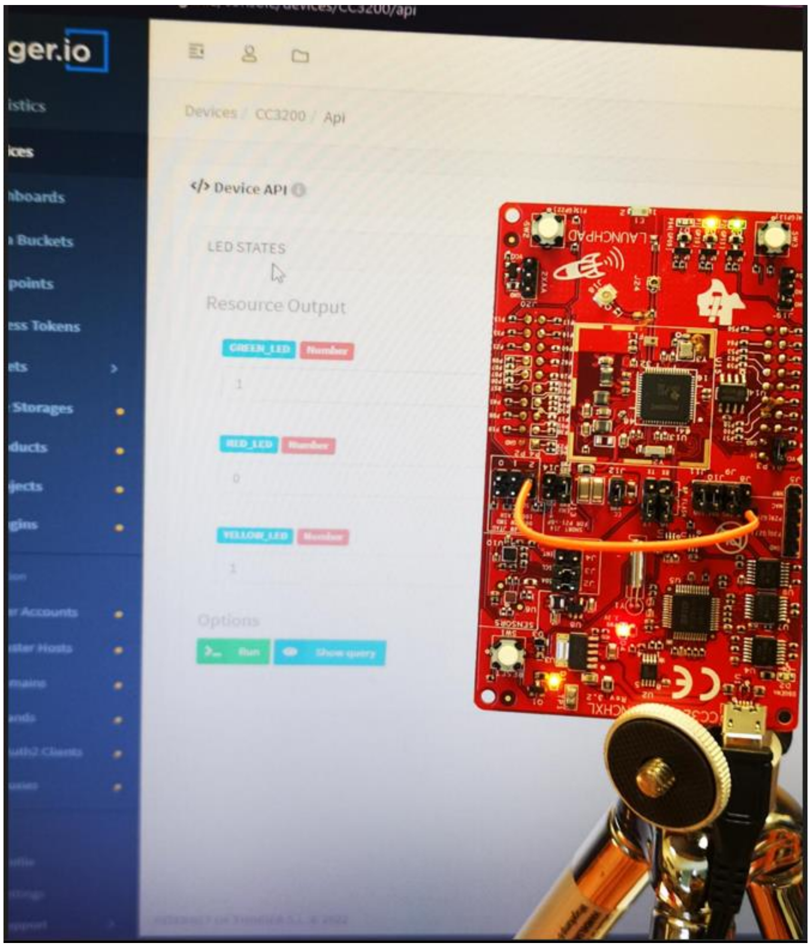


Figure 34 – CC3200 Sending the LED States into the Thinger.io platform.

7.1.2 Espressif 32 (ESP32) Initialization with Thinger.io

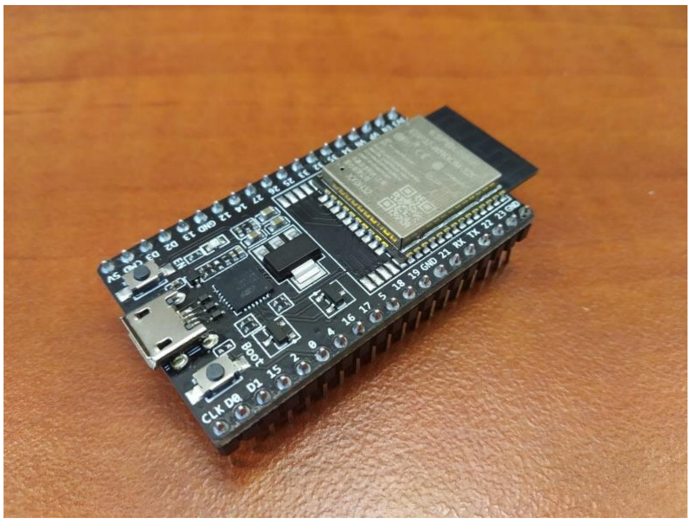


Figure 35 – ESP32 Dev Module.

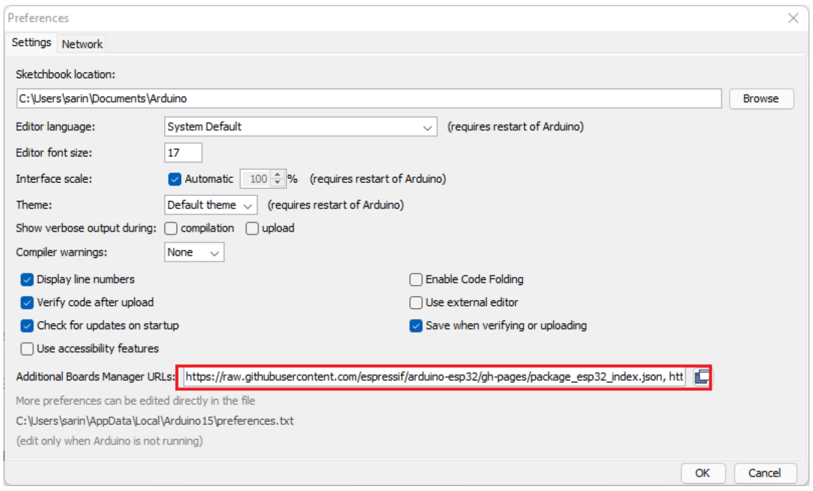
In this project, our own ESP32 – Dev Module was used. ESP32 is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs. ESP32 has a symmetric mapping which translates to, that they use the same addresses to access the same memory. Multiple peripherals in the board have direct memory access (DMA) (ESPRESSIF,2022a). ESP32, along with his predecessor ESP8266, rely on silicon from Espressif, a fabless silicon vendor that is based in China and was founded in 2008 (Cording & Cording,2022).

The cost of the ESP32 Dev Module is around to 10 US Dollars as of the time of this writing (Amazon.com,2019). ESP32 also includes integrated Wi-Fi and Bluetooth functionalities with multiple models that are based on this microcontroller with antennas, pinouts, and memory extensions that vary. ESP32 is designed to be one of the most popular choices when it comes to developing IoT applications, considering that it has a large diversity of variants that can utilize the capacities together with other peripherals (Thinger.io,2022d).

After creating a secondary profile through the Thinger.io website for the ESP32 board, there are a few project prerequisites that must be made too.

- First, the Arduino IDE is required which can be downloaded from here: https://www.arduino.cc/en/software
- After the IDE is installed, the user must navigate into File> Preferences> Additional Manager URLs and enter the link "https://raw.githubusercontent.com/espressif/arduino-esp32/gh-

pages/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json", as the picture bellow suggests.


Preferences

Settings | Network

Sketchbook location: C:\Users\sarin\Documents\Arduino
Editor language: System Default (requires restart of Arduino)
Editor font size: 17
Interface scale: [✓] Automatic 100 % (requires restart of Arduino)
Theme: Default theme (requires restart of Arduino)
Show verbose output during: [ ] compilation [ ] upload
Compiler warnings: None

[✓] Display line numbers
[✓] Verify code after upload
[✓] Check for updates on startup
[ ] Use accessibility features
[ ] Enable Code Folding
[ ] Use external editor
[✓] Save when verifying or uploading

Additional Boards Manager URLs: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json, htt

More preferences can be edited directly in the file
C:\Users\sarin\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Figure 36- Adding Additional Board Manager URLs

- Next, the user must go to the board manager and install "**ESP32 by Espressif Systems"** (and the esp8266, as it will be used on the Azure IoT Project).

After all the steps have been completed successfully without any errors, the application can begin. This step is necessary since both ESP32 & ESP8266 microcontrollers will be required to properly execute **all** the projects on the Chapters which will soon follow.

In this application example, ESP32 will need to send just like the CC3200 Launch XL, 3 LED States into the Thinger.io Platform. However, since ESP32 doesn't have 3 built-in LEDs and a built-in button (among many other features like CC3200 Launch XL has at its disposal), the LEDs must be connected externally with the help of a breadboard.

In the application that follows, I'll present how data can be sent to the Thinger.io website using the ESP32 microcontroller. A few prerequisites before the project begins, include:

- 3 LEDs
- 10 Male to Male Jumper Cables
- 3x Resistors of 100 Ω
- 1 Resistor of 1kΩ
- 1 Button
- A Breadboard

Once the project is uploaded successfully, from the Arduino IDE, the end result can be viewed from the Thinger.io Platform in the picture bellow:

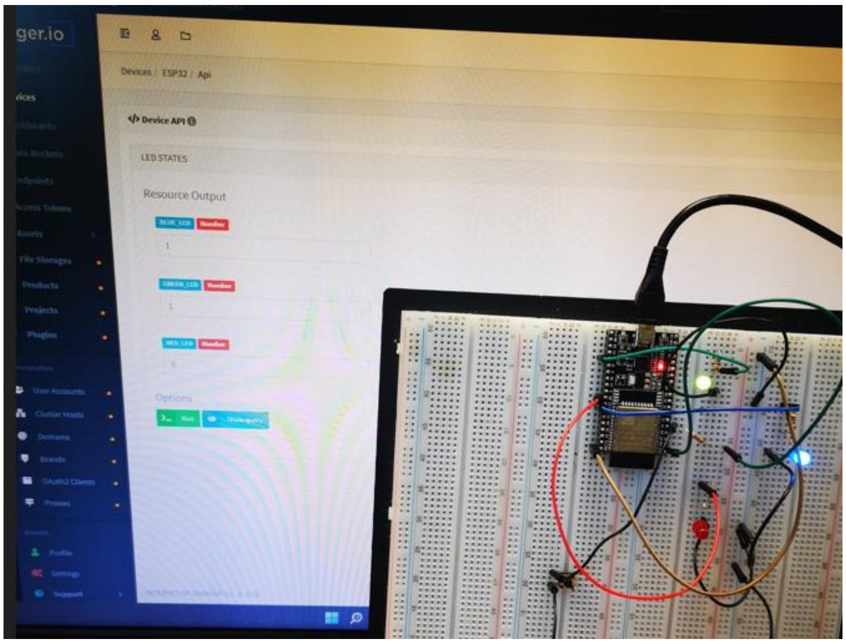


Figure 37 – ESP32 Sending the LED States into the Thinger.io platform.

7.2 Microsoft Azure IoT

As mentioned before on Chapter 6, Microsoft Azure IoT is a premium IoT platform with a pay as you go plan with many options, solutions, and programs to choose from. However, Microsoft does offer a 1-year free trial for academics and university students alike by using a certified academic account with the university's registered credentials as long as it is available in their university supported list.

Furthermore, Microsoft as a company strongly supports the use of open-source solutions and have their own GitHub page dedicated to Azure IoT solutions, which contains an extended platform documentation with many contributors that constantly expand, which can be found very useful for many Azure IoT enthusiasts and developers which can be found at the following URL: https://github.com/Azure.

Firstly, before anything can begin, a user must gain access through the Azure IoT main portal (which can be found at this URL: https://portal.azure.com/#home), by registering (or logging in with) a Microsoft account and choosing their premium support plan. Once the user has completed all the necessary steps of creating an account with their plan, he is able to gain access to the Azure IoT Central Platform by simply logging in with their registered credentials.

The way the data can be transmitted gives the user multiple options to choose from, including different protocols and methods that can be implemented. The way that both of the projects will transmit data through the Arduino IDE into the Azure IoT Services is done with the use of telemetry, MQTT protocol along with the help of JSON messages, which all function with the use of Wi-Fi.

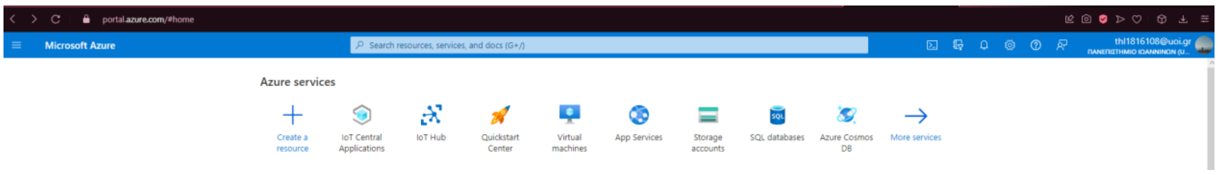


Figure 38 – Accessing the Microsoft Azure IoT Portal.

7.2.1 Accessing the Azure IoT Services with IoT Hub

The first method that a user can transmit data to the Azure IoT platform is by using the Azure IoT Hub. Once an account with a license is activated a user must navigate through the official Azure IoT Hub website (which can be found at https://portal.azure.com/#create/hub) and type "IoT Hub" into the search bar like the Figures 34 & 35 bellow suggests

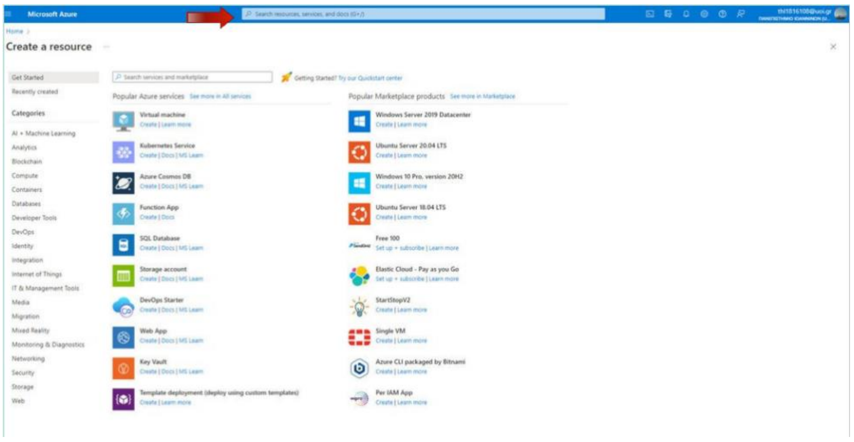


Figure 39 – Navigating through the Azure Portal.

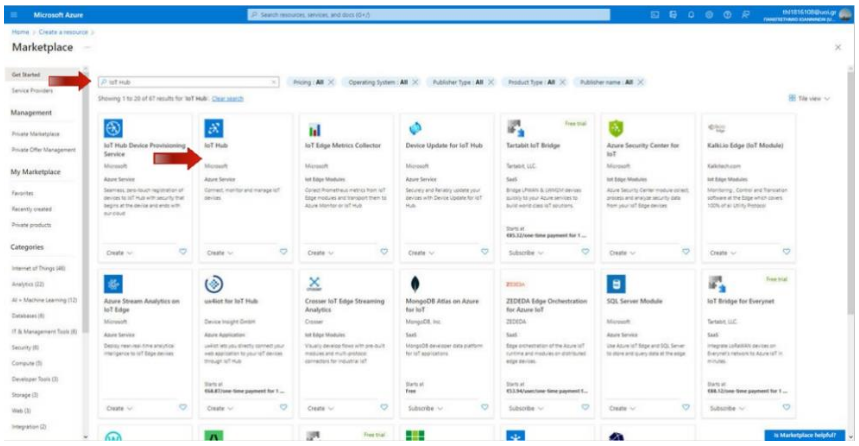
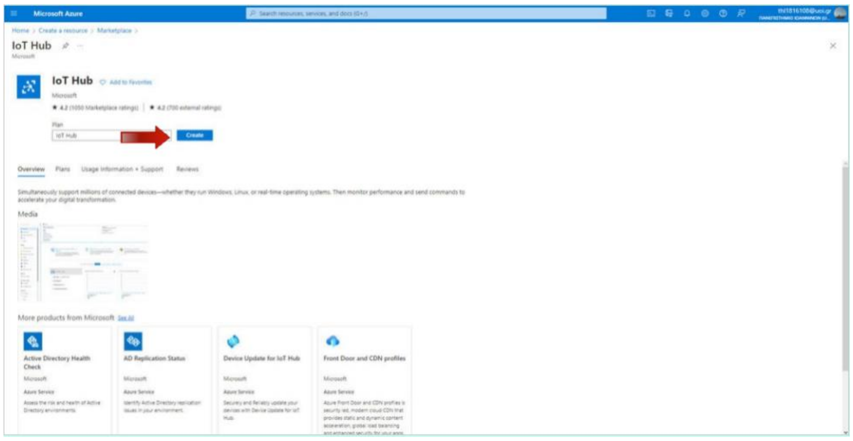

Microsoft Azure
Marketplace
Get Started
Service Providers
Management
My Marketplace
Categories
IoT Hub Device Provisioning Service
IoT Hub
IoT Edge Metrics Collector
Device Update for IoT Hub
Tartabit IoT Bridge
Azure Security Center for IoT
Kalki.io Edge (IoT Module)
Azure Stream Analytics on IoT Edge
ux4iot for IoT Hub
Crosser IoT Edge Streaming Analytics
MongoDB Atlas on Azure for IoT
ZEDEDA Edge Orchestration for Azure IoT
SQL Server Module
IoT Bridge for Everyinet
Create
Subscribe
Is Marketplace helpful?

Figure 40 – Searching for the IoT Hub through the Azure Portal.

There, the user will be required to click the IoT Hub and click on the "Create" blue button to create an IoT Hub like the Figure 41 suggests:

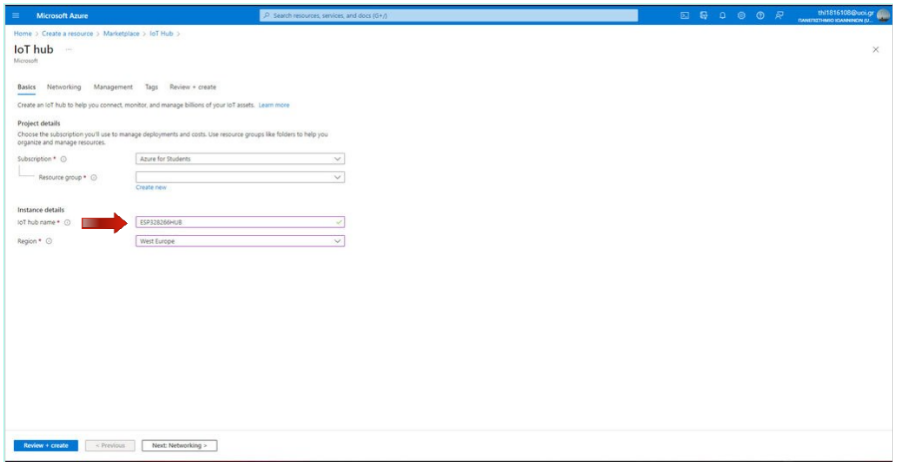

Microsoft Azure
Home > Create a resource > Marketplace >
IoT Hub
Microsoft
IoT Hub
Add to Favorites
Microsoft
★ 4.2 (1050 Marketplace ratings) | ★ 4.2 (700 external ratings)
Plan
IoT Hub
Create
Overview
Plans
Usage Information + Support
Reviews
Media
More products from Microsoft
See All
Active Directory Health Check
AD Replication Status
Device Update for IoT Hub
Front Door and CDN profiles

Simultaneously support millions of connected devices—whether they run Windows, Linux, or real-time operating systems. Then monitor performance and send commands to accelerate your digital transformation.

Figure 41 – Creating an IoT Hub.

After a user clicks on the "Create" button, he will be prompted into the following page which shows many options that a user can choose from to properly set-up the IoT Hub. For this scenario the hub was named "ESP328266HUB" which corresponds to both Espressif boards (ESP32 & ESP8266) that a user may use in order to create an application for the ESPRESSIF Boards.

As a sidenote, which might cause to a few users a minor inconvenience, is the fact that Microsoft's IoT Hub list of locations is very limited, hence why West Europe was chosen as a location even though Greece isn't located in the West Europe.



Create an IoT hub to help you connect, monitor, and manage billions of your IoT assets.

Project details

Choose the subscription you'll use to manage deployments and costs. Use resource groups like folders to help you organize and manage resources.

Subscription * : Azure for Students

Resource group * :

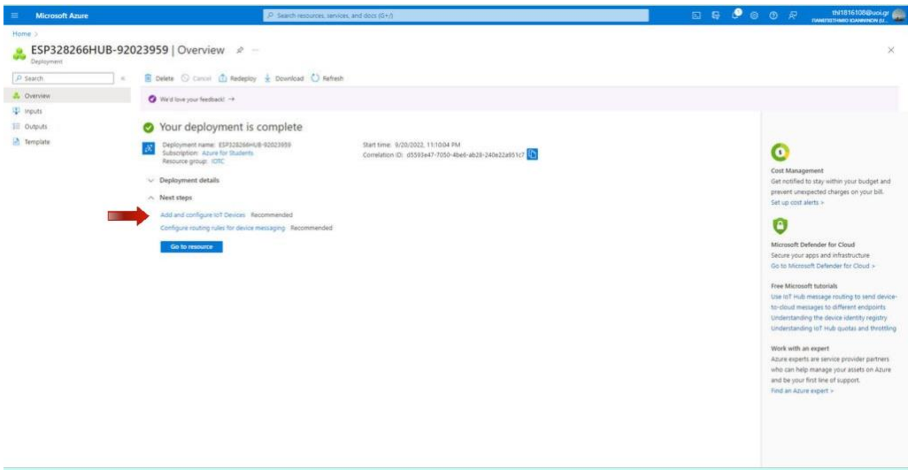
Instance details

IoT hub name * : ESP328266HUB

Region * : West Europe

Figure 42– Setting Up the IoT Hub.

After the user presses the "Review + Create" button, his payment and license will be reviewed along with the account's private payment details. Once everything is verified the user will be redirected into their created Hub page which can be viewed on the Figure 38:


Microsoft Azure

Home >

ESP328266HUB-92023959 | Overview
Deployment

Delete Cancel Redeploy Download Refresh

Overview
Inputs
Outputs
Template

We'd love your feedback! →

Your deployment is complete

Deployment name: ESP328266HUB-92023959
Subscription: Azure for Students
Resource group: IOTC

Start time: 9/20/2022, 11:10:04 PM
Correlation ID: d5593e47-7050-4be6-ab28-240e22a951c7

Deployment details

Next steps

Add and configure IoT Devices Recommended
Configure routing rules for device messaging Recommended

Go to resource

Cost Management
Get notified to stay within your budget and prevent unexpected charges on your bill.
Set up cost alerts >

Microsoft Defender for Cloud
Secure your apps and infrastructure
Go to Microsoft Defender for Cloud >

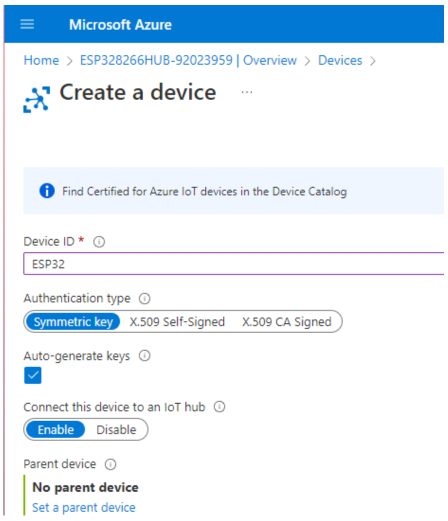
Free Microsoft tutorials
Use IoT Hub message routing to send device-to-cloud messages to different endpoints
Understanding the device identity registry
Understanding IoT Hub quotas and throttling

Work with an expert
Azure experts are service provider partners who can help manage your assets on Azure and be your first line of support.
Find an Azure expert >

Figure 43 – Custom IoT Hub created page.

Once the user is properly redirected into the overview hub's page, they will need to add and configure their IoT devices. In this step, both microcontrollers will become ready to be set-up through the IoT Platform's settings, this step will also provide unique keys that a microcontroller or a device will need to have for the device authentication and access.

The user will need to enter a device id along with many options at their disposal to choose from for authentication types, which can be seen on the Figure 39.


Microsoft Azure

Home > ESP328266HUB-92023959 | Overview > Devices >

Create a device

Find Certified for Azure IoT devices in the Device Catalog

Device ID *
ESP32

Authentication type
[✓ Symmetric key] X.509 Self-Signed X.509 CA Signed

Auto-generate keys
✓

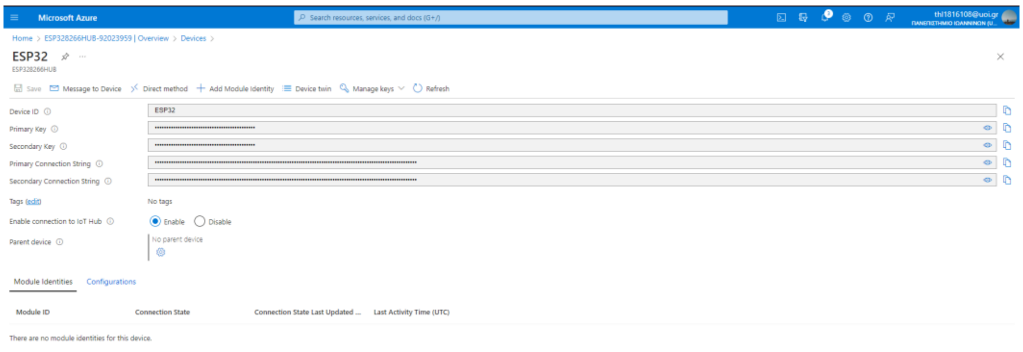
Connect this device to an IoT hub
[✓ Enable] Disable

Parent device
No parent device
Set a parent device

Figure 44 – Creating a device on the Azure IoT Hub.

After a few minutes have passed and the IoT Platform generates all the necessary data and encrypts it, the user will then have access to unique identification and key techniques, which can make a device, or in this case a microcontroller, connection very secure while at the same time assist the protection of sensitive data and data leakage.

Everything will be randomly generated and will only be accessible to the user (or organization) that is tied to the Azure IoT Hub.



ESP32
ESP328266HUB

Device ID	ESP32
Primary Key	••
Secondary Key	••
Primary Connection String	••
Secondary Connection String	••
Tags (edit)	No tags
Enable connection to IoT Hub	[✓ Enable] Disable
Parent device	No parent device

Module ID	Connection State	Connection State Last Updated ...	Last Activity Time (UTC)

There are no module identities for this device.

Figure 45 – Creating unique keys for the IoT Hub.

Once all the necessary steps have been completed without any issues, an IoT application can be built with the use of ESP32 & ESP8266 microcontrollers along with many different microcontrollers and devices.

However, because the data is difficult to be monitored and accessed since the Azure IoT Hub is using very strong Cloud Shell (bash) commands while require a lot of research, practice and experience with the Hub IoT documentation and the Microsoft Services in general that consistently change and get updated. For that reason, both applications will be connected to the Azure IoT Central and not the IoT Hub.

7.2.2 Accessing the Azure IoT Services with IoT Central

As mentioned before at Chapter 6.2, sending data to the Azure IoT Services with the assistance of the IoT Hub isn't the only acceptable method since Microsoft's Azure has a lot of tools, applications, and services at its disposal. The second and most popular method, which is also easier to be implemented, explained, and monitored which will be the transmission of data through the Azure IoT Central.

In order to begin the transmission of data, a user will be required to access the official Azure IoT website at https://apps.azureiotcentral.com/home and click on the "Build" button and then on "Create app" as the Figures 41 & 42 suggest. Alternatively, if a user has already completed this step, he can navigate into the My apps section.

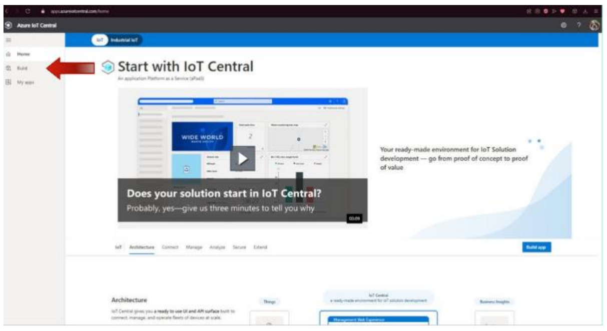


Figure 46 – Accessing the IoT Central.

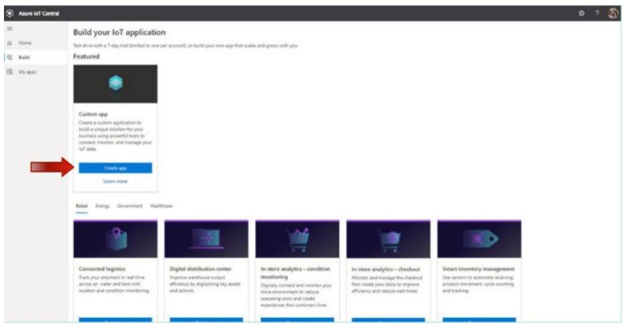
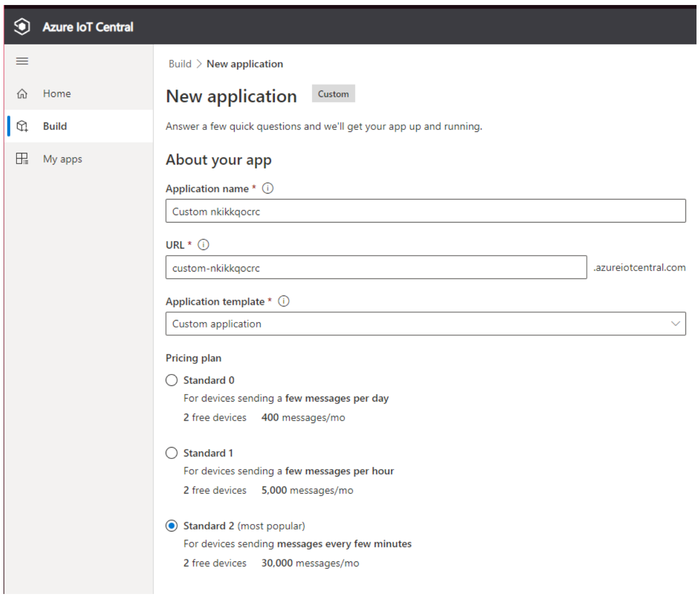


Figure 47 – IoT Central, building an application (1/2).

There a user will need to choose a desired billing plan with the number of devices that they want. Along with choosing an application name, a custom URL and an application template as the Figure 48 suggests.



New application Custom

Answer a few quick questions and we'll get your app up and running.

About your app

Application name *: Custom nkikkqocrc

URL *: custom-nkikkqocrc.azureiotcentral.com

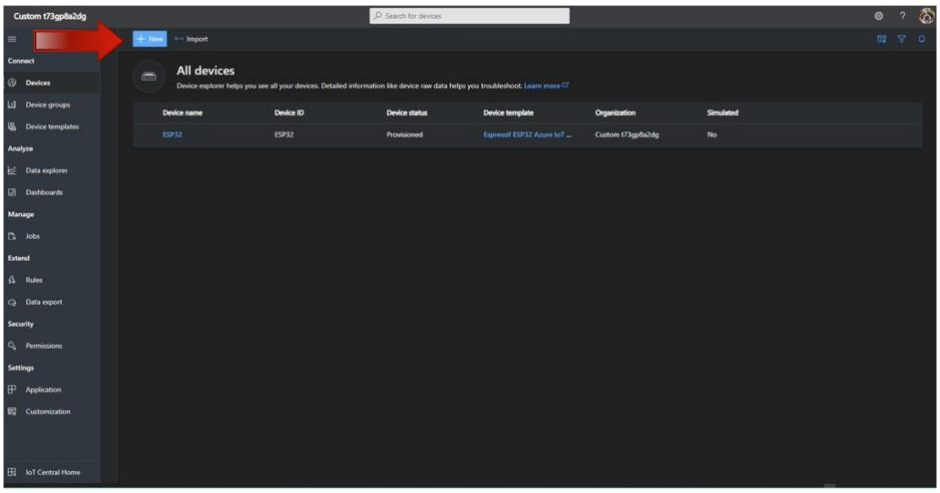
Application template *: Custom application

Pricing plan

- ○ Standard 0 — For devices sending a few messages per day — 2 free devices, 400 messages/mo
- ○ Standard 1 — For devices sending a few messages per hour — 2 free devices, 5,000 messages/mo
- ● Standard 2 (most popular) — For devices sending messages every few minutes — 2 free devices, 30,000 messages/mo

Figure 48 - IoT Central, building an application (2/2).

Once everything is completed, after a couple of minutes have passed for the IoT Central to create the application template along with all the settings, the user will be redirected to their application's portal. There he will need to add a device along many unique identifications which will all be used later into the Arduino projects.



All devices

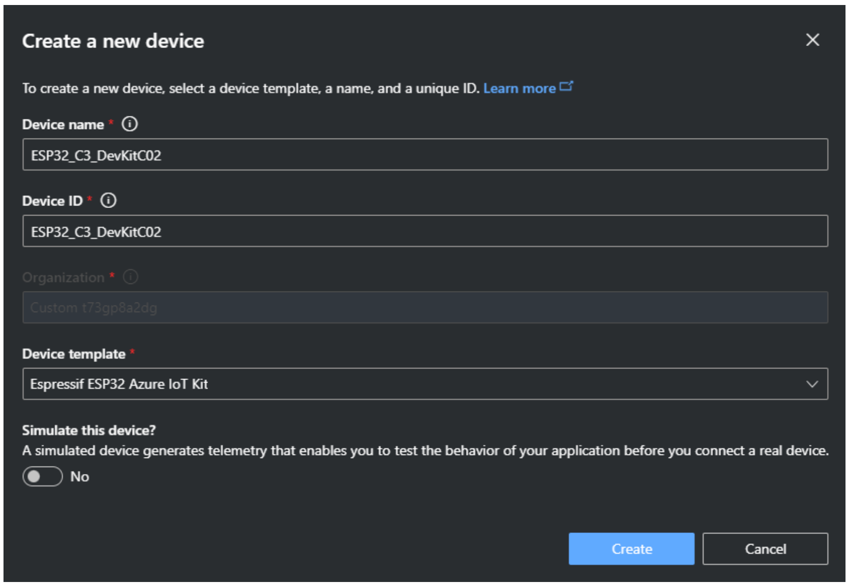
Device explorer helps you see all your devices. Detailed information like device raw data helps you troubleshoot. Learn more

Device name	Device ID	Device status	Device template	Organization	Simulated
ESP32	ESP32	Provisioned	Espressif ESP32 Azure IoT ...	Custom t73gp8a2dg	No

Figure 49 – Adding a Device into the Azure IoT Central (1/3).

The user will need to add two separate devices one for each microcontroller that will be used (one of the ESP32 Development Kit and one for the ESP32 C3 DevKitC02). By the time of this writing, I have already completed the process of adding a profile with unique identifications for the ESP32 microcontroller, however because the process is the same as adding any device, the following steps will explain how a user can add a device profile for the ESP32 C3 DevKitC02.

Firstly, user will need to enter a device name and a device id. For this example, I chose ESP32 C3 DevKitC02, which corresponds to the connection that will be established with the ESP32 C3 DevKitC02 microcontroller. Furthermore, a user can add a device template or even simulate a device which can assist the user to test how the devices will respond before connecting an actual device, which can make this a great feature for experienced developers. Keep in mind that for the following two applications, we will need to choose for now on the "Device Template" option which contains the pre-built template for Azure ESP32 Azure IoT Kit on both devices. As mentioned before, Chapter 7 will focus on the connection establishment and not the data visualization.


Create a new device

To create a new device, select a device template, a name, and a unique ID. Learn more

Device name *: ESP32_C3_DevKitC02

Device ID *: ESP32_C3_DevKitC02

Organization *: Custom t73gp8a2dg

Device template *: Espressif ESP32 Azure IoT Kit

Simulate this device?
A simulated device generates telemetry that enables you to test the behavior of your application before you connect a real device.
No

Figure 50 – Adding a Device into the Azure IoT Central (2/3).

Once all the devices have been added a user will need to gain access to the generated keys to add them into the Arduino projects later. To properly complete this step, a user will need to click to the “ESP32 C3 DevKitC02” then navigate to the Connect button on the upper left corner as the Figure 51 suggests.

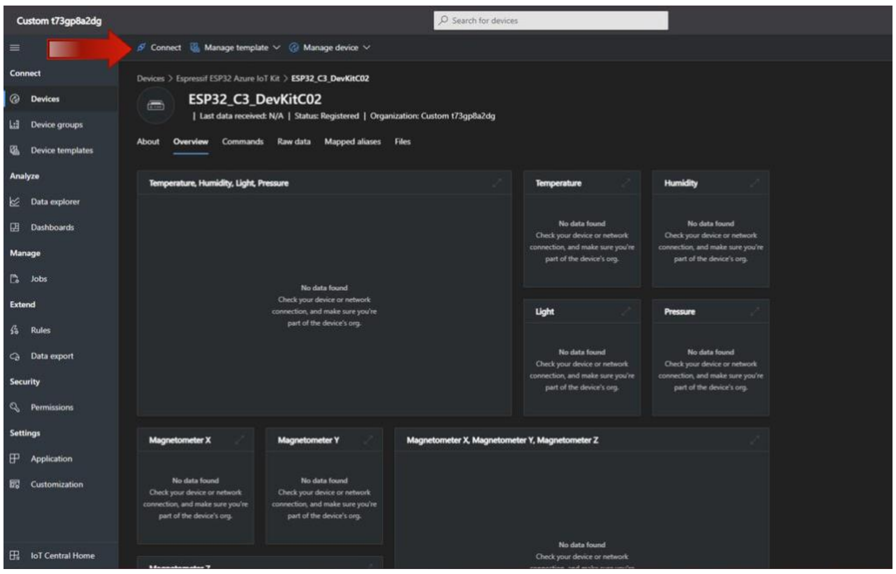
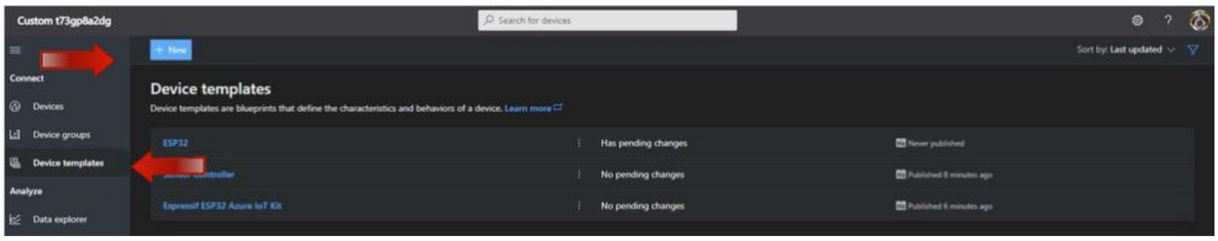


Figure 51 – Adding a Device into the Azure IoT Central (3/3).

As a precaution in case the Azure IoT Central – ESP32 Azure IoT Template isn’t available as a template for the user, to properly have the template imported a user will have to manually navigate to the options on the “Device Templates” corner and then click the “New” blue button like the Figure 47 suggests. There, he will need to navigate and find the “ESP-32 Azure IoT Kit”, which is also compatible with many of the available ESP32 models like the ESP32 Development Kit which will be currently used. There are many Device Templates available for many of the devices, which will make the creation of an application way easier instead of building a template from scratch and setting-up the telemetry and accessing the JSON messages.


Custom t73gp8a2dg

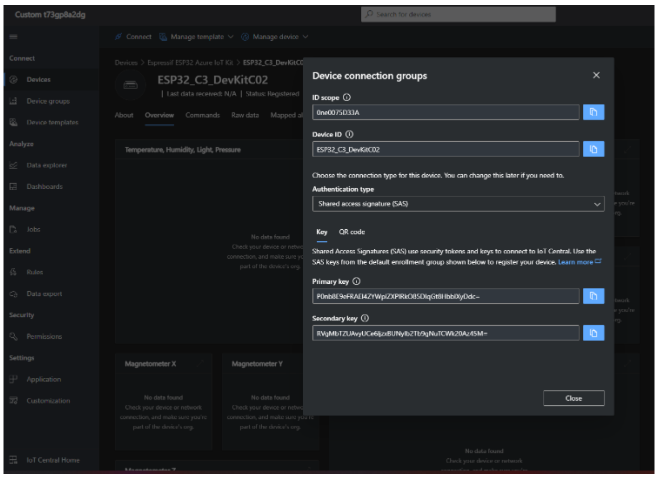
Device templates

Device templates are blueprints that define the characteristics and behaviors of a device. Learn more

ESP32	Has pending changes	Never published
[illegible]controller	No pending changes	Published 8 minutes ago
Espressif ESP32 Azure IoT Kit	No pending changes	Published 6 minutes ago

Figure 52 – Importing a Device Template.

Once the “Connect” button has been pressed, a new window will pop-up which gives the user many options to choose from and provides a lot of useful information which will all be used to establish a connection between the Azure IoT Central and a device or a microcontroller. This window also gives multiple options for key authentications (like Certificates X.509) which can be modified and added using Cloud Shell (bash) commands through Azure IoT Portal.


Device connection groups

ID scope: 0ne0075D33A

Device ID: ESP32_C3_DevKitC02

Choose the connection type for this device. You can change this later if you need to.

Authentication type: Shared access signature (SAS)

Shared Access Signatures (SAS) use security tokens and keys to connect to IoT Central. Use the SAS keys from the default enrollment group shown below to register your device. Learn more

Primary key: P0nb8E9eFRAEI4ZYWpIZXPIRkO85DIqGt8IIbblXyDdc=

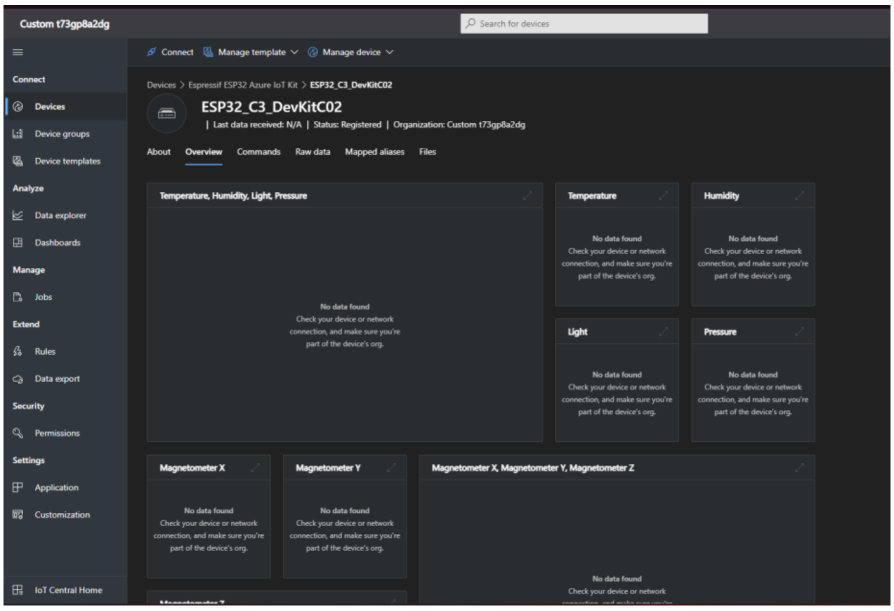
Secondary key: RVgMbTZUAvyUCe6ljzxBUNylb2Tb9gNuTCWk20Az45M=

Figure 53 – Accessing to the Azure IoT Central Device keys.

Once, the user saves all the keys he can use them on the Arduino Azure IoT Central example code provided by Microsoft. However there a few prerequisites that must be made too.
For the first step, a user will need to have the Arduino IDE installed along with both the ESP32 & ESP32 C3 DevKitC02 devices (Chapter 7.1.2 explains how this process can be done properly and more in-depth).

The second step is installing the necessary library Azure SDK For C which is made by Microsoft and can be found at the following URL through the Arduino’s official website https://www.arduino.cc/reference/en/libraries/azure-sdk-for-c/.

Once everything is completed both the applications can begin without any issues. Moreover, if the process is done correctly a user should see the overview page like the Figure 54 suggests.



No data found
Check your device or network connection, and make sure you're part of the device's org.

Figure 54 – Completing the Azure IoT Central Set-Up Process.

Not all the available options will be used for the applications which will follow, however, that doesn't prevent giving Microsoft from providing the users with examples for pre-built JSON data templates and giving the user a lot of brainstorming and project ideas that can be easily implemented. For now, the projects will strongly focus on the "Humidity", "Pressure" and "Temperature" values which will be sent by the following applications.

7.2.3 Espressif 32 (ESP32) Initialization with Azure IoT

In this application, my own ESPRESSIF32 Development Kit was used (also known as ESP32 Dev Module), which was briefly introduced during the Chapter 7.1.2. In addition to the ESP32 model, an external DHT11 sensor was used to measure the room temperature and the room humidity levels, which will be sent into the Azure IoT Central and will be able to be monitored by a user.

This work is based upon the pioneering work created officially by Microsoft and their official Azure IoT Central Documentation guides along with their official Arduino code temples, which are also part of the Azure SDK for C library package version 1.0.0 as of the time of this writing. The user will need to import the code which can be seen on Appendix C, along with the pre-built by Microsoft files that are provided in the official Microsoft library package for the Arduino IDE & the ESPRESSIF32 microcontrollers, so that the telemetry with JSON and MQTT with the use of telemetry message sending can be properly initialized and monitored throughout the Azure IoT Central.

For the current application, a user will be required to download the DHT sensor library provided by Adafruit and import it into the Arduino IDE (which can be downloaded directly from here: https://www.arduino.cc/reference/en/libraries/dht-sensor-library/, as of the time of this writing version 1.0.5 was used). A user will also need 3 Jumper Cables Male-to-Female which will be connected to the DHT11 pins. Lastly, the user will need a breadboard to connect the DHT11 sensor with the ESP32 microcontroller.

A few important details for the DHT11 sensor according to the official Adafruit website, is that DHT11 can achieve great humidity accuracy readings that can reach 5% along with temperature accuracy that can reach -+2% difference, while at the same time maintaining an ultra-low cost (Adafruit,2022).

The result of this application once the code is uploaded successfully along with all necessary files that are required and provided by Microsoft, while also configuring the Device authentication and the personal ID Scope and the Device ID, the end-result can be seen on the Figure 55.

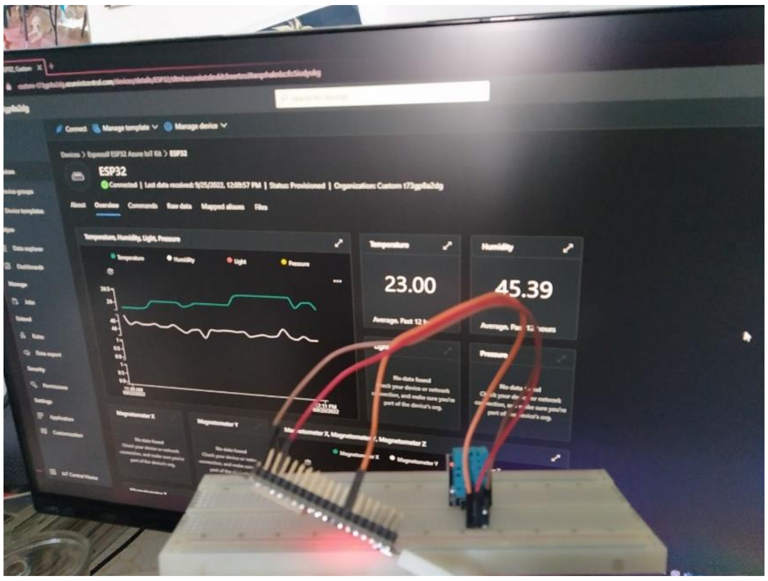


Figure 55 – Sending Data with the ESP32 into the Azure IoT Central.

As the Figure 55 suggests, the DHT11 sensor is gathering the temperature (in Celsius) and humidity (in Relative Humidity) values and then with the use of telemetry, JSON request messages with the MQTT standard, they are sent into the Azure IoT Central where they can be monitored and accessed anytime the user wants to. There, a communication between the ESP32 microcontroller connected with the DHT11 sensor and the Azure IoT Central is successfully established.

Once the data is received through the Azure IoT Central, a user can access it and debug it if needed and even investigate all the actions that the microcontroller took and sent by going to the "Raw Data" section, on the microcontroller device profile.

The code that was used is available in "APPENDIX C", however as mentioned before the used will need to manually install the rest of the Arduino files that are available in their official Arduino Library package for the code to properly work as intended.

The user is also highly advised to read the official library documentation, along with researching the official Microsoft Azure IoT forums, to fully understand how the Azure IoT Central oriented commands in the Arduino IDE and the libraries that are provided, function in general.

The current process may require an experienced microcontroller programmer with Internet of Things applications that also fully understands how the JSON message transmission and MQTT standard operate, which might make the Azure IoT Central along with the Azure IoT services not an attainable option for developers with little to none experience, however, the results that can be achieved are tremendous and the options that the Azure IoT provides to a developer are very powerful and can be further accustomed to suit their own personal requirements.

7.2.4 ESPRESSIF32 C3 DevKitC02 (ESP32 C3 DevKitC02) Initialization with Azure IoT

The same application can be made using the ESP32 C3 DevKitC02 microcontroller with the Adafruit DHT22 Sensor to monitor and transmit the temperature and the humidity in the Azure IoT Central.

Figure 56 – ESPRESSIF 32 C3 DevKitC02.

ESPRESSIF 32 C3 DevKitC02, according to the official ESPRESSIF product documentation, belongs to the series C3 of SoCs and it's an integrated microcontroller unit that has 2.4 GHz Wi-Fi and Bluetooth Low Energy (BLE) capabilities. It is also part of the ESPRESSIF32 (ESP32) family of microcontrollers. The current microcontroller is priced around 20-35 Euros (Amazon, 2022a).

A few noteworthy highlights that this microcontroller has at its disposal is a 32-Bit RISC-V single core processor, reliable security features, state of the art power and RF Performance along with storage capabilities [116]. This can in turn, make this microcontroller an ideal choice for ESP32 enthusiasts and IoT Developers alike who want to create low-power applications while at the same time having IoT-ready capabilities.

The DHT22 sensor is a low-cost sensor with 0-100% humidity readings with a 2-5% accuracy and good for -40 to 80 Celsius +- 0.5% accuracy difference. Meanwhile the DHT11 sensor has 20-80% humidity readings with a 0 to 50 Celsius readings +-2 Celsius difference, therefore making the DHT22 a better option to measure accurate details (Amazon.com,2021).

The ESPRESSIF32 C3 DevKitC02 along with the Adafruit DHT22 Sensor were provided to me kindly by the Autonomous Systems Laboratory of the Department of Informatics & Telecommunications in the University of Ioannina.

For the current application, a user will be required to download the DHT sensor library provided by Adafruit and import it into the Arduino IDE (which can be downloaded directly from here: https://www.arduino.cc/reference/en/libraries/dht-sensor-library/ as of the time of this writing version 1.0.5 was used). A user will also need 3 Jumper Cables Male-to-Female, a 10 kΩ resistance which will be connected to the DHT11 pins. Lastly, the user will need a breadboard to connect the DHT11 sensor.

Once everything is connected properly and the code that is provided in the APPENDIX D, along with the Microsoft provided files, compiles without any errors, while also configuring the personal Device keys, the temperature (in Celsius) and the humidity (in Relative Humidity) can be accessed and monitored through the Azure IoT Central.

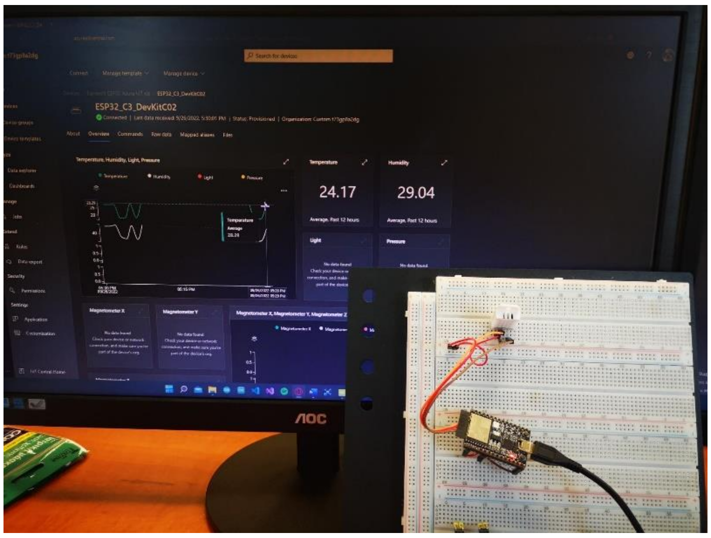


Figure 57 - Sending Data with the ESP32 C3 DevKitC02 into the Azure IoT Central.

8 IoT Platforms: Data Visualization & Data Storage

Figure 58- An illustration portraying Data Visualization.

Once a device properly transmits the data through the supported device (or in this case, a microcontroller), a user can access the data through the Internet of Things related platform. However, an IoT Platform doesn't only contain options to send the data and monitor the data remotely. This is because an Internet of Things developer has different requirements and where he must build a lot of different applications and monitor them in a unique way while also exporting the data into charts to draw logical conclusions.

Furthermore, an Internet of Things developer will need to access the data on certain times of the day and compare it to the data that he received on a specific time frame without being losing any of the data that he has so far gathered.

A solution to the above-mentioned problems would be to export somehow all the data into an external program, however that would be time consuming while also losing a lot of valuable data and most importantly, resources. Therefore, the IoT Platforms had massive problems that they needed to deal with: Data Storage and Data Visualizations.

One of the key requirements that an IoT Developer wants is to draw logical conclusions while dealing while also storing the data and accessing it whenever he needs to without the loss of any valuable information that he has so far acquired and sent into the IoT Platform. Furthermore, all the stored data will be needed to be compared, illustrated, and visualized accustomed to the Developer's and the Client's requirements.

These problems including many others, are a key problem that many of the above-mentioned Internet of Things Platforms of Chapter 6, have all faced and dealt with. Each of the above-mentioned Internet of Things Platforms provide their own unique ways of handling and encrypting and storing the data through the many of their Platforms settings, including, many ways that a user can access the gathered data, visualize, and draw logical conclusions. Moreover, the IoT Platforms have also made sure that the data handling and storage access is much as easy to handle as possible for the user to learn and use.

This Chapter will focus on the second part of utilizing an Internet of Things Platform, storing the data and visualize it through the platform's provided settings in two of the previously mentioned Internet of Things Platforms: The Azure IoT Central and the Thinger.io. An interesting sidenote that can be added is overall difference that the 2 Platforms offer – for example the cost which plays a significant role in the following two applications.

More specifically, the cost for the Azure IoT Platform can cost a total of 152.62 US dollars just for one month of use, with additional subscription service fee of 152.62 US Dollars per month (Azure,2022d), 9 US Dollars for the purchase of the microcontroller (ESP32 C3 DevKitC02) (Digi Electronics,2022) and 9 US Dollars for the DHT22 Sensor] (Industries,n.d.), and surprisingly enough, even though Microsoft does offer an extensive documentation with many articles officially written by their website, the Azure IoT Central doesn't seem to have enough community support which can make the connection establishment between a microcontroller and the Azure IoT Central a very task, especially for beginner IoT developers. However, with enough experience the capabilities that the Azure IoT Central can offer are limitless with very powerful tools at their disposal that an experienced developer can very easily use like creating Permissions, Data Export adding rules and so much more.

On the other hand, Thinger.io even though it is very limited and restricted, it is open source. For example, the same application that was made on 7.2.4, can be made with an ESP8266, which costs around 4 US Dollars (Amazon,2022c) and a DHT11 Sensor which costs around 5 US Dollars (Mouser Electronics,2022), which roughly translates to a 9 US Dollars total! Plus, the Thinger.io platform has a massive community with many projects that are already made, which can make it perfect for IoT beginner developers, However, it doesn't have as many features or security measures as the Azure IoT Central has or that many supported devices among many other features. On the Chapters 8.1 & 8.2 which will follow a presentation will be made which will present the two IoT platforms by showcasing each one of the features that both of the IoT Platforms have their disposal. This Chapter also aims to showcase how Data Visualization can be made depending on the platform, along with how a data storage can be made.

8.1 Azure IoT Central: Data Visualization & Data Storage

The code to transmit the data along with the microcontroller is the same as the one that was previously mentioned on Chapter 7.2.3, which can be found on APPENDIX D. More specifically, the data transmission will be made using the ESPRESSIF32 C3 DevKitC02 (ESP32 C3 DevKitC02) along with the DHT22 Temperature & Humidity Sensor. By going through the dashboard and configuring a widget along with the appropriate settings a user can represent their data very easily with powerful customization a user can create a dashboard as the Figure 59 suggests:

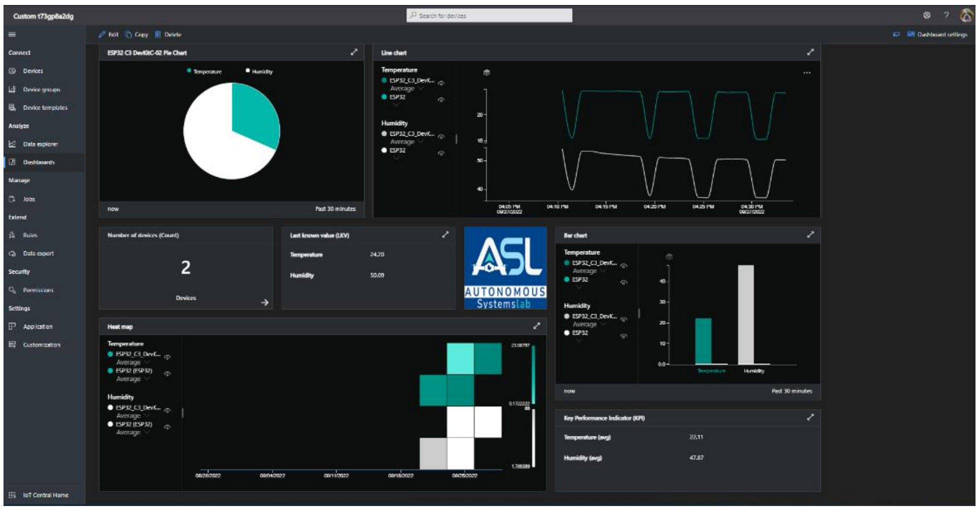


Figure 59 – An example of an Azure IoT Central dashboard.

A few of the options that a dashboard widget usually contains can be seen on Figure 60.

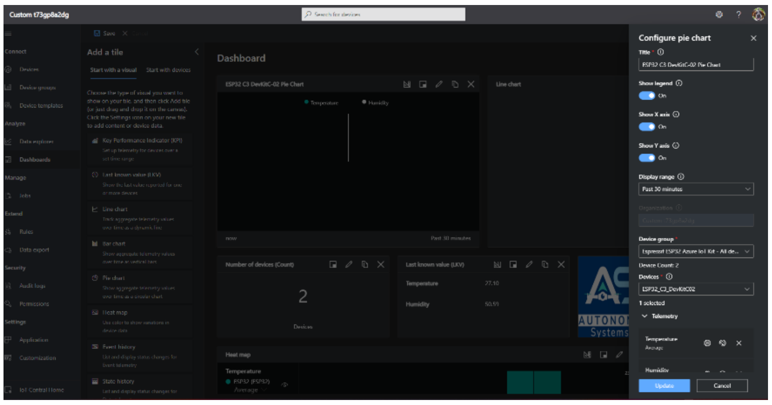


Figure 60 – Azure IoT Central - Typical Dashboard settings

The widgets can be very easily edited and created with multiple different options a user can choose from.

Moreover, according to the official Microsoft Azure IoT Central documentation, data retention can reach 30 days automatically. If the days exceed, all the so far gathered data can be exported. Furthermore, the number of devices can reach 1.000.000 can be surpassed through contacting an official Microsoft representative (Dominicbetts,2022).

8.2 Thinger.io: Data Visualization & Data Storage

However, Azure IoT Central isn't the only Internet of Things Platform that has many useful tools and features to visualize the gathered data and store it into the IoT Platform.

Thinger.io, also provides tools and features where a user can access and store the data that are accustomed to their individual requirements. For the current experiment my ESPRESSIF8266 (also known as ESP8266) was used along with a DHT11 Temperature &

Humidity sensor, that the Autonomous Systems Laboratory has thoughtfully provided me with for the completion of my thesis.

The code that was used can be found on APPENDIX E. To properly compile the code, a user must ensure that the Adafruit DHT11 sensor library is installed, along all the prerequisites including the ESP8266 Board Support and of course the Thinger.io library package, the process of installing the appropriate libraries along with setting-up the ESP32 & ESP8266 Boards through the Arduino IDE is explained more in depth on the Chapter 7. Lastly, he will need to add a device through the Thinger.io device, the process was explained on Chapter 7.

A few noteworthy information about the ESPRESSIF8266 or more commonly known as ESP8266, is that this microcontroller is the predecessor of the ESP32 family series. ESP8266 runs on a Tensilica L106 32-Bit processor with integrated Wi-Fi SoC solutions along with efficient power usage functionality. A few more key features that the ESP8266 has at its disposal are built-in power amplifiers and power management modules (ESPRESSIF,n.d.).

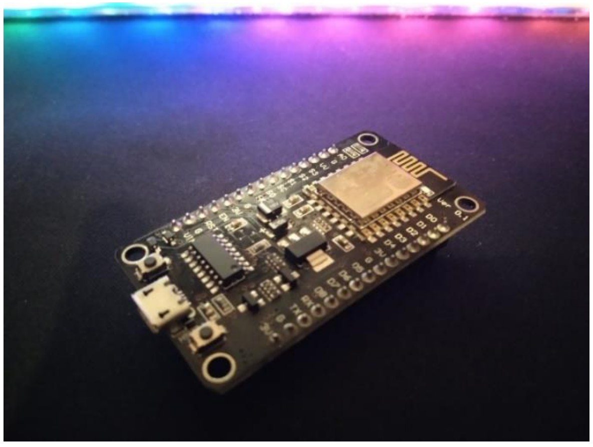


Figure 61 – The ESP8266 microcontroller.

However, the best feature that the ESP8266 has to offer is the price, which costs around 4-6 Euros (Amazon, 2022b). Therefore, making this microcontroller the best price/value microcontroller option in my opinion where there is also an extensive documentation for people who are just getting-started with Internet of Things & Wi-Fi related applications.

Unfortunately, as of the time of this writing, Data Buckets became a premium option, and a user is unable to access and store data without choosing a premium plan. However, Thinger.io also offers free dashboard options for users to use and illustrate their data as they see fit.

With the use of dashboards, a user can very easily by adding a widget can represent their data according to their own personal requirements an interesting example that I've very easily made can be seen as the Figure 62 suggests.

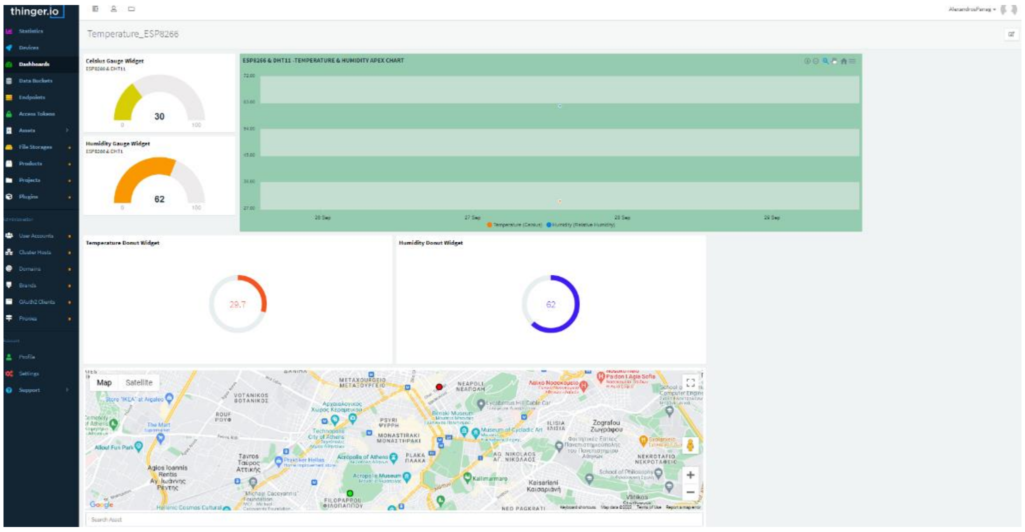


Figure 62 – Thinger.io Data Visualization.

9 Conclusions

To summarize, Internet of Things has a long history and will continue to have a long future too, considering how fast it has evolved and will continue to evolve as our requirements become different and more demanding.

Internet of Things can move the technology forward and yield great opportunities for many aspects that constantly surround of our daily lives like agriculture, industrial production, our eco-footprint, our smart cities, our healthcare, our homes, and the applications that we consider mundane and much, much more.

As the term Internet of Things continues to evolve, so will its definition and the future vision of what it can accomplish and can overcome, at the same time, many technologies may become part of the Internet of Thing’s vision and its definition. Many if not all, the domains of what the Internet of Things can accomplish, will require a platform to function where platforms will offer different and unique advantages. In turn, it might make the decision of picking a certain platform over a different one a very difficult decision. When it all comes down to, as our requirements expand, the technology evolves, and our requirements increase so will the vision of what the Internet of Things is, what it will evolve eventually into and what is can become will change and expand upon.

Image Sources

Figure 1 - Kevin Ashton. (Online: 17/06/2022)

https://static.politico.com/7c/56/9b80979549c7b765f2ed82272462/newlede-usleadership.jpg

Figure 2 – Auto ID Labs Logo. (Online: 17/06/2022)

https://en.wikipedia.org/wiki/Auto-ID_Labs#/media/File:Auto-ID_Labs-logo.jpg

Figure 3 – An illustration of the Agriculture Internet of Things. (Online: 21/06/2022)

https://www.smart-akis.com/wp-content/uploads/2016/10/iot-farming.jpg

Figure 4 – An illustration of the Industrial Internet of Things. (Online: 16/06/2022)

https://miro.medium.com/max/722/1*hoVBU2OT1oil2wL-BMznaA.jpeg

Figure 5 – An illustration of the Green Internet of Things. (Online: 21/06/2022)

https://www.speranzainc.com/wp-content/uploads/2021/10/未命名-2.jpg

Figure 6 – An illustration of a smart city. (Online: 21/06/2022)

https://smartcircularcity.isi.gr/wp-content/uploads/2021/05/smart-4308821_1920.jpg

Figure 7 - An illustration of smart applications that are connected to the Internet of Things. (Online:22/06/2022)
https://miro.medium.com/max/860/1*ONYIlo3Zdtk6ZerPkj0ELg.jpeg

Figure 8 - An illustration of how the Ambient Intelligence works as a general idea. (Online:08/07/2022) https://www.semanticscholar.org/paper/Ambient-Intelligence%3A-

Concepts-and-applications-Augusto-Mccullagh/2b64fae8054ebffd3e41ebe0cf7eb6c70f4725f9/figure/0

Figure 9 – A picture of Gordon Moore, author of Moore's Law. (Online:08/07/2022) https://images.computerhistory.org/siliconengine/1965-1-1.jpg

Figure 10 – A picture illustrating the interoperability between computer systems. (Online:18/07/2022)

https://www.articlestheme.com/wp-content/uploads/2022/04/workflow-automation-for-healthcare-interoperability-intely.jpg

Figure 11 - IoT taxonomy of interoperability. (Online:19/07/2022)

https://link.springer.com/content/pdf/10.1007/s11036-018-1089-9.pdf

Figure 12 – The official Thinger.io logo (2022). (Online:14/07/2022)

https://s3.eu-west-1.amazonaws.com/thinger.io.files/logos/thinger_logo/thinger_logo.png

Figure 13 – An illustration of the IoT standards. (Online:20/07/2022)

https://www.jstage.jst.go.jp/article/ipsjjip/25/0/25_23/_pdf

Figure 14 – Thinger.io Pricing plans as of 2022. (Online:14/07/2022)

https://pricing.thinger.io/#!/cloud

Figure 15 – The Thinger.io platform architecture. (Online:28/09/2022)

https://docs.thinger.io

Figure 16 - Azure IoT products and services by Microsoft. (Online:14/07/2022)

https://azure.microsoft.com/en-us/overview/iot/#products

Figure 17 - The architecture of Azure IoT Central. (Online:28/09/2022)

https://learn.microsoft.com/en-us/azure/iot-central/core/concepts-architecture

Figure 18 – ThingsBoard official logo. (Online:15/07/2022)

https://camo.githubusercontent.com/b952842a7ff1f28a5eae3c6f5ec42346da0ff04497ad011
41e57f2383e739695/68747470733a2f2f7468696e6773626f6172642e696f2f696d61676573
2f7468696e6773626f6172645f6c6f676f2e706e67

Figure 19 – The ThingsBoard architecture (Online:29/09/2022)

https://thingsboard.io/docs/reference/

Figure 20 – Amazon Web Services IoT official icon. (Online:15/07/2022)

https://aws.amazon.com/iot/

Figure 21- AWS IoT connecting IoT devices to the AWS services. (Online:07/08/2022)

https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

Figure 22 – Google Cloud IoT official icon. (Online:15/07/2022)

https://www.gstatic.com/devrel-devsite/prod/vc168bdd3636a4d0c346e3f7ef8a8dc9473171e3b493766c11713b40799c9ad55/cloud/images/social-icon-google-cloud-1200-630.png

Figure 23 – IBM Watson official icon. (Online:20/07/2022)

https://eetech.com/wp-content/uploads/2021/11/IBM-Watson-IoT.png

Figure 24 – How devices operate with the IBM Watson IoT. (Online:07/08/2022)

https://internetofthings.ibmcloud.com/

Figure 25 – The Oracle Internet of Things Cloud Service Architecture. (Online:29/08/2022)

https://docs.oracle.com/en/solutions/internet-of-things-options-connect-devices/index.html#GUID-2C6B480E-812A-48DE-96F9-AF0F3C565487

Figure 26 – OpenRemote official logo. (Online:16/08/2022)

https://openremote.io/wp-content/uploads/2020/06/OpenRemote-Logo.png

Figure 27 - The OpenRemote Platform Architecture. (Online:30/09/2022)

https://www.openremote.io/developers/

Figure 28 - The official Thinger.io website. (Online:25/08/2022)

https://thinger.io

Figure 29 - Thinger.io sign up page. (Online:25/08/2022)

https://console.thinger.io/signup

Figure 30 – The Thinger.io console. (Online 25/08/2022)

https://console.thinger.io/console/statistics

Figure 31 – Adding a device into your account. (Online:25/08/2022)

https://console.thinger.io/console/devices/add

Figure 32 – Thinger.io Device Console. (Online:25/08/2022)

https://console.thinger.io/console/devices/CC3200/status

Figure 33 – CC3200 Launch XL

Figure 34 – CC3200 sending the LED States into the Thinger.io Platform.

Figure 35 - Adding Additional Board Manager URLs. (Online:30/08/2022)

https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

Figure 58 – An illustration portraying the Data Visualization (Online:27/09/2022)

https://boostlabs.com/wp-content/uploads/2019/09/10-types-of-data-visualization-1.jpg

References

A

Abbasi, M., Yaghmaee, M. H., & Rahnama, F. (2019). Internet of things in agriculture: A survey. *2019 3rd International Conference on Internet of Things and Applications (IoT)*. Available at: https://doi.org/10.1109/iicita.2019.8808839 .(Online 17/06/2022)

Aarts, E., & Wichert, R. (2009a). Ambient intelligence. *Technology Guide*, pp.: 244–249. Available at: https://doi.org/10.1007/978-3-540-88546-7_47 (Retrieved at:08/07/2022)

Aarts, E., & de Ruyter, B. (2009b). New research perspectives on Ambient Intelligence. *Journal of Ambient Intelligence and Smart Environments*, *1*(1), pp.5–14. Available at: https://doi.org/10.3233/ais-2009-0001(Retrieved at:08/07/2022)

Adafruit. (2022). *DHT11, dht22 and AM2302 sensors*. Adafruit Learning System. Available at: https://learn.adafruit.com/dht . (Retrieved at:26/09/2022)

Ahonen, P., & Wright, D. (2008). *Safeguards in a world of ambient intelligence*. Springer.

Alleven, M. (2022). *Google Cloud IOT shutdown likely to have minimal impact on csps: MachNation*. Fierce Wireless. Available at: https://www.fiercewireless.com/tech/google-cloud-iot-shutdown-likely-have-minimal-impact-csps-machnation#:~:text=The%20service%20is%20being%20retired%20on%20August%2016%2C%202023 (Retrieved at:29/09/2022)

Alshqaqi, S. A., Zahary, A. T., & Zayed, M. M. (2019). Ubiquitous computing environment: Literature review. *2019 First International Conference of Intelligent Computing and Engineering (ICOICE)*. Available at: https://doi.org/10.1109/icoice48418.2019.9035157 (Retrieved at:13/07/2022)

Al-Qaseemi, S. A., Almulhim, H. A., Almulhim, M. F., & Chaudhry, S. R. (2016). IOT architecture challenges and issues: Lack of standardization. *2016 Future Technologies Conference (FTC)*. Available at: https://doi.org/10.1109/ftc.2016.7821686. (Retrieved at:20/07/2022)

Amazon.com,(2019). *ESP32 development board wireless wifi+bluetooth dual core module with ...* (1996). Available at: https://www.amazon.com/Development-Wireless-Bluetooth-Module-ESP32-D0WDQ6/dp/B07KTV2RRM. (Retrieved at:30/08/2022)

Amazon.com,(2021). *DHT11, dht22 and AM2302 sensors - adafruit industries*. Available at: https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf. (Retrieved at:26/09/2022)

Amazon.(2022a). *RCmall ESP32-C3-DEVKITC-02 development board Wi-Fi+BT+BLE 4MB SPI Flash based on Espressif ESP32-C3-WROOM-02 module (1 piece)*. Amazon.co.uk: Computers & Accessories. Available at: https://www.amazon.co.uk/RCmall-ESP32-C3-DevKitC-02-Development-Espressif-ESP32-C3-WROOM-02/dp/B09GVG5BLK/ref=sr_1_fkmr0_2?crid=1Z9QIYW2HD14O&keywords=ESPRESSIF%2B32%2BC3%2BDevKitC02&qid=1664229206&sprefix=espressif%2B32%2Bc3%2Bdevkitc02%2Caps%2C486&sr=8-2-fkmr0&th=1. (Retrieved at:27/09/2022)

Amazon.(2022b). *Azdelivery NODEMCU WIFI Lolin v3 ESP8266 ESP-12F internet development board module (32mbit) Flash memory chip series port adapter with CH340 compatible with Arduino including e-book!* Amazon.co.uk: Computers & Accessories. (2021). Available at: https://www.amazon.co.uk/AZDelivery-NodeMcu-ESP8266-Development-including/dp/B06Y1ZPNMS/ref=sr_1_11?crid=3H90X5LQIXTWB&keywords=generic%2Besp8266&qid=1664232201&sprefix=generic%2Besp8266%2Caps%2C100&sr=8-11&th=1. (Retrieved at:27/09/2022)

Amazon. (2022c). *Amazon.com: Wireless Module NodeMcu V3 CH340 Lua WIFI internet of* ... Available at: https://www.amazon.com/Wireless-NodeMcu-Internet-Development-ESP8266/dp/B08F7HNGRT (Retrieved at:29/09/2022)

AWS.(2022a). *IOT: L'émancipation des objets. Amazon*. Available at: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html (Retrieved at:15/07/2022)

AWS.(2022b). IoT Core Easily and securely connect devices to the cloud. Amazon. Available at: https://aws.amazon.com/iot-core/?c=i&sec=srv (Retrieved at:15/07/2022)

AWS.(2022c). *IoT Solution Repository. Amazon.* Available at: https://aws.amazon.com/iot/solutions/?iot-solution-repository-cards.sort-by=item.additionalFields.headline&iot-solution-repository-cards.sort-order=asc&awsf.iot-solution-repository-filter-industry=%2Aall&awsf.iot-solution-repository-filter-products=%2Aall&awsf.iot-solution-repository-filter-usecase=%2Aall&awsf.iot-solution-repository-filter-solution-type=%2Aall (Retrieved at:15/07/2022)

AWS.(2022d). *IoT Core pricing*. *Amazon*. Available at: https://aws.amazon.com/iot-core/pricing/ (Retrieved at:15/07/2022)

Azure. (2021a). *Twins – Modeling and simulations: Microsoft Azure*. Digital Twins – Modeling and Simulations | Microsoft Azure.Available at: https://azure.microsoft.com/en-us/services/digital-twins/#overview (Retrieved at:14/07/2022)

Azure.(2022a). *Azure Percept: Edge computing solution: Microsoft Azure*. Azure Percept | Edge Computing Solution | Microsoft Azure. Available at: https://azure.microsoft.com/en-us/services/azure-percept/#features (Retrieved at:14/07/2022)

Azure.(2022b). *Azure IOT Central - IOT solution development: Microsoft Azure*. IoT Solution Development | Microsoft Azure. Available at: https://azure.microsoft.com/en-us/services/iot-central/#overview (Retrieved at:14/07/2022)

Azure.(2022c). *Azure sphere – IOT device security platform: Microsoft azure*. – IOT Device Security Platform | Microsoft Azure. Available at: https://azure.microsoft.com/en-us/services/azure-sphere/#overview (Retrieved at:14/07/2022)

Azure.(2022d). *Pricing calculator: Microsoft Azure*. Pricing Calculator | Microsoft Azure. Available at: https://azure.microsoft.com/en-us/pricing/calculator/ (Retrieved at:15/07/2022)

B

Balaji, S., Nathani, K., & Santhakumar, R. (2019). IOT technology, applications and challenges: A contemporary survey. *Wireless Personal Communications*, *108*(1),pp. 363–388. Available at: https://doi.org/10.1007/s11277-019-06407-w. (Retrieved at:22/06/2022)

Bashar, D. A. (2019). Review on Sustainable Green Internet of things and itsapplication. *IRO Journal on Sustainable Wireless Systems*, *1*(04),p.p.:256–264. Available at: https://doi.org/10.36548/jsws.2019.4.006 . (Retrieved at:20/06/2022)

Bibri, S. E. (2015). The shaping of ambient intelligence and the internet of things. *Atlantis Ambient and Pervasive Intelligence*. Available at: https://doi.org/10.2991/978-94-6239-142-0. (Retrieved at: 08/07/2022)

Bui, N., & Zorzi, M. (2011). Health Care Applications. *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies - ISABEL '11*. Available at: https://doi.org/10.1145/2093698.2093829 (Retrieved at:22/06/2022)

C

Chen, Y.K. & Kung, S.Y. (2005). Trends and challenges with system-on-chip technology for Multimedia System Design. *Conference, Emerging Information Technology 2005.* Available at: https://doi.org/10.1109/eitc.2005.1544365. (Online 24/06/2022)

Chace, J. (2013). In White paper, Texas Instruments. *The evolution of the internet of things - ti.com*. September,2013. Available at: https://www.ti.com/lit/pdf/swrb028 . (Online 24/06/2022)

Chaouchi, H., & Chaouchi, H. (2010). *The internet of things connecting objects to the web*. ISTE. Available at: https://link.springer.com/content/pdf/10.1007/978-3-319-99516-8.pdf . (Retrieved at:16/06/2022)

Cording, S., & Cording, S. (2022). *What is the ESP32? its brief history and how to get started*. Elektor. Available at: https://www.elektormagazine.com/articles/what-is-the-esp32. (Retrieved at:30/08/2022)

D

Dahlqvist, F., Patel, M., Rajko, A., & Shulman, J. (2019). *Growing opportunities in the internet of things*. (n.d.). At Mac Kinsey and Company. July 2019. Available at: https://www.mckinsey.com/~/media/McKinsey/Industries/Private%20Equity%20and%20Principal%20Investors/Our%20Insights/Growing%20opportunities%20%20in%20the

%20Internet%20of%20Things/Growing-opportunities-in-the-Internet-of-Things-v5.pdf .(Online 24/06/2022)

Dameri, R. P. (2017). *Smart city implementation: Creating economic and public value in innovative urban systems*. Springer.

Digi. Electronics. (2022). *ESP32-C3-DEVKITC-02: Digi-Key Electronics*. Available at: https://www.digikey.com/en/products/detail/espressif-systems/ESP32-C3-DEVKITC-02/14553009 (Retrieved at:30/09/2022)

Dominicbetts. (2022). *Azure IOT central quotas and limits*. Azure IoT Central quotas and limits Microsoft Learn. Available at:https://learn.microsoft.com/en-us/azure/iot-central/core/concepts-quotas-limits (Retrieved at:30/09/2022)

E

Ebling, M. R. (2016). Pervasive computing and the internet of things. *IEEE Pervasive Computing*, *15*(1),pp.2–4. Available at: https://doi.org/10.1109/mprv.2016.7 (Retrieved at:13/07/2022)

Elijah, O., Rahman, T. A., Orikumhi, I., Leow, C. Y., & Hindia, M. H. D. N. (2018). An overview of internet of things (IOT) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of Things Journal*, *5*(5), 3758–3773. Available at: https://doi.org/10.1109/jiot.2018.2844296 (Retrieved at:16/06/2022)

ESPRESSIF.(n.d.). Alldatasheet.com. *ESP8266EX datasheet(pdf) - ESPRESSIF systems (shanghai) co., ltd.* Electronic Parts Datasheet Search. Available at: https://www.alldatasheet.com/datasheet-pdf/pdf/1148030/ESPRESSIF/ESP8266EX.html. (Retrieved at:27/09/2022)

ESPRESSIF.(2022a). *Technical reference manual - ESPRESSIF*. Available at: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (Retrieved at:30/08/2022)

ESPRESSIF.(2022b). ESP*32C3 series - espressif*. Available at: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf . (Retrieved at:26/09/2022)

Estevez, C., & Wu, J. (2015). Recent advances in Green Internet of Things. *2015 7th IEEE Latin-American Conference on Communications (LATINCOM)*. Available at: https://doi.org/10.1109/latincom.2015.7430133 . (Retrieved at:20/06/2022)

G

Gaikwad, P. P., Gabhane, J. P., & Golait, S. S. (2015). A survey based on Smart Homes System using internet-of-things. *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*. Available at: https://doi.org/10.1109/iccpeic.2015.7259486 (Retrieved at: 20/06/2022)

Garrity, J. (2015). Harnessing the internet of things for global development. *SSRN Electronic Journal*. Available at: https://doi.org/10.2139/ssrn.2588129 . (Retrieved at: 7/06/2022)

González-Usach, R.; Yacchirema-Vargas, DC.; Julián-Seguí, M.; Palau Salvador, CE. (2019). Interoperability in IoT. Handbook of Research on Big Data and the IoT. 149-173. Available at: http://hdl.handle.net/10251/150250 (Retrieved at:18/07/2022)

Google. (2022a). *Cloud IOT core | google cloud*. Google. Available at: https://cloud.google.com/iot-core (Retrieved at:15/07/2022)

Google. (2022b). *Google cloud IOT - fully managed IOT services*. Google. Available at: https://cloud.google.com/solutions/iot. (Retrieved at:15/07/2022)

H

Hansmann, U. (2011). *Pervasive computing: The mobile world*. Springer.

Hassan, Q. F. (2018). *Internet of things A to Z: Technologies and applications*. John Wiley and Sons, Inc.

I

IBM.(n.d.a). *A computer called Watson*. Available at: https://www.ibm.com/ibm/history/ibm100/us/en/icons/watson/ (Retrieved at:21/07/2022)

IBM.(n.d.b). *Product architecture*. (n.d.). Available at: https://www.ibm.com/docs/en/mapms/1_cloud?topic=features-product-architecture. (Retrieved at:21/07/2022)

IBM.(n.d.c). *IOT Solutions*. Available at: https://www.ibm.com/cloud/internet-of-things (Retrieved at:21/07/2022)

IBM.(2015). *IBM Watson IOT Platform*. Published on 2015, September 29. Available at: https://internetofthings.ibmcloud.com/. (Retrieved at:07/08/2022)

Ide, N., & Pustejovsky, J. (2010). *What Does Interoperability Mean, Anyway? Toward an Operational Definition of Interoperability for Language Technology*, 8. Available at: https://doi.org/https://www.cs.vassar.edu/~ide/papers/ICGL10.pdf. (Retrieved at:18/07/2022)

Industries, A. (n.d.). *DHT22 temperature-humidity sensor + extras*. adafruit industries blog RSS. Available at: https://www.adafruit.com/product/385 (Retrieved at:30/09/2022)

Intel. (2009). WHITE PAPER Intel® Embedded Processors The Embedded Internet *Rise of the embedded internet*. Available at: https://download.intel.com/newsroom/kits/embedded/pdfs/ECG_WhitePaper.pdf

Ismail, Y. (2019). *Internet of things (Iot) for automated and smart applications*.

K

Kindberg, T., & Fox, A. (2002). System software for ubiquitous computing. *IEEE Pervasive Computing*, *1*(1),pp.70–81. Available at: https://doi.org/10.1109/mprv.2002.993146. (Retrieved at:13/07/2022)

Konduru, V. R., & Bharamagoudra, M. R. (2017). Challenges and solutions of interoperability on IOT: How far have we come in resolving the IOT interoperability issues. *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*.

Available at: https://doi.org/10.1109/smarttechcon.2017.8358436. (Retrieved at:18/07/2022)

Kotha,D. H., & Mnssvkr Gupta, V. (2018). IOT application, a survey. *International Journal of Engineering & Technology*, *7*(2.7), 891. Available at: https://doi.org/10.14419/ijet.v7i2.7.11089. (Retrieved at:13/07/2022)

Krotov, V. (2017). The internet of things and New Business Opportunities. *Business Horizons*, *60*(6), 831–841. Available at: https://doi.org/10.1016/j.bushor.2017.07.009 (Online 24/06/2022)

Krumm, J. (2018). *Ubiquitous computing fundamentals*. Chapman and Hall/CRC.

L

Lueth, KL. (2015). Getting Started with the Internet of Things. Available at:

http://smiriengineering.com/Books/2015-March-Whitepaper-IoT-basics-Getting-started-with-the-Internet-of-Things.pdf. (Online 16/06/2022)

M

Madakam, S., Ramaswamy, R., & Tripathi, S. (2015). Internet of things (IOT): A literature review. *Journal of Computer and Communications*, *03*(05), 164–173. Available at: https://doi.org/10.4236/jcc.2015.35021 . (Retrieved at:7/06/2022)

Makhdoom, I., Abolhasan, M., Lipman, J., Liu, R. P., & Ni, W. (2019). Anatomy of threats to the internet of things. *IEEE Communications Surveys & Tutorials*, *21*(2),pp. 1636–1675. Available at: https://doi.org/10.1109/comst.2018.2874978. (Retrieved at:08/07/2022)

Microsoft. (2022). Concepts in Azure IoT Central | Microsoft Learn. Available at: https://learn.microsoft.com/en-us/azure/iot-central/core/concepts-architecture. (Retrieved at:28/09/2022)

Microsoft, Azure. (2022). *Azure iot – internet of things platform: Microsoft azure*. – Internet of Things Platform | Available at: https://azure.microsoft.com/en-us/solutions/iot/#overview (Retrieved at:6/08/2022)

Microsoft, Press. (2022). *AWS IoT Core*. Amazon. Available at: https://aws.amazon.com/iot-core/?c=i&sec=srv. (Retrieved at:07/08/2022)

Mohanty, S. P., Choppali, U., & Kougianos, E. (2016). Everything you wanted to know about smart cities: The internet of things is the backbone. *IEEE Consumer Electronics Magazine*, *5*(3), 60–70. Available at: https://doi.org/10.1109/mce.2016.2556879. (Retrieved at:20/06/2022)

Mouser,Electronics.(2022). *386 Adafruit: Mouser*. Available at:https://gr.mouser.com/ProductDetail/Adafruit/386?qs=GURawfaeGuDbeGFpZ2393w%3D%3D (Retrieved at:30/09/2022)

Mukherjee, S., Aarts, R. M., Ouwerkerk, M., Roovers, R., & Widdershoven, F. (2006). *AmIware hardware technology drivers of ambient intelligence*. Springer.

Mukherjee, S., Aarts, E., & Doyle, T. (2008). Special issue on Ambient Intelligence. *Information Systems Frontiers*, *11*(1), 1–5. Available at: https://doi.org/10.1007/s10796-008-9146-8. (Retrieved at:11/07/2022)

Mukhopadhyay, S. C. (2014). *Internet of things challenges and opportunities*. Springer International Publishing.

N

Naito, K. (2016). A survey on the internet-of-things: Standards, challenges and future prospects. *Journal of Information Processing*, *25*, 23–31. Available at: https://doi.org/10.2197/ipsjjip.25.23. (Retrieved at:20/07/2022)

Nakhuva, B., & Champaneria, T. (2015). Study of various internet of things platforms. *International Journal of Computer Science & Engineering Survey*, *6*(6), 61–74. Available at: https://doi.org/10.5121/ijcses.2015.6605. (Retrieved at:21/07/2022)

Németi, F., Pauletto, G., & Duay, D. (2017). *What AWS IoT can do - AWS IoT core*. Amazon. Available at: https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-solutions.html. (Retrieved at:15/07/2022)

Nižetić, S., Šolić, P., López-de-Ipiña González-de-Artaza, D., & Patrono, L. (2020). Internet of things (IOT): Opportunities, issues and challenges towards a smart and sustainable future. *Journal of Cleaner Production*, 274, 122877. Available at: https://doi.org/10.1016/j.jclepro.2020.122877 (Online 24/06/2022)

Noura, M., Atiquzzaman, M., & Gaedke, M. (2018). Interoperability in internet of things: Taxonomies and open challenges. *Mobile Networks and Applications*, *24*(3), 796–809. Available at: https://doi.org/10.1007/s11036-018-1089-9. (Retrieved at:19/07/2022)

O

Okta.(2022). *HMAC (hash-based message authentication codes) definition*. Available at: https://www.okta.com/identity-101/hmac/#:~:text=Hash%2Dbased%20message%20authentication%20code,use%20signatures%20and%20asymmetric%20cryptography. (Retrieved at:19/08/2022)

OpenRemote.(2015).*Documentation to build your own open source IOT platform*. OpenRemote. Published on 2022, May 5. Available at: https://www.openremote.io/developers/.(Retrieved at:16/08/2022)

OpenRemote. (2022a). *Get started with the free IOT platform*. OpenRemote. Publishde on 2022, May 10. Available at: https://openremote.io/get-started-iot-platform/ . (Retrieved at:16/08/2022)

OpenRemote.(2022b). *In the vein of JBoss*. Published on 2022, May 4. Available at: https://openremote.io/about/. (Retrieved at:19/08/2022)

OpenRemote.(2022c.). *Open-source enterprise IOT platform*. Published on 2022, May 4. Available at: https://openremote.io/product/. (Retrieved at:19/08/2022)

Oracle.(2022a). *Administering oracle internet of things cloud service*. (n.d.). Available at: https://docs.oracle.com/en/cloud/paas/iot-cloud/iotsu/administering-oracle-internet-things-cloud-service.pdf (Retrieved at:21/07/2022)

Oracle.(2022b). *Accelerate your operations with IOT*. Oracle. (n.d.). Available at: https://www.oracle.com/internet-of-things/#:~:text=Oracle%27s%20Fusion%20Cloud%20Internet%20of,safety%2C%20and%20connected%20customer%20experience. (Retrieved at:21/07/2022)

Oracle Help Center. (2020). *Learn about options for connecting devices to Oracle internet of things applications*. Available at: https://docs.oracle.com/en/solutions/internet-of-things-options-connect-devices/index.html#GUID-2C6B480E-812A-48DE-96F9-AF0F3C565487. (Retrieved at:29/09/2022)

Oracle, Help Center. (2022a). *Developing applications with Oracle Internet of Things Cloud Service*. Published on 2022, July 11). Available at: https://docs.oracle.com/en/cloud/paas/iot-cloud/iotgs/manage-devices.html (Retrieved at:21/07/2022)

Oracle, Help Center. (2022b). *Developing applications with Oracle Internet of Things Cloud Service*. Oracle Help Center. (Published on 2022, July 11). Available at: https://docs.oracle.com/en/cloud/paas/iot-cloud/iotgs/cloud-service-device-types.html (Retrieved at:21/07/2022)

P

Palem, K. V., Chakrapani, L. N. B., Kedem, Z. M., Lingamneni, A., & Muntimadugu, K. K. (2009). Sustaining Moore's law in embedded computing through probabilistic and approximate design. *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems - CASES '09*. Available at: https://doi.org/10.1145/1629395.1629397. (Retrieved at:13/07/2022)

Payne, R., & MacDonald, B. (2006). Ambient technology — now you see it, now you don't. *Computer Communications and Networks*, 199–217. Available at: https://doi.org/10.1007/978-1-84628-429-8_13. (Retrieved at:12/07/2022)

Psychoula, I., Singh, D., Chen, L., Chen, F., Holzinger, A., & Ning, H. (2018). Users' privacy concerns in IOT based applications. *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City*

Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). Available at: https://doi.org/10.1109/smartworld.2018.00317 (Retrieved at:22/06/2022)

Q

Qian, Y., Wu, D., Bao, W., & Lorenz, P. (2019). The internet of things for smart cities: Technologies and applications. *IEEE Network*, *33*(2), 4–5. Available at: https://doi.org/10.1109/mnet.2019.8675165 (Retrieved at: 20/06/2022)

R

Rabby, M. K., Islam, M. M., & Imon, S. M. (2019). A review of IOT application in a Smart Traffic Management System. *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*. Available at: https://doi.org/10.1109/icaee48663.2019.8975582. (Retrieved at:22/06/2022)

Rana, B., Singh, Y., & Singh, P. K. (2020). A systematic survey on internet of things: Energy efficiency and interoperability perspective. *Transactions on Emerging Telecommunications Technologies*, *32*(8). Available at: https://doi.org/10.1002/ett.4166. (Retrieved at:18/07/2022)

Rayes, A. M. M. A. R. (2022). *Internet of things from hype to reality: The road to digitization*. SPRINGER. Available at: *https://link.springer.com/content/pdf/10.1007/978-3-319-99516-8.pdf .* (Retrieved at:16/06/2022)

S

Samie, F., Bauer, L., & Henkel, J. (2016). IOT technologies for embedded computing. *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. Available at: https://doi.org/10.1145/2968456.2974004. (Retrieved at:12/07/2022)

Sendler, U. (2018). *The internet of things industrie 4.0 unleashed*. Springer Vieweg.

Sepranos, Dimitrios. (2019). *Internet-of-things Iot systems: Architectures, algorithms, methodologies*. SPRINGER.

Shadbolt, N. (2003). Ambient intelligence. Available at: http://echo.iat.sfu.ca/library/shadbolt_03_ambient_intelligence.pdf (Retrieved at:08/07/2022)

Sisinni, E., Saifullah, A., Han, S., Jennehag, U., & Gidlund, M. (2018). Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, *14*(11), 4724–4734. Available at: https://doi.org/10.1109/tii.2018.2852491 (Online 17/06/2022)

Stackowiak. (2019). *Azure internet of things revealed*. Apress. Stackowiak. (2019). *Azure internet of things revealed*. Apress.

Su, K., Li, J., & Fu, H. (2011). Smart city and the applications. *2011 International Conference on Electronics, Communications and Control (ICECC)*. Available at: https://doi.org/10.1109/icecc.2011.6066743. (Retrieved at:20/06/2022)

T

Thilakarathne, N. N., Kagita, M.K. & Madhuka Priyashan, W.D. (2012). *Green Internet of Things: The Next Generation Energy Efficient Internet of Things*. Available at: https://arxiv.org/ftp/arxiv/papers/2012/2012.01325.pdf. (Retrieved at:20/06/2022)

Thinger.io, (2020). DASHBOARDS - Thinger.io Documentation. *Create a Dashboard*. Available at: https://docs.thinger.io/features/dashboards. (Retrieved at:14/07/2022)

Thinger.io, (2021a). *Ota programming - thinger.io documentation*. Available at: https://docs.thinger.io/extended-features/ota (Retrieved at:14/07/2022)

Thinger.io, (2021b). *Web serial interface*. REMOTE CONSOLE - Thinger.io Documentation. (n.d.). Available at: https://docs.thinger.io/extended-features/remote-console. (Retrieved at:14/07/2022)

Thinger.io.(2021c). *Creating HTTP device profile*. HTTP DEVICES - Thinger.io Documentation. Available at: https://docs.thinger.io/http-devices (Retrieved at:30/08/2022)

Thinger.io.(2021d). *What is Thinger.io?* OVERVIEW - Thinger.io Documentation. (2021). Available at: https://docs.thinger.io/. (Retrieved at:28/09/2022)

Thinger.io. (2022a).History. *ABOUT - Thinger.io Documentation*. Available at: https://docs.thinger.io/about. (Retrieved at:14/07/2022)

Thinger.io. (2022b). *What is Thinger.io?* OVERVIEW - Thinger.io Documentation. Available at: https://docs.thinger.io/. (Retrieved at:14/07/2022)

Thinger.io.(2022c). *DEVICES*. Devices - thinger.io documentation. Available at: https://docs.thinger.io/arduino. (Retrieved at:14/07/2022)

Thinger.io.(2022d). *Introduction*. ESPRESSIF ESP32 - Thinger.io Documentation. Available at: https://docs.thinger.io/arduino/espressif-esp32 (Retrieved at:30/08/2022)

Thingsboard.(2022a). *Open-source IOT platform*. ThingsBoard. Available at: https://thingsboard.io/ (Retrieved at:15/07/2022)

Thingsboard.(2022b). *Pricing*. ThingsBoard. Available at: https://thingsboard.io/pricing/ (Retrieved at:15/07/2022)

Thingsboard.(2022c). *Our company*. ThingsBoard. Available at: https://thingsboard.io/company/ (Retrieved at:15/07/2022)

Thingsboard.(2022d). *Support ukraine*. ThingsBoard. Available at: https://thingsboard.io/support-ukraine/ (Retrieved at:15/07/2022)

Thingsboard.(2022e). *Support plans*. ThingsBoard. Available at: https://thingsboard.io/docs/services/support/ (Retrieved at:15/07/2022)

Thingsboard.(2022f). *What is thingsboard?* ThingsBoard. Available at: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/ (Retrieved at:15/07/2022)

Thingsboard.(2022g). *FAQ*. ThingsBoard. Available at: https://thingsboard.io/docs/faq/ . (Retrieved at:6/08/2022)

Thingsboard.(2022h). *Thingsboard architecture*. ThingsBoard. Available at: https://thingsboard.io/docs/reference/ (Retrieved at:29/09/2022)

TI.com. (2022a). *CC3200-LAUNCHXL*. CC3200-LAUNCHXL Evaluation board | Available at: https://www.ti.com/tool/CC3200-LAUNCHXL. (Retrieved at:25/08/2022)

Ti.com.(2022b). *CC3200 SimpleLink™ Wi-Fi® and Internet of Things Solution with MCU LaunchPad™ Hardware User's Guide*. Available at: https://www.ti.com/lit/ug/swru372c/swru372c.pdf (Retrieved at:25/08/2022)

Tripathy, B.K. & Anuradha. J. (2020). *CRC Press Taylor & Francis Group, Internet of Things (IoT) Technologies, Applications, Challenges and Solutions.* B.K. Tripathy & J. Anuradha

U

Uckelmann,D., Harrison.M., Michahelles.F., (2011). *Architecting the Internet of Things*. Available at: https://doi.org/10.1007/978-3-642-19157-2. (Online 16/06/2022)

V

Varjovi, A. E., & Babaie, S. (2020). Green internet of things (giot): Vision, applications and research challenges. *Sustainable Computing: Informatics and Systems*, *28*, 100448. Available at: https://doi.org/10.1016/j.suscom.2020.100448 . (Retrieved at:20/06/2022)

W

Weber, W., Rabaey, J., & Aarts, E. H. L. (2005). *Ambient intelligence*. Springer Berlin Heidelberg.

Wikipedia. (2022). *Wikimedia, Foundation. OpenRemote*. Available at: https://en.wikipedia.org/wiki/OpenRemote. (Retrieved at:19/08/2022)

Wortmann, F., & Flüchter, K. (2015). Internet of things. *Business & Information Systems Engineering*, 57(3), pp. 221–224. Available at: https://doi.org/10.1007/s12599-015-0383-3. (Online 16/06/2022)

Wolf, W. (2007). *High-performance embedded computing: Architectures, applications, and methodologies*. Morgan Kaufmann Pubs.

Z

Zhang, Y. (2011). Technology framework of the internet of things and its application. *2011, International Conference on Electrical and Control Engineering*. Available at: https://doi.org/10.1109/iceceng.2011.6057290 . (Retrieved at:7/06/2022)

Zhu, C., Leung, V. C., Shu, L., & Ngai, E. C.-H. (2015). Green internet of things for smart world. *IEEE Access*, *3*, 2151–2162. Available at: https://doi.org/10.1109/access.2015.249731 (Retrieved at: 20/06/2022)

Zyrianoff, I., Heideker, A., Silva, D., Kleinschmidt, J., Soininen, J.-P., Salmon Cinotti, T., & Kamienski, C. (2019). Architecting and deploying IOT Smart Applications: A performance–oriented approach. *Sensors*, *20*(1), 84. Available at: https://doi.org/10.3390/s20010084 (Retrieved at:22/06/2022)

APPENDIX A

```
/*********
 By Alexandros Panagiotakopoulos
AM:1716/ OLD AM:16108
UNIVERSITY OF IOANNINA - DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS
*********/
//Random(), source used :https://www.arduino.cc/reference/en/language/functions/random-numbers/random/
//Thinger.io(),source used: https://www.arduino.cc/reference/en/libraries/thinger.io/
//Based on a few older projects that i created, which can be found here: https://github.com/AlexandrosPanag/Texas_Instruments/blob/main/CC3200/ALL%20LEDS%20BLINK/CC3200_3_LEDS_BLINK.ino
//and here: https://github.com/AlexandrosPanag/Texas_Instruments/tree/main/CC3200/Button_Utilization

//in this project, the CC3200 BOARD will send random led states into the Thinger.io platform
//usually the random command takes one loop away to be properly initialized, hence why we require a guardian variable to
//properly protect our code. For a better explanation i encourage you to read my thesis "Internet of Things:Standards & Platforms
// NOTE: The code does take a while in order to be properly compiled so be patient

// This Project is an original work created by Alexandros Panagiotakopoulos during the academic year 2021-2022
// and is also part of the Thesis "IoT: Standards & Platforms".

//Include all the necessary libraries
#include <WiFi.h> //necessary to establish a Wi-Fi connection
#include <ThingerWifi.h> //necessary to establish a Thinger.io connection

#define USERNAME "AlexandrosPanag" //enter your Thinger.io username here
#define DEVICE_ID "CC3200" //enter your device id from your Thinger.io account here
```

```
#define DEVICE_CREDENTIAL "5oNbJhIgnB3bnM5!" //enter the given device credentials from the Thinger.io here

#define SSID "Redmi 7" //declare your router's username (SSID)
#define SSID_PASSWORD "mypassword" //declare your router's password (SSID_PASSWORD)

//Here, all 3 LED statuses will be declared. Each status will be sent to the Thinger.io website which can be remotely
//viewed from there. All statuses represent whether or not, our LED is turned on or off
int red_status=0; //the status which represents whether or not our RED LED is turned On or Off.
int yellow_status=0;//the status which represents whether or not our YELLOW LED is turned On or Off.
int green_status=0; //the status which represents whether or not our GREEN LED is turned On or Off.



int randRed = 0; //variable to store the random generated number for the red led
int randYellow = 0; //variable to store the random generated number for the yellow led
int randGreen = 0; //variable to store the random generated number for the green led



const int leftButton = PUSH1; //initialize the right button
const int rightButton = PUSH2;     // initialize the left button
int rightbuttonState = 0;         // variable for reading the pushbutton status
int leftbuttonState = 0;         // variable for reading the pushbutton status



int guardian=0; //the protector of our code, who can properly help the initialization of the random()
//command, and at the same time, help the Thinger.io connection properly to be established
```

```
ThingerWifi thing(USERNAME, DEVICE_ID, DEVICE_CREDENTIAL); //this command,
adds your Thinger.io credentials
//to establish a connection properly

void setup() {
 // set the boards led to output
 pinMode(RED_LED, OUTPUT);
 pinMode(GREEN_LED, OUTPUT);
 pinMode(YELLOW_LED, OUTPUT);
 pinMode(rightButton, INPUT_PULLUP); //declare the right button input
 pinMode(leftButton, INPUT_PULLUP); //declare the left button input
 // configure wifi network
 thing.add_wifi(SSID, SSID_PASSWORD);
 Serial.begin(9600); //initialize the Serial port to 9600 bauds/sec
 randomSeed(analogRead(0));// if analog input pin 0 is unconnected, random analog
 // noise will cause the call to randomSeed() to generate
 // different seed numbers each time the sketch runs.
 // randomSeed() will then shuffle the random function.

   //output into the Thinger.io the status of your LEDs, which can be viewed from there
   thing["LED STATES"] >> [](pson& out){
   out["GREEN_LED"] = green_status; //output the Green LED status (whether it is turned on
or off).
   out["YELLOW_LED"] = yellow_status; //output the Yellow LED status (whether it is turned
on or off).
   out["RED_LED"] = red_status; ////output the Red LED status (whether it is turned on or off).
   };
}


void loop() {
 //Generate a random number from 10 to 99, the code must run 3 times at least for the random()
 //command to be properly initialized
```

```
    const int randRed = random(10, 100); //choose a random number for the red LED from 10 to 99
    const int randYellow = random(10,100); //choose a random number for the yellow LED from 10 to 99
    const int randGreen = random(10, 100); //choose a random number for the green LED from 10 to 99

  thing.handle(); //handle the connection with the Thinger.io platform

  if(guardian>2){ //if the guardian is less than 3, dont output anything. The random command hasn't properly established yet
  Serial.println(); //print a message into the serial port
  Serial.println("************************************"); //print a message into the serial port
  // generate a random number from 10 to 19, then store them to each counter that represents an LED



  if(randRed%2==1){ //if the red random number can't be modulated
    Serial.println("RED WILL TURN ON"); //print a message into the serial port
    digitalWrite(RED_LED, HIGH);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for about 1 second
     red_status=1; //output into the Thinger.io website "1" under the RED_LED box, which represents that the LED is on.
  }
  else{ //if the red random number can be modulated
    Serial.println("RED WILL TURN OFF");
    digitalWrite(RED_LED, LOW);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for about 1 second
     red_status=0;//output into the Thinger.io website "0" under the RED_LED box, which represents that the LED is off.
  }
  if(randYellow%2==1){ //if the yellow random number can't be modulated
    Serial.println("YELLOW WILL TURN ON"); //print a message into the serial port
    digitalWrite(YELLOW_LED, HIGH);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
     yellow_status=1;//output into the Thinger.io website "1" under the YELLOW_LED box,
which represents that the LED is on.
  }
  else{
     Serial.println("YELLOW WILL TURN OFF"); ////if the yellow random number can be
modulated
    digitalWrite(YELLOW_LED, LOW);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
     yellow_status=0;//output into the Thinger.io website "0" under the YELLOW_LED box,
which represents that the LED is off
  }


  if(randGreen%2==1){ //if the green random number can't be modulated
    Serial.println("GREEN WILL TURN ON"); //print a message into the serial port
    digitalWrite(GREEN_LED, HIGH);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
    green_status=1;//output into the Thinger.io website "1" under the GREEN_LED box, which
represents that the LED is on.
  }
  else{ //if the green random number can be modulated
    Serial.println("GREEN WILL TURN OFF"); //print a message into the serial port
    digitalWrite(GREEN_LED, LOW);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
     green_status=0;//output into the Thinger.io website "0" under the RED_LED box, which
represents that the LED is off.
  }
  Serial.println("***********************************"); //print a message into the serial port
  delay(100); // wait for 0.1 second
  //OPTIONAL COMMANDS --to check whether or not the number is truly random, or not. Just uncomment the lines 139-144.
  //Serial.println("the random red value is:");
  //Serial.print(randRed);
  //Serial.println("the random yellow value is:");
  //Serial.print(randYellow);
  //Serial.prinln("the random green value is:");
  //Serial.print(randGreen);

  if(guardian==3){ //if the guardian reaches the third counter (third time the code has run) we want to send
    //the data into our Thinger.io API, and we also don't want our code to lose the connection.
    //However,if the user desires, he can press one of the built-in buttons and let the code compile all over again
    while(true){
      // read the state of the pushbutton value:
      rightbuttonState = digitalRead(rightButton); //declare the right button reading
      leftbuttonState = digitalRead(leftButton); //declare the left button reading
      thing.handle(); //handle the connection with the Thinger.io platform
       if (leftbuttonState == HIGH || rightbuttonState == HIGH) { //if the right or the left button is pressed
          guardian=0; //then reset the guardian (re-establish the random() command
          break; //and break the while(true) loop
          }
      }
    }
  }
  guardian++; //increase the counter each time the code has run, until it reaches the maximum run of 3 which will be trapped, on the while true() loop
  //unless the user decides to push one of the buttons
}
```

APPENDIX B

```
/*
*  By Alexandros Panagiotakopoulos
AM:1716/ OLD AM:16108
UNIVERSITY OF IOANNINA - DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS
*/



//Random(), source used :https://www.arduino.cc/reference/en/language/functions/random-numbers/random/
//Thinger.io(),source used: https://www.arduino.cc/reference/en/libraries/thinger.io/
//Based on a few older projects that i created, which can be found here: https://github.com/AlexandrosPanag/Texas_Instruments/blob/main/CC3200/ALL%20LEDS%20BLINK/CC3200_3_LEDS_BLINK.ino
//and here: https://github.com/AlexandrosPanag/Texas_Instruments/tree/main/CC3200/Button_Utilization



//in this project, the CC3200 BOARD will send random led states into the Thinger.io platform
//usually the random command takes one loop away to be properly initialized, hence why we require a guardian variable to
//properly protect our code. For a better explanation i encourage you to read my thesis "Internet of Things:Standards & Platforms
```

```
// NOTE: The code does take a while in order to compile

// This Project is an original work created by Alexandros Panagiotakopoulos during the academic year 2021-2022
// and is also part of the Thesis "IoT: Standards & Platforms".

#include <SPI.h> //include the SPI.h library
#include <Ethernet.h> //include the Ethernet.h library
#include <ThingerESP32.h> //include the ThingerESP32 library



#define BUTTON 34 // GIOP21 pin connected to button
#define USERNAME "AlexandrosPanag" //enter your Thinger.io username here
#define DEVICE_ID "ESP32" //enter your device id from your Thinger.io account here
#define DEVICE_CREDENTIAL "5oNbJhIgnB3bnM5!" //enter the given device credentials from the Thinger.io here



#define SSID "Redmi 7" //declare your router's username (SSID)
#define SSID_PASSWORD "mypassword" //declare your router's password (SSID_PASSWORD)



#define BLUE_LED 19 //declare GPIO 18 as Blue Led
#define RED_LED 18 // declare GPIO 19 as Red Led
#define GREEN_LED 2 // declare GPIO 2 as Green Led




int guardian=0; //the protector of our code, who can properly help the initialization of the random()
```

```
//command, and at the same time, help the Thinger.io connection properly to be established



//Here, all 3 LED statuses will be declared. Each status will be sent to the Thinger.io website which can be remotely
//viewed from there. All statuses represent whether or not, our LED is turned on or off
int BLUE_STATUS=0; //the status which represents whether or not our RED LED is turned On or Off.
int RED_STATUS=0;//the status which represents whether or not our YELLOW LED is turned On or Off.
int GREEN_STATUS=0; //the status which represents whether or not our GREEN LED is turned On or Off.

int randBLUE = 0; //variable to store the random generated number for the red led
int randRED = 0; //variable to store the random generated number for the yellow led
int randGREEN = 0; //variable to store the random generated number for the green led


int buttonState = 0;         // variable for reading the pushbutton status


// thinger.io config
ThingerESP32 thing(USERNAME, DEVICE_ID, DEVICE_CREDENTIAL); //generate a connection



// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(BLUE_LED, OUTPUT); // declare the Blue led as an output
  pinMode(RED_LED, OUTPUT); // declare the Red Led as an output
  pinMode(GREEN_LED, OUTPUT); // declare the Green Led as an output
  pinMode(BUTTON, INPUT_PULLUP); //declare the left button input
```

```
 // configure wifi network
 thing.add_wifi(SSID, SSID_PASSWORD);
 Serial.begin(9600); //initialize the Serial port to 9600 bauds/sec
 randomSeed(analogRead(0));// if analog input pin 0 is unconnected, random analog
 // noise will cause the call to randomSeed() to generate
 // different seed numbers each time the sketch runs.
 // randomSeed() will then shuffle the random function.



  //output into the Thinger.io the status of your LEDs, which can be viewed from there
   thing["LED STATES"] >> [](pson& out){
   out["BLUE_LED"] = BLUE_STATUS; //output the Green LED status (whether it is turned
on or off).
   out["RED_LED"] = RED_STATUS; //output the Yellow LED status (whether it is turned on
or off).
    out["GREEN_LED"] = GREEN_STATUS; ////output the Red LED status (whether it is
turned on or off).
   };
}



// the loop function runs over and over again forever
void loop() {
 //Generate a random number from 10 to 99, the code must run 3 times at least for the random()
 //command to be properly initialized
   const int randBLUE = random(10, 100); //choose a random number for the red LED from 10
to 99
   const int randRED = random(10,100); //choose a random number for the yellow LED from
10 to 99
   const int randGREEN = random(10, 100); //choose a random number for the green LED from
10 to 99
```

```
 thing.handle(); //handle the connection with the Thinger.io platform

  if(guardian>2){ //if the guardian is less than 3, dont output anything. The random command
hasn't properly established yet
  Serial.println(); //print a message into the serial port
  Serial.println("***********************************"); //print a message into the serial
port
  // generate a random number from 10 to 19, then store them to each counter that represents an
LED


 if(randBLUE%2==1){ //if the red random number can't be modulated
    Serial.println("BLUE WILL TURN ON"); //print a message into the serial port
    digitalWrite(BLUE_LED, HIGH);   // turn the BLUE LED on (HIGH is the voltage level)
    delay(100); //wait for about 1 second
   BLUE_STATUS=1; //output into the Thinger.io website "1" under the RED_LED box, which
represents that the LED is on.
  }
  else{ //if the red random number can be modulated
    Serial.println("BLUE WILL TURN OFF"); //print a message into the serial port
    digitalWrite(BLUE_LED, LOW);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for about 1 second
   BLUE_STATUS=0;//output into the Thinger.io website "0" under the RED_LED box, which
represents that the LED is off.
  }


   if(randRED%2==1){ //if the yellow random number can't be modulated
    Serial.println("RED WILL TURN ON");//print a message into the serial port
    digitalWrite(RED_LED, HIGH);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
    RED_STATUS=1;//output into the Thinger.io website "1" under the YELLOW_LED box,
which represents that the LED is on.
  }
  else{
    Serial.println("RED WILL TURN OFF"); ////if the yellow random number can be modulated
    digitalWrite(RED_LED, LOW);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
    RED_STATUS=0;//output into the Thinger.io website "0" under the YELLOW_LED box,
which represents that the LED is off
  }


  if(randGREEN%2==1){ //if the green random number can't be modulated
    Serial.println("GREEN WILL TURN ON"); //print a message into the serial port
    digitalWrite(GREEN_LED, HIGH);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
    GREEN_STATUS=1;//output into the Thinger.io website "1" under the GREEN_LED box,
which represents that the LED is on.
  }
  else{ //if the green random number can be modulated
    Serial.println("GREEN WILL TURN OFF"); //print a message into the serial port
    digitalWrite(GREEN_LED, LOW);   // turn the RED LED on (HIGH is the voltage level)
    delay(100); //wait for 0.1 second
     GREEN_STATUS=0;//output into the Thinger.io website "0" under the RED_LED box,
which represents that the LED is off.
  }

  Serial.println("**********************************");
  delay(100); // wait for 0.1 second
  //OPTIONAL COMMANDS --to check whether or not the number is truly random, or not.
Just uncomment the lines 139-144.
  //Serial.println("the random red value is:");
  //Serial.print(randRed);
  //Serial.println("the random yellow value is:");
  //Serial.print(randYellow);
  //Serial.prinln("the random green value is:");
  //Serial.print(randGreen);

  if(guardian==3){ //if the guardian reaches the third counter (third time the code has run) we
want to send
    //the data into our Thinger.io API, and we also don't want our code to lose the connection.
    //However,if the user desires, he can press one of the built-in buttons and let the code compile
all over again
    while(true){
      // read the state of the pushbutton value:
      buttonState = digitalRead(34); //declare the right button reading
      thing.handle(); //handle the connection with the Thinger.io platform
       if (buttonState == LOW) { //if the right or the left button is pressed
          guardian=0; //then reset the guardian (re-establish the random() command
          break; //and break the while(true) loop
          }
    }
  }
  }
  guardian++; //increase the counter each time the code has run, until it reaches the maximum
run of 3 which will be trapped, on the while true() loop
  //unless the user decides to push one of the buttons
}
```

APPENDIX C

```
/*
*  By Alexandros Panagiotakopoulos
AM:1716/ OLD AM:16108
UNIVERSITY OF IOANNINA - DEPARTMENT OF INFORMATICS &
TELECOMMUNICATIONS
*/
```

```
// This work is based upon the pioneering work provided by Microsoft
// on their project on the Azure_IoT_PnP Template, which is also part of their ESPRESSIF32
Azure IoT Kit
// and Azure SDK For C library packages
//
// This Project is an original work created by Alexandros Panagiotakopoulos during the
academic year 2021-2022
// and is also part of the Thesis "IoT: Standards & Platforms".

// SOURCES USED:
// AZURE SDK LIBRARY FOR C, https://www.arduino.cc/reference/en/libraries/azure-sdk-
for-c/
// DHT11/22 Library Built-In examples, https://github.com/adafruit/DHT-sensor-library

//include all the necessary library files in order to function as intended
#include <stdlib.h> //include the stdlib.h library
#include <stdarg.h> //include the stdarg.h library
#include "DHT.h" //include the DHT.h library
#include <az_core.h> //include the az_core library
#include <az_iot.h> //include the az_iot.h library

#include "AzureIoT.h" //include the Azure IoT.h library
#include "Azure_IoT_PnP_Template.h" //include the Azure_IoT_PnP_Template header file

#include <az_precondition_internal.h> //include the az_precondition_internal header file


//Global Declaration - Definition variables
#define AZURE_PNP_MODEL_ID "dtmi:azureiot:devkit:freertos:Esp32AzureIotKit;1"

//Useful information about the microcontroller which will be displayed into the Azure IoT
Central
```

```
#define DEVICE_INFORMATION_NAME                    "ESP32 DEVELOPMENT KIT" //declare information about the board's model
#define MANUFACTURER_PROPERTY_NAME                  "ESPRESSIF" //declare the manfacturer's name
#define MODEL_PROPERTY_NAME                 "Development Kit" //declare the the board's model
#define MANUFACTURER_PROPERTY_VALUE                 "ESPRESSIF" //declare the information about the manufacturer
#define MODEL_PROPERTY_VALUE                 "ESP32 Development Kit" //declare the model property


//delcare the sensor's values which will be sent into the Azure IoT Central
#define TELEMETRY_TEMPERATURE                    "temperature" //declare the sensor's temperature
#define TELEMETRY_HUMIDITY                  "humidity" //declare the sensor's humidity
#define WRITABLE_PROPERTY_TELEMETRY_FREQ_SECS              "telemetryFrequencySecs" //declare the telemetry frquency which will be sent into the Azure IoT Central.
#define WRITABLE_PROPERTY_RESPONSE_SUCCESS              "success" //declare the success value if the telemetry succeeds the value transmission
#define DOUBLE_DECIMAL_PLACE_DIGITS 2 //place digits to display the information which will be transmitted

//--------------------------Initializing the DHT11 sensor----------------------------------------
#define DHTPIN 4     // Declare the DHTPIN on PIN 4
#define DHTTYPE DHT11   // Declare the DHT model that is used -- in this case the DHT11

DHT dht(DHTPIN, DHTTYPE); //declare the DHT11 sensor depending on the model

//----------------------------------------------------------------------------------------------

// --- Function Checks and Returns in case everything worked as intened -----------------------
#define RESULT_OK       0
```

```
#define RESULT_ERROR    __LINE__

#define EXIT_IF_TRUE(condition, retcode, message, ...)                     \
  do                                                     \
  {                                                     \
    if (condition)                                          \
    {                                                   \
      LogError(message, ##__VA_ARGS__ );                              \
      return retcode;                                     \
    }                                                   \
  } while (0)

#define EXIT_IF_AZ_FAILED(azresult, retcode, message, ...)                    \
  EXIT_IF_TRUE(az_result_failed(azresult), retcode, message, ##__VA_ARGS__ )

// --- end of error checking

#define DATA_BUFFER_SIZE 1024 // --- declare the data buffer size



static uint8_t data_buffer[DATA_BUFFER_SIZE]; // --- declare the data buffer size
static uint32_t telemetry_send_count = 0; //telemtry sending counter
static size_t telemetry_frequency_in_seconds = 10; // With default frequency of once in 10
seconds.
static time_t last_telemetry_send_time = INDEFINITE_TIME; //find the last telemetry time
that something was sent



//--- Function telemetries that are available into transmitting the JSON messages
static int generate_telemetry_payload( //functions in order to generate the telemetry
    uint8_t*  payload_buffer,  size_t  payload_buffer_size,  size_t*  payload_buffer_length);
//functions to initialize the buffer
```

```
static int generate_device_info_payload( //functions to generate information payload
  az_iot_hub_client const* hub_client, uint8_t* payload_buffer,
  size_t payload_buffer_size, size_t* payload_buffer_length);

static int consume_properties_and_generate_response( //function to generate response through
the Azure IoT Central server
  azure_iot_t* azure_iot, az_span properties,
  uint8_t* buffer, size_t buffer_size, size_t* response_length);



// --- function to generate the pnp payload -------
void azure_pnp_init()
{
}

//function the generate the pnp model information
const az_span azure_pnp_get_model_id()
{
  return AZ_SPAN_FROM_STR(AZURE_PNP_MODEL_ID);
}

//function that can calculate the Azure IoT Central telemetry in seconds
void azure_pnp_set_telemetry_frequency(size_t frequency_in_seconds)
{
  telemetry_frequency_in_seconds = frequency_in_seconds;
      LogInfo("Telemetry frequency set to once every %d seconds.",
telemetry_frequency_in_seconds);
}


// Code to generate telemetry and transmit it throught the Azure IoT Central
int azure_pnp_send_telemetry(azure_iot_t* azure_iot)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);
  time_t now = time(NULL); //get the current system time
  if (now == INDEFINITE_TIME) //check if the current time can be received
  {
      LogError("Failed getting current time for controlling telemetry."); //if not, print the appropriate time message
    return RESULT_ERROR; //return the error
  }
  else if (last_telemetry_send_time == INDEFINITE_TIME || //else, get the right time
          difftime(now, last_telemetry_send_time) >= telemetry_frequency_in_seconds)
  {
    size_t payload_size; //declare the payload size as a global type variable (size_t)

     last_telemetry_send_time = now; //get the latest telemetry and store it into the variable last_telemtry

     if (generate_telemetry_payload(data_buffer, DATA_BUFFER_SIZE, &payload_size) != RESULT_OK) //check if the buffer telemetry payload can be generated
    {
        LogError("Failed generating telemetry payload."); //if the telemetry is unable to be generated
      return RESULT_ERROR; //then return an error code
    }

    if (azure_iot_send_telemetry(azure_iot, az_span_create(data_buffer, payload_size)) != 0) //if the azure iot telemetry can be generated
    {
      LogError("Failed sending telemetry."); //if the telemetry is unable to be generated
      return RESULT_ERROR; //then return an error code
    }
  }

  return RESULT_OK; //if the everything succeeded without any issues return ok
}
```

```
//Code in order to transmit Azure IoT Central Information
int azure_pnp_send_device_info(azure_iot_t* azure_iot, uint32_t request_id)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);

  int result; //declare an integer result value
  size_t length; //declare a global (size_t) length

    result = generate_device_info_payload(&azure_iot->iot_hub_client, data_buffer,
DATA_BUFFER_SIZE, &length); //check if the telemetry payload can be generated
  EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed generating telemetry
payload."); //if not, print the appropriate message

  result = azure_iot_send_properties_update(azure_iot, request_id, az_span_create(data_buffer,
length)); //check if the data buffer (reported properties) can be sent
   EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed sending reported
properties update.");  //if not, print the appropriate message

  return RESULT_OK; //if everything procceeded without any errors then return an "ok"
message
}

int azure_pnp_handle_command_request(azure_iot_t* azure_iot, command_request_t
command)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);

  uint16_t response_code;

    LogError("Command not recognized (%.*s).", az_span_size(command.command_name),
az_span_ptr(command.command_name));
```

```
  return azure_iot_send_command_response(azure_iot, command.request_id, response_code, AZ_SPAN_EMPTY);
}

int azure_pnp_handle_properties_update(azure_iot_t* azure_iot, az_span properties, uint32_t request_id) //declare necessary pnp properties and update them
{
  _az_PRECONDITION_NOT_NULL(azure_iot);
  _az_PRECONDITION_VALID_SPAN(properties, 1, false);

  int result;
  size_t length;

   result = consume_properties_and_generate_response(azure_iot, properties, data_buffer, DATA_BUFFER_SIZE, &length); //check if a response can be generated
  EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed generating properties ack payload."); //check if any issues occured and print the appropriate message

  result = azure_iot_send_properties_update(azure_iot, request_id, az_span_create(data_buffer, length)); //check if a response can be generated
   EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed sending reported properties update."); //check if any issues occured and print the appropriate message

  return RESULT_OK;
}

// Code to get the DHT11 Sensor's temperature
float get_temperature()
{
   float t = dht.readTemperature(); //declare a float t variable which will store the DHT temperature
  if (isnan(t)) {
    return -30;
  }
  return t; //send the t variable into the Azure IoT Central when the function is called
}

// Code to get the DHT11 Sensor's humidity
float get_humidity() //declare a float h variable which will store the DHT humidity (relative
humidity)
{
  float h = dht.readHumidity(); //declare a float h variable which will store the DHT humidity
  if (isnan(h)) {
    return -30;
  }
  return h; ///send the h variable into the Azure IoT Central when the function is called
}

//Code in order to generate the telemetry payload
static int generate_telemetry_payload(uint8_t* payload_buffer, size_t payload_buffer_size,
size_t* payload_buffer_length)
{
  az_json_writer jw; //declare the JSON writter object
  az_result rc; //declare the JSON result
    az_span payload_buffer_span = az_span_create(payload_buffer, payload_buffer_size);
//declare a JSON buffer that includes the payload buffer and the size
  az_span json_span; //declare the json span
  float temperature, humidity; //declare the DHT11 temperature & humidity


  // Acquiring the temperature& humidity from the DHT11 sensor
  dht.begin();
  temperature = get_temperature(); //call the function to get the temperature
  humidity = get_humidity(); //call the function to get the humidity

  rc = az_json_writer_init(&jw, payload_buffer_span, NULL);
   EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed initializing json writer for
telemetry.");
```

```
  rc = az_json_writer_append_begin_object(&jw);
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed setting telemetry json root.");

//--------CODE TO TRANSMIT THE TEMPERATURE TO THE AZURE IOT CENTAL-----------------------------------------------------
  rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(TELEMETRY_TEMPERATURE)); //declare the temperature through the use of telemetry
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding temperature property name to telemetry payload."); //if the transmittion fails print a failure message
  rc = az_json_writer_append_double(&jw, temperature, DOUBLE_DECIMAL_PLACE_DIGITS); //place the appropriate digits into the temperature
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding temperature property value to telemetry payload. "); //if the transmittion fails print a failure message

//--------CODE TO TRANSMIT THE HUMIDITY TO THE AZURE IOT CENTAL-----------------------------------------------------
  rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(TELEMETRY_HUMIDITY)); //declare the humidity through the use of telemetry
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding himidity property name to telemetry payload."); //if the transmittion fails print a failure message
  rc = az_json_writer_append_double(&jw, humidity, DOUBLE_DECIMAL_PLACE_DIGITS); //place the appropriate digits into the temperature
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding humidity property value to telemetry payload. "); //if the transmittion fails print a failure message

  payload_buffer_span = az_json_writer_get_bytes_used_in_destination(&jw); //write the json messages into the buffer

  if ((payload_buffer_size - az_span_size(payload_buffer_span)) < 1) //check if there is an efficient buffer size to generate the telemetry
  {
    LogError("Insufficient space for telemetry payload null terminator."); //print an error message in case the telemetry is inefficient
    return RESULT_ERROR; //return the error message
  }

  payload_buffer[az_span_size(payload_buffer_span)] = null_terminator; //give the payload buffer null terminator value
  *payload_buffer_length = az_span_size(payload_buffer_span); //store the payload buffer span into the payload buffer value if there werent any issues

  return RESULT_OK; //return an ok result
}

static int generate_device_info_payload(az_iot_hub_client const* hub_client, uint8_t* payload_buffer, size_t payload_buffer_size, size_t* payload_buffer_length)
{
  az_json_writer jw; //declare the JSON writter object
  az_result rc; //declare the JSON result
    az_span payload_buffer_span = az_span_create(payload_buffer, payload_buffer_size); //declare the json span
  az_span json_span; //call the function to get the humidity

  rc = az_json_writer_init(&jw, payload_buffer_span, NULL); //check if the payload buffer size is available
    EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed initializing json writer for telemetry.");  //if not, print the appropriate message

  rc = az_json_writer_append_begin_object(&jw); //check if the root object writing is available
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed setting telemetry json root.");  //if not, print the appropriate message

  rc = az_iot_hub_client_properties_writer_begin_component( //check if Azure IoT Hub client properties writter is available
    hub_client, &jw, AZ_SPAN_FROM_STR(DEVICE_INFORMATION_NAME)); //if it is available then print the device information name into the platform
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed writting component name.");  //if not, print the appropriate message

  //add the manufacturer name to the JSON payload and check if there are any issues
                    rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(MANUFACTURER_PROPERTY_NAME)); //check if the manufacturer's property name can be transmitted
          EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MANUFACTURER_PROPERTY_NAME to payload."); //if an error occured return the appropriate error message
                        rc = az_json_writer_append_string(&jw, AZ_SPAN_FROM_STR(MANUFACTURER_PROPERTY_VALUE)); //check if the manufacturer's property name can be transmitted
          EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MANUFACTURER_PROPERTY_VALUE to payload. "); //if an error occured return the appropriate error message

    //add the property name to the JSON payload and check if there are any issues
                    rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(MODEL_PROPERTY_NAME)); //check if the model's property name can be transmitted
          EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MODEL_PROPERTY_NAME to payload."); //if an error occured return the appropriate error message
                        rc = az_json_writer_append_string(&jw, AZ_SPAN_FROM_STR(MODEL_PROPERTY_VALUE));//check if the model's property name can be transmitted
          EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MODEL_PROPERTY_VALUE to payload. "); //if an error occured return the appropriate error message
  rc = az_iot_hub_client_properties_writer_end_component(hub_client, &jw);  //check the hub
is available to write components and settings
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed closing component object.");  //if an
error occured return the appropriate error message

  rc = az_json_writer_append_end_object(&jw); //check if the JSON messages are available to
write
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed closing telemetry json payload."); //if
an error occured return the appropriate error message

  payload_buffer_span = az_json_writer_get_bytes_used_in_destination(&jw); //write JSON
byte messages and store them in the payload buffer

  if ((payload_buffer_size - az_span_size(payload_buffer_span)) < 1) //check if the payload size
is not enough
  {
    LogError("Insufficient space for telemetry payload null terminator.");  //if an error occured
return the appropriate error message
    return RESULT_ERROR; //return the reason as to why this error occured
  }

  payload_buffer[az_span_size(payload_buffer_span)] = null_terminator; //give the payload
buffer null terminator value
  *payload_buffer_length = az_span_size(payload_buffer_span); //store the payload buffer span
into the payload buffer value if there werent any issues

  return RESULT_OK; //if everything proceeded without any issues then procceed
}

static int generate_properties_update_response( //in the following lines check if the JSON
writer is initialized and update a response
  azure_iot_t* azure_iot,
  az_span component_name, int32_t frequency, int32_t version,
  uint8_t* buffer, size_t buffer_size, size_t* response_length)
```

```
{
  az_result azrc;
  az_json_writer jw;
  az_span response = az_span_create(buffer, buffer_size);

  azrc = az_json_writer_init(&jw, response, NULL); //check if the JSON writter is writtable or not
    EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed initializing json writer for properties update response."); //if not, then print the appropriate message

   azrc = az_json_writer_append_begin_object(&jw); //check if the JSON in properties is available
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed opening json in properties update response."); //if not, then print the appropriate message

  // This Azure PnP Template does not have a named component,
  // so az_iot_hub_client_properties_writer_begin_component is not needed.

   azrc = az_iot_hub_client_properties_writer_begin_response_status( //check if the azure iot hub is writtable and sends a response
    &azure_iot->iot_hub_client,
    &jw,
    AZ_SPAN_FROM_STR(WRITABLE_PROPERTY_TELEMETRY_FREQ_SECS),
    (int32_t)AZ_IOT_STATUS_OK,
    version,
     AZ_SPAN_FROM_STR(WRITABLE_PROPERTY_RESPONSE_SUCCESS)); //check if the IoT hub has responded
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed appending status to properties update response."); //return the appropriate failure message

  azrc = az_json_writer_append_int32(&jw, frequency); //declare the frequency and store it into the AZRC variable
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed appending frequency value to properties update response."); //if there were any errors reading the frequency then return the appropriate message

    azrc = az_iot_hub_client_properties_writer_end_response_status(&azure_iot->iot_hub_client, &jw); //initialize the azure JSON response value
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed closing status section in properties update response."); //check if any issues occured and print the appropriate message


  azrc = az_json_writer_append_end_object(&jw); //initialize the azure JSON writer append end object
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed closing json in properties update response."); //check if any issues occured and print the appropriate message

   *response_length = az_span_size(az_json_writer_get_bytes_used_in_destination(&jw)); //write the JSON span size

  return RESULT_OK; //if everything procceeded without any errors then return ok
}

static int consume_properties_and_generate_response(
  azure_iot_t* azure_iot, az_span properties,
  uint8_t* buffer, size_t buffer_size, size_t* response_length)
{
  int result;
  az_json_reader jr;
  az_span component_name;
  int32_t version = 0;

  az_result azrc = az_json_reader_init(&jr, properties, NULL);
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed initializing json reader for properties update."); //check if any issues occured and print the appropriate message
  const az_iot_hub_client_properties_message_type message_type =
    AZ_IOT_HUB_CLIENT_PROPERTIES_MESSAGE_TYPE_WRITABLE_UPDATED;

  azrc = az_iot_hub_client_properties_get_properties_version(
    &azure_iot->iot_hub_client, &jr, message_type, &version);
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed writable properties version.");
//check if any issues occured and print the appropriate message

  azrc = az_json_reader_init(&jr, properties, NULL);
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed re-initializing json reader for
properties update."); //check if any issues occured and print the appropriate message

  while (az_result_succeeded(
    azrc = az_iot_hub_client_properties_get_next_component_property(
      &azure_iot->iot_hub_client, &jr, message_type,
      AZ_IOT_HUB_CLIENT_PROPERTY_WRITABLE, &component_name)))
  {
                                                   if                        (az_json_token_is_text_equal(&jr.token,
AZ_SPAN_FROM_STR(WRITABLE_PROPERTY_TELEMETRY_FREQ_SECS)))
    {
      int32_t value;
        azrc = az_json_reader_next_token(&jr); //check if the JSON reader next token can be
written
       EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed getting writable properties next
token.");  //if not, print the appropriate message

      azrc = az_json_token_get_int32(&jr.token, &value); //check if the JSON reader next token
can be written
          EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed getting writable properties
int32_t value.");  //if not, print the appropriate message

          azure_pnp_set_telemetry_frequency((size_t)value);  //apply the appropriate telemetry
frequency

```

```
      result = generate_properties_update_response(
        azure_iot, component_name, value, version, buffer, buffer_size, response_length);
                              EXIT_IF_TRUE(result       !=       RESULT_OK,       RESULT_ERROR,
"generate_properties_update_response failed.");  //if not, print the appropriate message
    }
    else
    {
       LogError("Unexpected property received (%.*s).", //check if any unexpected properties
were received
        az_span_size(jr.token.slice), az_span_ptr(jr.token.slice));
    }

    azrc = az_json_reader_next_token(&jr); //check if the JSON next token is available
     EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed moving to next json token of
writable properties.");  //if not, print the appropriate message

      azrc = az_json_reader_skip_children(&jr); //check if the JSON reader skip children is
available
      EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed skipping children of writable
properties.");  //if not, print the appropriate message

    azrc = az_json_reader_next_token(&jr); //check if JSON reader next token can be written
     EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed moving to next json token of
writable properties (again)."); //if not, print the appropriate message
  }

  return RESULT_OK; //if everything procceeded without any errors return the appropriate
message
}
```

APPENDIX D

```
/*
*  By Alexandros Panagiotakopoulos
AM:1716/ OLD AM:16108
UNIVERSITY OF IOANNINA - DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS
*/

// This work is based upon the pioneering work provided by Microsoft
// on their project on the Azure_IoT_PnP Template, which is also part of their ESPRESSIF32 Azure IoT Kit
// and Azure SDK For C library packages
//
// This Project is an original work created by Alexandros Panagiotakopoulos during the academic year 2021-2022
// and is also part of the Thesis "IoT: Standards & Platforms".
// SOURCES USED:
// AZURE SDK LIBRARY FOR C, https://www.arduino.cc/reference/en/libraries/azure-sdk-
for-c/
// DHT11/22 Library Built-In examples, https://github.com/adafruit/DHT-sensor-library


//include all the necessary library files in order to function as intended
#include <stdlib.h> //include the stdlib.h library
#include <stdarg.h> //include the stdarg.h library
#include "DHT.h" //include the DHT.h library
#include <az_core.h> //include the az_core library
#include <az_iot.h> //include the az_iot.h library

#include "AzureIoT.h" //include the Azure IoT.h library
#include "Azure_IoT_PnP_Template.h" //include the Azure_IoT_PnP_Template header file

#include <az_precondition_internal.h> //include the az_precondition_internal header file



//Global Declaration - Definition variables
#define AZURE_PNP_MODEL_ID "dtmi:azureiot:devkit:freertos:Esp32AzureIotKit;1"

//Useful information about the microcontroller which will be displayed into the Azure IoT
Central
#define DEVICE_INFORMATION_NAME                "ESP32 DEVELOPMENT KIT"
//declare information about the board's model
#define MANUFACTURER_PROPERTY_NAME             "ESPRESSIF" //declare the
manfacturer's name
#define MODEL_PROPERTY_NAME              "Development Kit" //declare the the board's
model
#define MANUFACTURER_PROPERTY_VALUE            "ESPRESSIF" //declare the
information about the manufacturer
```

```
#define MODEL_PROPERTY_VALUE                "ESP32 Development Kit" //declare the model property

//delcare the sensor's values which will be sent into the Azure IoT Central
#define TELEMETRY_TEMPERATURE                   "temperature" //declare the sensor's temperature
#define TELEMETRY_HUMIDITY                "humidity" //declare the sensor's humidity
#define
WRITABLE_PROPERTY_TELEMETRY_FREQ_SECS              "telemetryFrequencySecs"
//declare the telemetry frquency which will be sent into the Azure IoT Central.
#define WRITABLE_PROPERTY_RESPONSE_SUCCESS             "success" //declare the success value if the telemetry succeeds the value transmission
#define DOUBLE_DECIMAL_PLACE_DIGITS 2 //place digits to display the information which will be transmitted

//--------------------------Initializing the DHT11 sensor----------------------------------------
#define DHTPIN 4     // Declare the DHTPIN on PIN 4
#define DHTTYPE DHT22   // Declare the DHT model that is used -- in this case the DHT11

DHT dht(DHTPIN, DHTTYPE); //declare the DHT11 sensor depending on the model



//-----------------------------------------------------------------------------------------------

// --- Function Checks and Returns in case everything worked as intened -----------------------
#define RESULT_OK       0
#define RESULT_ERROR    __LINE__

#define EXIT_IF_TRUE(condition, retcode, message, ...)                         \
  do                                                                        \
  {                                                                        \
    if (condition)                                                            \
    {                                                                        \
      LogError(message, ##__VA_ARGS__ );                                     \
      return retcode;                                                        \
    }                                                                        \
  } while (0)

#define EXIT_IF_AZ_FAILED(azresult, retcode, message, ...)                   \
  EXIT_IF_TRUE(az_result_failed(azresult), retcode, message, ##__VA_ARGS__ )

// --- end of error checking

#define DATA_BUFFER_SIZE 1024 // --- declare the data buffer size



static uint8_t data_buffer[DATA_BUFFER_SIZE]; // --- declare the data buffer size
static uint32_t telemetry_send_count = 0; //telemtry sending counter
static size_t telemetry_frequency_in_seconds = 10; // With default frequency of once in 10
seconds.
static time_t last_telemetry_send_time = INDEFINITE_TIME; //find the last telemetry time
that something was sent



//--- Function telemetries that are available into transmitting the JSON messages
static int generate_telemetry_payload( //functions in order to generate the telemetry
    uint8_t*  payload_buffer,  size_t  payload_buffer_size,  size_t*  payload_buffer_length);
//functions to initialize the buffer
static int generate_device_info_payload( //functions to generate information payloard
  az_iot_hub_client const* hub_client, uint8_t* payload_buffer,
  size_t payload_buffer_size, size_t* payload_buffer_length);

static int consume_properties_and_generate_response( //function to generate response through
the Azure IoT Central server
```

```
  azure_iot_t* azure_iot, az_span properties,
  uint8_t* buffer, size_t buffer_size, size_t* response_length);



// --- function to generate the pnp payload -------
void azure_pnp_init()
{
}

//function the generate the pnp model information
const az_span azure_pnp_get_model_id()
{
  return AZ_SPAN_FROM_STR(AZURE_PNP_MODEL_ID);
}

//function that can calculate the Azure IoT Central telemetry in seconds
void azure_pnp_set_telemetry_frequency(size_t frequency_in_seconds)
{
  telemetry_frequency_in_seconds = frequency_in_seconds;
      LogInfo("Telemetry frequency set to once every %d seconds.",
telemetry_frequency_in_seconds);
}


// Code to generate telemetry and transmit it throught the Azure IoT Central
int azure_pnp_send_telemetry(azure_iot_t* azure_iot)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);
  time_t now = time(NULL); //get the current system time
  if (now == INDEFINITE_TIME) //check if the current time can be received
  {
     LogError("Failed getting current time for controlling telemetry."); //if not, print the
appropriate time message
```

```
    return RESULT_ERROR; //return the error
  }
  else if (last_telemetry_send_time == INDEFINITE_TIME || //else, get the right time
           difftime(now, last_telemetry_send_time) >= telemetry_frequency_in_seconds)
  {
    size_t payload_size; //declare the payload size as a global type variable (size_t)

     last_telemetry_send_time = now; //get the latest telemetry and store it into the variable
last_telemtry

     if (generate_telemetry_payload(data_buffer, DATA_BUFFER_SIZE, &payload_size) !=
RESULT_OK) //check if the buffer telemetry payload can be generated
    {
        LogError("Failed generating telemetry payload."); //if the telemetry is unable to be
generated
      return RESULT_ERROR; //then return an error code
    }

    if (azure_iot_send_telemetry(azure_iot, az_span_create(data_buffer, payload_size)) != 0) //if
the azure iot telemetry can be generated
    {
      LogError("Failed sending telemetry."); //if the telemetry is unable to be generated
      return RESULT_ERROR; //then return an error code
    }
  }

  return RESULT_OK; //if the everything succeeded without any issues return ok
}

//Code in order to transmit Azure IoT Central Information
int azure_pnp_send_device_info(azure_iot_t* azure_iot, uint32_t request_id)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);
  int result; //declare an integer result value
  size_t length; //declare a global (size_t) length

     result = generate_device_info_payload(&azure_iot->iot_hub_client, data_buffer,
DATA_BUFFER_SIZE, &length); //check if the telemetry payload can be generated
  EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed generating telemetry
payload."); //if not, print the appropriate message

  result = azure_iot_send_properties_update(azure_iot, request_id, az_span_create(data_buffer,
length)); //check if the data buffer (reported properties) can be sent
   EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed sending reported
properties update.");  //if not, print the appropriate message

  return RESULT_OK; //if everything procceeded without any errors then return an "ok"
message
}

int azure_pnp_handle_command_request(azure_iot_t* azure_iot, command_request_t
command)
{
  _az_PRECONDITION_NOT_NULL(azure_iot);

  uint16_t response_code;

    LogError("Command not recognized (%.*s).", az_span_size(command.command_name),
az_span_ptr(command.command_name));

  return azure_iot_send_command_response(azure_iot, command.request_id, response_code,
AZ_SPAN_EMPTY);
}

int azure_pnp_handle_properties_update(azure_iot_t* azure_iot, az_span properties, uint32_t
request_id) //declare necessary pnp properties and update them
{
  _az_PRECONDITION_NOT_NULL(azure_iot);
  _az_PRECONDITION_VALID_SPAN(properties, 1, false);

  int result;
  size_t length;

   result = consume_properties_and_generate_response(azure_iot, properties, data_buffer, DATA_BUFFER_SIZE, &length); //check if a response can be generated
  EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed generating properties ack payload."); //check if any issues occured and print the appropriate message

  result = azure_iot_send_properties_update(azure_iot, request_id, az_span_create(data_buffer, length)); //check if a response can be generated
   EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "Failed sending reported properties update."); //check if any issues occured and print the appropriate message

  return RESULT_OK;
}

// Code to get the DHT11 Sensor's temperature
float get_temperature()
{
  float t = dht.readTemperature();
  if (isnan(t)) {
    return -30;
  }
  return t; //send the t variable into the Azure IoT Central when the function is called
}

// Code to get the DHT11 Sensor's humidity
float get_humidity() //declare a float h variable which will store the DHT humidity (relative humidity)
{
  float h = dht.readHumidity();
  if (isnan(h)) {
    return -30;
  }
  return h; //send the t variable into the Azure IoT Central when the function is called
}

//Code in order to generate the telemetry payload
static int generate_telemetry_payload(uint8_t* payload_buffer, size_t payload_buffer_size,
size_t* payload_buffer_length)
{
  az_json_writer jw; //declare the JSON writter object
  az_result rc; //declare the JSON result
    az_span payload_buffer_span = az_span_create(payload_buffer, payload_buffer_size);
//declare a JSON buffer that includes the payload buffer and the size
  az_span json_span; //declare the json span
  float temperature, humidity; //declare the DHT11 temperature & humidity


  // Acquiring the temperature& humidity from the DHT11 sensor
  dht.begin();
  temperature = get_temperature(); //call the function to get the temperature
  humidity = get_humidity(); //call the function to get the humidity

  rc = az_json_writer_init(&jw, payload_buffer_span, NULL);
   EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed initializing json writer for
telemetry.");

  rc = az_json_writer_append_begin_object(&jw);
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed setting telemetry json root.");

//--------CODE TO TRANSMIT THE TEMPERATURE TO THE AZURE IOT CENTAL-----
-----------------------------------------------
```

```
  rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(TELEMETRY_TEMPERATURE)); //declare the temperature through the use of telemetry
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding temperature property name to telemetry payload."); //if the transmittion fails print a failure message
  rc = az_json_writer_append_double(&jw, temperature, DOUBLE_DECIMAL_PLACE_DIGITS); //place the appropriate digits into the temperature
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding temperature property value to telemetry payload. "); //if the transmittion fails print a failure message

//--------CODE TO TRANSMIT THE HUMIDITY TO THE AZURE IOT CENTAL-------------------------------------------------------
  rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(TELEMETRY_HUMIDITY)); //declare the humidity through the use of telemetry
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding himidity property name to telemetry payload."); //if the transmittion fails print a failure message
  rc = az_json_writer_append_double(&jw, humidity, DOUBLE_DECIMAL_PLACE_DIGITS); //place the appropriate digits into the temperature
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding humidity property value to telemetry payload. "); //if the transmittion fails print a failure message

  payload_buffer_span = az_json_writer_get_bytes_used_in_destination(&jw); //write the json messages into the buffer

  if ((payload_buffer_size - az_span_size(payload_buffer_span)) < 1) //check if there is an efficient buffer size to generate the telemetry
  {
    LogError("Insufficient space for telemetry payload null terminator."); //print an error message in case the telemetry is inefficient
    return RESULT_ERROR; //return the error message
  }
  payload_buffer[az_span_size(payload_buffer_span)] = null_terminator; //give the payload buffer null terminator value
  *payload_buffer_length = az_span_size(payload_buffer_span); //store the payload buffer span into the payload buffer value if there werent any issues

  return RESULT_OK; //return an ok result
}

static int generate_device_info_payload(az_iot_hub_client const* hub_client, uint8_t* payload_buffer, size_t payload_buffer_size, size_t* payload_buffer_length)
{
  az_json_writer jw; //declare the JSON writter object
  az_result rc; //declare the JSON result
    az_span payload_buffer_span = az_span_create(payload_buffer, payload_buffer_size); //declare the json span
  az_span json_span; //call the function to get the humidity

  rc = az_json_writer_init(&jw, payload_buffer_span, NULL); //check if the payload buffer size is available
    EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed initializing json writer for telemetry.");  //if not, print the appropriate message

  rc = az_json_writer_append_begin_object(&jw); //check if the root object writing is available
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed setting telemetry json root.");  //if not, print the appropriate message

  rc = az_iot_hub_client_properties_writer_begin_component( //check if Azure IoT Hub client properties writter is available
    hub_client, &jw, AZ_SPAN_FROM_STR(DEVICE_INFORMATION_NAME)); //if it is available then print the device information name into the platform
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed writting component name.");  //if not, print the appropriate message

  //add the manufacturer name to the JSON payload and check if there are any issues
```

```
        rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(MANUFACTURER_PROPERTY_NAME));  //check if the manufacturer's property name can be transmitted
        EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MANUFACTURER_PROPERTY_NAME to payload."); //if an error occured return the appropriate error message
        rc = az_json_writer_append_string(&jw, AZ_SPAN_FROM_STR(MANUFACTURER_PROPERTY_VALUE));  //check if the manufacturer's property name can be transmitted
        EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MANUFACTURER_PROPERTY_VALUE to payload. "); //if an error occured return the appropriate error message

    //add the property name to the JSON payload and check if there are any issues
        rc = az_json_writer_append_property_name(&jw, AZ_SPAN_FROM_STR(MODEL_PROPERTY_NAME)); //check if the model's property name can be transmitted
        EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MODEL_PROPERTY_NAME to payload."); //if an error occured return the appropriate error message
        rc = az_json_writer_append_string(&jw, AZ_SPAN_FROM_STR(MODEL_PROPERTY_VALUE));//check if the model's property name can be transmitted
        EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed adding MODEL_PROPERTY_VALUE to payload. "); //if an error occured return the appropriate error message


  rc = az_iot_hub_client_properties_writer_end_component(hub_client, &jw);  //check the hub is available to write components and settings
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed closing component object.");  //if an error occured return the appropriate error message
  rc = az_json_writer_append_end_object(&jw); //check if the JSON messages are available to write
  EXIT_IF_AZ_FAILED(rc, RESULT_ERROR, "Failed closing telemetry json payload."); //if an error occured return the appropriate error message

   payload_buffer_span = az_json_writer_get_bytes_used_in_destination(&jw); //write JSON byte messages and store them in the payload buffer

  if ((payload_buffer_size - az_span_size(payload_buffer_span)) < 1) //check if the payload size is not enough
  {
    LogError("Insufficient space for telemetry payload null terminator."); //if an error occured return the appropriate error message
    return RESULT_ERROR; //return the reason as to why this error occured
  }

   payload_buffer[az_span_size(payload_buffer_span)] = null_terminator; //give the payload buffer null terminator value
  *payload_buffer_length = az_span_size(payload_buffer_span); //store the payload buffer span into the payload buffer value if there werent any issues

  return RESULT_OK; //if everything proceeded without any issues then procceed
}

static int generate_properties_update_response( //in the following lines check if the JSON writer is initialized and update a response
  azure_iot_t* azure_iot,
  az_span component_name, int32_t frequency, int32_t version,
  uint8_t* buffer, size_t buffer_size, size_t* response_length)
{
  az_result azrc;
  az_json_writer jw;
  az_span response = az_span_create(buffer, buffer_size);
  azrc = az_json_writer_init(&jw, response, NULL); //check if the JSON writter is writtable or not
    EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed initializing json writer for properties update response."); //if not, then print the appropriate message

   azrc = az_json_writer_append_begin_object(&jw); //check if the JSON in properties is available
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed opening json in properties update response."); //if not, then print the appropriate message

  // This Azure PnP Template does not have a named component,
  // so az_iot_hub_client_properties_writer_begin_component is not needed.

  azrc = az_iot_hub_client_properties_writer_begin_response_status( //check if the azure iot hub is writtable and sends a response
    &azure_iot->iot_hub_client,
    &jw,
    AZ_SPAN_FROM_STR(WRITABLE_PROPERTY_TELEMETRY_FREQ_SECS),
    (int32_t)AZ_IOT_STATUS_OK,
    version,
     AZ_SPAN_FROM_STR(WRITABLE_PROPERTY_RESPONSE_SUCCESS)); //check if the IoT hub has responded
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed appending status to properties update response."); //return the appropriate failure message

  azrc = az_json_writer_append_int32(&jw, frequency); //declare the frequency and store it into the AZRC variable
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed appending frequency value to properties update response."); //if there were any errors reading the frequency then return the appropriate message

        azrc = az_iot_hub_client_properties_writer_end_response_status(&azure_iot->iot_hub_client, &jw); //initialize the azure JSON response value
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed closing status section in properties update response."); //check if any issues occured and print the appropriate message

  azrc = az_json_writer_append_end_object(&jw); //initialize the azure JSON writer append end object
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed closing json in properties update response."); //check if any issues occured and print the appropriate message

    *response_length = az_span_size(az_json_writer_get_bytes_used_in_destination(&jw)); //write the JSON span size

  return RESULT_OK; //if everything procceeded without any errors then return ok
}

static int consume_properties_and_generate_response(
  azure_iot_t* azure_iot, az_span properties,
  uint8_t* buffer, size_t buffer_size, size_t* response_length)
{
  int result;
  az_json_reader jr;
  az_span component_name;
  int32_t version = 0;

  az_result azrc = az_json_reader_init(&jr, properties, NULL);
   EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed initializing json reader for properties update."); //check if any issues occured and print the appropriate message

  const az_iot_hub_client_properties_message_type message_type =
    AZ_IOT_HUB_CLIENT_PROPERTIES_MESSAGE_TYPE_WRITABLE_UPDATED;

  azrc = az_iot_hub_client_properties_get_properties_version(
    &azure_iot->iot_hub_client, &jr, message_type, &version);
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed writable properties version."); //check if any issues occured and print the appropriate message

  azrc = az_json_reader_init(&jr, properties, NULL);
  EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed re-initializing json reader for properties update."); //check if any issues occured and print the appropriate message

  while (az_result_succeeded(
    azrc = az_iot_hub_client_properties_get_next_component_property(
      &azure_iot->iot_hub_client, &jr, message_type,
      AZ_IOT_HUB_CLIENT_PROPERTY_WRITABLE, &component_name)))
  {
    if (az_json_token_is_text_equal(&jr.token, AZ_SPAN_FROM_STR(WRITABLE_PROPERTY_TELEMETRY_FREQ_SECS)))
    {
      int32_t value;
      azrc = az_json_reader_next_token(&jr); //check if the JSON reader next token can be written
      EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed getting writable properties next token.");  //if not, print the appropriate message

      azrc = az_json_token_get_int32(&jr.token, &value); //check if the JSON reader next token can be written
      EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed getting writable properties int32_t value.");  //if not, print the appropriate message

      azure_pnp_set_telemetry_frequency((size_t)value);  //apply the appropriate telemetry frequency

      result = generate_properties_update_response(
        azure_iot, component_name, value, version, buffer, buffer_size, response_length);
      EXIT_IF_TRUE(result != RESULT_OK, RESULT_ERROR, "generate_properties_update_response failed.");  //if not, print the appropriate message
    }
    else
    {
       LogError("Unexpected property received (%.*s).", //check if any unexpected properties were received
        az_span_size(jr.token.slice), az_span_ptr(jr.token.slice));
    }

    azrc = az_json_reader_next_token(&jr); //check if the JSON next token is available
     EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed moving to next json token of writable properties.");  //if not, print the appropriate message

     azrc = az_json_reader_skip_children(&jr); //check if the JSON reader skip children is available
     EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed skipping children of writable properties.");  //if not, print the appropriate message

    azrc = az_json_reader_next_token(&jr); //check if JSON reader next token can be written
     EXIT_IF_AZ_FAILED(azrc, RESULT_ERROR, "Failed moving to next json token of writable properties (again)."); //if not, print the appropriate message
  }

  return RESULT_OK; //if everything procceeded without any errors return the appropriate message
}
```

APPENDIX E

```
/*
*  By Alexandros Panagiotakopoulos
AM:1716/ OLD AM:16108
UNIVERSITY OF IOANNINA - DEPARTMENT OF INFORMATICS &
TELECOMMUNICATIONS
*/

//Thinger.io(),source used: https://www.arduino.cc/reference/en/libraries/thinger.io/

// This Project is an original work created by Alexandros Panagiotakopoulos during the
academic year 2021-2022
// and is also part of the Thesis "IoT: Standards & Platforms".

#include <SPI.h> //include the SPI.h official library
#include <Ethernet.h> //include the Etherner.h library
#include <ESP8266WiFi.h> //include the ESP8266 library
#include <ThingerESP8266.h> //include the neccessary libraries
#include "DHT.h" //include the adafruit DHT sensor library
```

```
// dht config
#define DHTPIN 2 //declare the pin that the sensor is connected to
#define DHTTYPE DHT11 //declare the DHT type
#define USERNAME "AlexandrosPanag" //define your username's password from thinger.io
#define DEVICE_ID "ESP8266" //define your device's id from thinger.io
#define DEVICE_CREDENTIAL "HL+1&FCQ@Kq4G-lH" //define your randomly
generated credentials from thinger.io

#define SSID "Redmi 7" //define your router's id
#define SSID_PASSWORD "mypassword" //define your router's password

DHT dht(DHTPIN, DHTTYPE); //initialize the DHT11 sensor

// thinger.io config
ThingerESP8266 thing(USERNAME, DEVICE_ID, DEVICE_CREDENTIAL); //generate a
connection

void setup() {
  Serial.begin(9600); //initialize the Serial baud rate to 9600 bauds/sec
  dht.begin(); //begin the temperature & humidity measurements
  thing.add_wifi(SSID, SSID_PASSWORD); //add the wi-fi password
  thing["DHT11"] >> [](pson& out){ //output into the Thinger.io API the sensor values
     out["HUMIDITY"] = dht.readHumidity(); //output into the Thinger.io API the Humidity
(Humidity)
    out["CELSIUS"] = dht.readTemperature(); //output into the Thinger.io the Celsius (Celsius)
       //out["Fahrenheit"] = dht.readTemperature(true); OPTIONAL CODE to output the
Fahrenheit
  };
}

void loop() {
  int humidity=dht.readHumidity(); // read the sensor's humidity
  int celcius=dht.readTemperature(); //read the sensor's temperature (celsius)
  //int fahrenheit=dht.readTemperature(true); OPTIONAL CODE to output the temperature into 
Fahrenheit


  Serial.println("Humidity:  "); //print a message into the Serial Port
  Serial.print(humidity); //output the Humidity into the Serial Port
  Serial.println(" RH"); //print a message into the Serial Port

  Serial.println("Celsius:  "); //print a message into the Serial Port
  Serial.print(celcius); //output the Celsius into the Serial Port
  Serial.println("  °C"); //print a message into the Serial Port

  //Serial.println("Fahrenheit: ");
  //Serial.print(fahrenheit);
  //Serial.println(" °F");

  thing.handle(); //Handle the connection with the Thinger.io platform
  delay(1000); //delay for about 1 second before taking another measurement
}
```