Master Thesis on the Academic Master of Information and Communication Electronic System

Design and Implementation of a Complete Internet of Things Infrastructure: Insightful Lightweight Platform with Real-Time Devices Automation and Alerts, Gateway and Device Firmware, and Communication Protocol.

Author
Vladislav Rykov

Director
Dr.Agustin Carlos Caminero Herraez

Master Thesis presented at

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, CONTROL, TELEMÁTICA Y QUÍMICA APLICADA A LA INGENIERÍA ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

as part of the mandatory requirements to fulfill the Grade of Academic Master of Information and Communication Electronic System 2020

Research line in Electronic Technology and Industrial Equipment in ICES, Communications, Mobile and Secure Systems in ICES

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, CONTROL, TELEMÁTICA Y QUÍMICA APLICADA A LA INGENIERÍA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Master Thesis

Design and Implementation of a Complete Internet of Things Infrastructure: Insightful Lightweight Platform with Real-Time Devices Automation and Alerts, Gateway and Device Firmware, and Communication Protocol.

Author:

Vladislav Rykov

Director:

Dr.Agustin Carlos Caminero Herraez

Evaluation Committee:

Chair:

Secretary:

Member:

Presentation Date: October 6, 2020

Grade:

UNESCO Code/s:

☑ The author authorizes the internal publication of this Master Thesis (inside the Master community) pdf document for the use of the Master students.

☑ The author marking this space declare that is the sole responsible of this Master Thesis as well as is the only responsible of the originality of this work and has followed all the references and bibliography searching granting other works as necessary.

Master Thesis presented at

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, CONTROL, TELEMÁTICA Y QUÍMICA APLICADA A LA INGENIERÍA
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

as part of the mandatory requirements to fulfill the Grade of

Academic Master of Information and Communication Electronic System

2020

Research line in Electronic Technology and Industrial Equipment in ICES, Communications, Mobile and Secure Systems in ICES

Sworn Declaration of authorship of scientific work to defend the Master's Thesis

Date: 17/09/2020.

Who subscribe:

Author: Vladislav Rykov

ID (Spanish): Y3170552Q

Notes that is the author of the work.

Design and Implementation of a Complete Internet of Things Infrastructure: Insightful Lightweight Platform with Real-Time Devices Automation and Alerts, Gateway and Device Firmware, and Communication Protocol.

In this regard, I express the originality of the conceptualization of work, interpretation of data and compilation of findings, leaving established that those intellectual contributions of others, have been properly referenced in the text of the work.

DECLARATION:

☑ I guarantee that the work we refer is an original document has not been published, in whole or in part, in another journal, and I did not do plagiarism or self-plagiarism in anyway.

☑ I certify that I have contributed directly to the intellectual content of this manuscript, the genesis and analysis of their data, so I'm able to take public responsibility for it.

☑ I have not committed scientific fraud, plagiarism or vices of authorship; otherwise, I will accept the penalties appropriate disciplinary measures.

☑ I remark I asked for and they are granted the needed authorizations for data inclusion or materials overt third parties have rights.

Signature.

Vladislav Rykov

Resumen

Este documento contiene el trabajo fin de máster en el cual fue propuesta y desarrollada una infraestructura íntegra de Internet de las Cosas (IdC). La infraestructura consiste de una plataforma computacionalmente eficiente que puede ser fácilmente ejecutada en una Raspberry Pi; gateway y un protocolo de comunicación. El proyecto fue desarrollado durante la actividad de investigación en la Universitat Jaume I para los propósitos relacionados con IdC y Redes de Sensores Inalámbricos (RSI).

El desarrollo de RSI y IdC continuamente abre nuevas oportunidades en diferentes niveles. Gracias a estas tecnologías emergentes se hacen posibles aumentos de márgenes lucrativos de empresas, control de variables del medioambiente, aplicaciones de domótica, así como la introducción de ciudades inteligentes aplicando las técnicas de la Inteligencia Artificial (IA) sobre los datos extraídos.

La infraestructura desarrollada constituye una alternativa a las arquitecturas existentes de IdC para aplicaciones de propósito general y específico. El objetivo es ofrecer un entorno de IdC gratuito, de fácil acceso y listo para usar para los propósitos educativos, de PYMEs y empresas de tamaño mediano y hobby. A menudo es complicado encontrar una solución eficiente y de código libre que se adapte en práctica a cualquier aplicación. El proyecto actual ofrece tal solución, así como una visión profunda en sus mecanismos internos.

El proyecto proporciona un entorno de ejecución para un conjunto amplio de aplicaciones de IdC. Por ejemplo, las aplicaciones de la agricultura de precisión pueden ser desplegadas para monitorizar las condiciones de huerta como temperatura y humedad de aire, humedad del suelo y control de riego. Aplicaciones de monitorización del medioambiente pueden ser efectuadas recolectando datos de nivel de pH en ríos o datos de la calidad del aire. La automatización de riego en función de la humedad del suelo de una huerta, las alertas por correo electrónico en caso si la calidad del agua es muy baja son las posibles funcionalidades que proporciona la infraestructura.

Los principios de una gestión eficiente de energía en los dispositivos de IdC fueron tomados como la base para el desarrollo del proyecto. Para la implementación de la solución actual las actividades de diseño de la infraestructura, desarrollo e implementación del protocolo de comunicación y gateway y el proceso de validación, es decir, un ciclo completo de desarrollo de un producto, fueron llevadas a cabo.

Palabras Claves

Internet de las Cosas (IdC), Redes de Senores Inhalámbricos (RSI), Arquitectura de IdC, Protocolo de Comunicación de IdC, Sistemas Embebidos, Firmware

Abstract

This document contains a final master thesis project in which a complete Internet of Things (IoT) infrastructure is proposed and developed. The infrastructure consists of a lightweight platform, easily capable to run on a Raspberry Pi board; gateway and communication protocol. The project was developed during the research activity at University Jaume I for the purposes of the research group related to the IoT and Wireless Sensor Networks (WSN).

The development of WSN and the IoT is continuously opening new opportunities on many levels. It broadens corporational profit margins, controls the environmental variables, brings smart commodities into homes, and makes smart cities more intelligent applying Artificial Intelligence (AI) over extracted data.

The developed infrastructure constitutes an alternative for existent IoT architectures for general-purpose and custom applications. It aims to provide free easily-accessible 'out-of-box' ready IoT environment which can be used for Small Medium Enterprise (SME), educational, and hobby purposes. It is often too difficult to find an efficient open-source IoT solution that can fit generally any IoT application. This project provides such solution along with deep insights into internal mechanisms.

The project provides a running environment for a rich suit of IoT applications. For instance, precision agriculture applications can be deployed for field conditions monitoring as air temperature and humidity, soil moisture, and watering control; environmental monitoring applications can be deployed gathering pH data of rivers or air quality data. Watering automation as a function of field soil moisture state, and email alerts when water quality is too low are possible features provided by the infrastructure.

The principles of efficient device power management are employed as the basis for the development of the project. For the development of this solution a complete cycle of product development activities such as IoT infrastructure design, communication protocol design and implementation, gateway firmware programming, and validation tests was carried out.

Keywords

Internet of Things (IoT), Wireless Sensor Networks (WSN), IoT Architecture, IoT Communication Protocol, Embedded Systems, Firmware

Contents

List of Figures

List of Tables

Acronyms

ADC Analog-to-Digital Converter. 49

AI Artificial Intelligence. 7, 114

API Application Programming Interface. 33, 36, 52, 74

AWS Amazon Web Services. 39

CAN Control Area Network. 24

CH Cluster Head. 30

CRC Cyclic Redundancy Check. 81, 136

CRM Customer relationship management. 33

CSV Comma-Separated Values. 58, 112

DAG Directed Acyclic Graph. 31

DAO Data Access Object. 59, 69, 91, 108

DCCP Datagram congestion control protocol. 111

EXEHDA Execution Environment for Highly Distributed Applications. 36

FDI Field Device Integration. 34

FI-PPP Future Internet Public Private Partnership. 32

FOTA Firmware-Over-The-Air. 81, 136

FSA Frequency Spectrum Allocation. 54

GPIO General-Purpose Input/Output. 49

GPRS General Packet Radio Service. 51, 111

I^2C Inter-Integrated Circuit. 48

IC Integrated Circuit. 53

ICT Information and Communication Technologies. 23

IIoT Industrial Internet of Things. 34

IIRA Industrial Internet Reference Architecture. 34

IL Immutable Ledger. 30

IoT Internet of Things. 7, 23–26, 29, 30, 41, 51, 83, 126

IP Internet Protocol. 56

ISM Industrial, Scientific and Medical. 54

JGC Java Garbage Collector. 37

JSON JavaScript Object Notation. 57, 58, 111

JVM Java Virtual Machine. 37, 42

LoT Lab of Things. 38

LSB Lightweight Scalable Blockchain. 30, 31, 41

MCU Microcontroller. 50, 53, 60, 68, 123

MIT Massachusetts Institute of Technology. 23

ML Machine Learning. 114

MSFC Message Sequence Flow Chart. 75

NB-IoT Narrow-Band IoT. 51

OPC-UA Ole for Process Control - Unified Architecture. 34

OS Operating System. 84

OSI Open System Interconnection. 26

PCB Printed Circuit Board. 54

PWM Pulse-Width Modulation. 49

RAM Random Access Memory. 49

RAMI Reference Architecture Model Industry. 34

RF Radio Frequency. 51, 54

ROM Read Only Memory. 81

RPi Raspberry Pi. 25, 39, 42, 59, 68

RTC Real-Time Clock. 49

RTSJ Real-Time Specification for Java. 37, 42

SME Small Medium Enterprise. 7, 32, 41, 135

SoC System-on-Chip. 25, 59, 68

SPI Serial Peripheral Interface. 48, 53

SQL Structured Query Language. 57

SRD Short Range Devices. 54

SSH Secure Shell. 61

SSL Secure Sockets Layer. 35, 110

TCP Transmission Control Protocol. 56, 74

TLS Transport Layer Security. 35, 110

TPS Transactions Per Second. 32

TTN The Things Network. 51

UART Universal Asynchronous Receiver/Transmitter. 48, 53

UDP User Datagram Protocol. 56, 70, 74

UNED Universidad Nacional de Educacón a Distancia. 38

URI Universal Resource Identifier. 104

UTC Coordinated Universal Time. 76

WOT Web of Things. 13, 39

WSGI Web Server Gateway Interface. 62

WSN Wireless Sensor Networks. 7, 40

Chapter 1

Introduction

Contents

1.1 Context and Motivation of the Project

Last decade Information and Communication Technologies (ICT) induced significant changes in all areas of human activity as economy, education, science, and, most of all, business. One of the innovative trends which has been powerfully developed last years is the IoT.

The term Internet of Things was first publicly used in 2009 by Kevin Ashton, professor at Massachusetts Institute of Technology (MIT), giving a public conference, and later, it was formally announced in RFID Journal [6]. Though, Ashton commented that it was used in the research circles since 1999. The concept was rapidly picked by the scientists, journalists, engineers, and even politicians, and became part of research articles and European Union conferences titles.

The IoT represents the next Internet evolution which will require much more capacity from the present communication technologies in order to put together and analyze data from billions of 'things'. Despite presence of technological constraints, the extracted advantages opened new horizons and perspectives for many applications. In a culmination, the data generated by 'things' are processed to obtain knowledge which in turn becomes wisdom.

There were about 6.3 billion of people on our planet in 2003 while 500 million of connected to the Internet devices [32]. On that moment, it was 0.08 device connected to the Internet per person. The world experienced an explosive growth of smartphones and tablets in 2010 what

increased the number of devices connected to the Internet to about 12.5 billion. It was the point in history when the number of devices connected to the Internet was greater than the population of the Earth (1.84 devices per person).

According to [55], 127 new IoT devices are being connected to the Internet every second, 26.6 billion of active IoT devices maintain stable connections with their platforms, 90% of senior managers claim that the IoT is vital for their businesses, and the predicted IoT market size will be about 520 billion of dollars in 2021.

Currently, the IoT comprises a set of distributed networks which serve different purposes. As vehicles have different control systems installed which are interconnected normally through Control Area Network (CAN) [43] bus, similarly, power plants, apartments, and other facilities have control systems as heating, air conditioning, and lightening which can be connected to and controlled from the Internet. Conforming the IoT industry is growing, more infrastructures are connected to the Internet.

There are many application areas of IoT. The biggest industries and fields are mentioned below.

- Aerospace and Aviation

  Airplane wireless monitoring is achieved deploying a network of intelligent devices. On one hand, the control systems are enriched with the data from the devices, and, on the other hand, data are sent to the internet platform where they are stored and processed.

- Automotive

  Modern vehicles are full of control devices that monitor from wheel pressure to foreign vehicles proximity. On the other hand, the vehicle manufacturing and logistics are also accelerated using intelligent warehouse features and tracking.

- Smart Buildings

  Home and building automation and smart monitoring were under intensive development last years. Communication technologies as ZigBee [33] and 6LoWPAN [85] allow real-time bidirectional communication for automation and control purposes.

- Health

  There exists a plenty of IoT applications in the health industry as patients physical state monitoring for preventive and premature diagnosis of diverse diseases.

- Smart Cities

  Smart parking, city lighting, and traffic management are few examples which improve city power efficiency and congestion control and alleviate pollution level.

- Logistics and supply chain

  IoT tracking applications are massively applied in the logistics field to optimize business processes. This way, clients can also receive precise product delivery status.

- Product life cycle maintenance

  From the manufacturing until the end of the product life cycle, a product can be monitored and its data analyzed. Many vehicles, fridges, coffee machines are already being monitored.

- Petroleum and gas industries

  Infrastructures for extraction and transmission of petroleum and gas are connected to the Internet for facilities condition as well as material leakage monitoring.

- Environmental monitoring

  The quality of air, precipitations, river or lake water quality monitoring helps to care about flora and fauna especially in national parks and protected areas.

- Insurance

  Innovative approaches appear in this industry as well. Assured items, e.g. vehicles, are connected to the Internet for driver behaviour monitoring and the next policy cost estimation.

The IoT applications just mentioned and any other require a working infrastructure. The infrastructure covers a lot of aspects as networking, communication protocols, data models, database technologies, and servers. It is a layered tree-like structure where devices represent leafs and the IoT platform does the root.

There are various IoT architectures and each one is better suited for certain application area. Therefore, depending on the application needs, the user will opt for one or another architecture, though most of them focus on the data recollection.

The present project proposes a simple yet powerful and stable IoT architecture and describes its development on each layer. The main disadvantage of almost all architectures is their complexity and computational wage they put on the server. The efforts were done to make the proposed architecture lightweight and easily installable on a Raspberry Pi (RPi) [96] or any other Linux-based System-on-Chip (SoC).

1.2 Project Objectives

This section covers the project objectives reached during the project development time and future objectives to cover during revisions.

1.2.1 General Objectives

During the period of the project development the implementation of a complete IoT infrastructure was expected. The objectives were split into general and specific ones. Each general objective represents a workpackage that was covered by various specific objectives which resemble more closely the actual workflow. Moreover, in practice, each specific objective was divided into subtasks which completion marked the development more precisely.

The general objectives of the project are presented as follows:

1. Design and implementation of a multi-layer IoT architecture.

2. Design and development of a database schema using the most suitable database technologies.

3. Enable the deployment of applications with real-time devices automation and alerts.

1.2.2 Specific Objectives

In order to reach the general objectives, the specific objectives must be reached first. They are related to, on one side, the very development process and, on the other, to the application of concepts learned in the subjects presented in the Master Program. Detailed specific objectives description is presented in the following listing.

As was mentioned earlier, the specific objectives are related to the general objectives, and, thus, they are presented according to the existent relations.

Specific objectives that cover the general objective 1.

1. Design of the IoT infrastructure architecture.

2. Design and development of a lightweight IoT platform.

3. Design and development of gateway firmware.

4. Design and development of a template firmware for devices.

5. Design and development of a communication protocol on the application level (Open System Interconnection (OSI) L7) for device-gateway communication.

6. Unit testing of the developed components.

Specific objectives that cover the general objective 2.

1. Selection of the database technology for the project.

2. Creation of the database schema.

Specific objectives that cover the general objective 3.

1. Design and development of a communication protocol on device level for platform-device interaction.

2. Integration of an email service to the platform.

3. Adaption of event driven features of the selected database technology.

1.3 Project Structure

The first chapter is dedicated to the project introduction. Project context and motivations, objectives to achieve, brief explanations, and development workplace were presented to the reader.

The second chapter discusses the existent state of the art of IoT architectures. It examines various infrastructures highlighting their advantages and disadvantages as well as application areas.

The third chapter describes, first, the infrastructure architecture, next, the components which comprise it, furthermore, the technologies, frameworks, and software that were indispensable in order to finish the development.

The fourth chapter brings details into working methodology and initial project planning. Materials and project costs estimation are described along with the project monitoring and its evolution.

The fifth chapter exposes the communication protocol structure, design, purposes, and implementation. It also critically discusses it and gives ideas about its possible future improvements.

The sixth chapter gives the details of the gateway firmware design, limitations and, possible improvements.

The seventh chapter specifies the implementation details of the IoT platform. It brings to light a description of each step of development and how every specific objective was reached.

The eighth chapter reveals the verification and validation tests of the architecture's functionality.

To conclude, the ninth chapter makes a conclusive meditation and discusses possible future work related to the project.

Additional information that was considered to be superfluous during the document writing was included into the Appendices section.

Chapter 2

State of The Art

Contents

The IoT emerged in 1999 and evolved in a strong and standalone industry quite recently, about 10 years ago. Though, this area can be considered as young it was quickly developing during this period. Now it conquers and revolutionizes every business.

During these years many IoT architectures were conceived being inspired by various factors and situations. A great work has been done to get to the current state of the art.

This chapter will discuss various IoT architectures. Some of them have not gained popularity yet, and some have done.

2.1 Blockchain-inspired Architectures

Blockchain [53] is a rapidly expanding over its original purposes technology. It securely maintains continuously increasing list of immutable records, called blocks, that contain a timestamp, a link to the previous record, and encrypted data about the previous records.

It is not a centralized architecture, but rather distributed peer-to-peer one. When a new record appears in the system, it must be validated by the nodes based on the previous data.

Blockchain brings security and privacy to the IoT projects. When the both merged, secure data storage is ensured.

There were conducted many researches on the topic of merging the IoT and blockchain. As a result, the IoT architecture is transformed. This section considers two most popular IoT architectures based on merging the IoT and blockchain.

2.1.1 LSB: A Lightweight Scalable Blockchain

Authors of [30] address issues as computational costs, limited scalability, bandwidth overheads, and delays of the traditional blockchain technology and propose a new model, suitable for IoT architectures.

The main framework for the architecture consists of two tiers: local private application network and overlay. Basic communication for exchanging data between entities are defined as transaction. In a local application devices use a local private Immutable Ledger (IL) for local transactions. It intends to optimize resources consumption. IL is similar to blockchain but is managed centrally. For local transactions security a symmetric encryption is used.

The overlay tier consists of computationally capable nodes similar to service provider servers. The overlay nodes cooperate in managing a public blockchain that records transactions.

The scalability challenge is addressed clustering overlay nodes and assigning Cluster Head (CH) for managing public blockchain. A lightweight consensus algorithm as also proposed for limiting a number of new records introduced by CHs.

As verification of new blocks carries significant computational burden, Lightweight Scalable Blockchain (LSB) makes use of a distributed trust algorithm. Every CH shares its information about validity of new records. If the CH trust each other, less transactions need to be verified.

To address throughput issues, a Distributed Throughput Management mechanism is proposed. It dynamically adjusts system parameters and ensures that the throughput of the public blockchain does not exceed the network transaction load. On the other hand, it improves the scalability since more transactions can be appended to public blockchain as the network grows.

The proposed architecture is depicted in Figure 2.1. The LSB is adapted for Smart Home applications network.

It is remarkable that the flow of data is from and to IoT devices and transactions flow are kept separate. Overlay nodes are in charge of transactions while data packets are sent directly to the devices. It allows to reduce significantly possible delays.

LSB requires a separate network with computationally capable equipment what is not always possible. It imposes a tedious infrastructure deployment with complex structure and has sophisticated mechanisms. These requirements make it unsuitable for businesses that need to keep their applications local or just want to prove a hypothesis about benefits a concrete IoT application can bring. Device data format is not specified. Though, privacy and security are

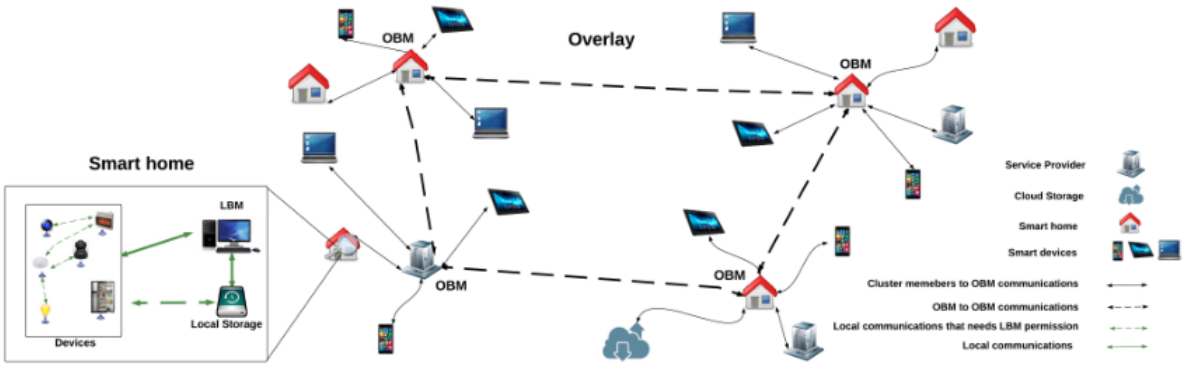


Figure 2.1: LSB architecture overview [30]

natural characteristics of a blockchain-based infrastructure, it imposes throughput issues and does not support real-time communications.

2.1.2 IOTA: The Tangle

IOTA [79] focuses on the same properties as the LSB, namely security and privacy. It aims to create a scalable and sustainable infrastructure with no cost. Only with these conditions billions of IoT devices will be securely connected and a ground for new business models created without predefined rules imposed by economic models.

Traditional blockchain-based architectures always require nodes which carry computational burden of made transactions, i.e. miners in Bitcoin and Cluster Heads in LSB. IOTA made efforts to exclude these entities with the cost of making the architecture collaborative. That is, the devices help each other by vetting transactions. The problem of free riders is solved by punishing devices which do not collaborate. Thus, the principle behind is: "Help others, and others will help you".

The Tangle is a mathematical model designed to make possible such a system. Transactions are seen as vertices in a Directed Acyclic Graph (DAG) and arrows represent validations. Figure 2.2 depicts a part of the concrete Tangle.

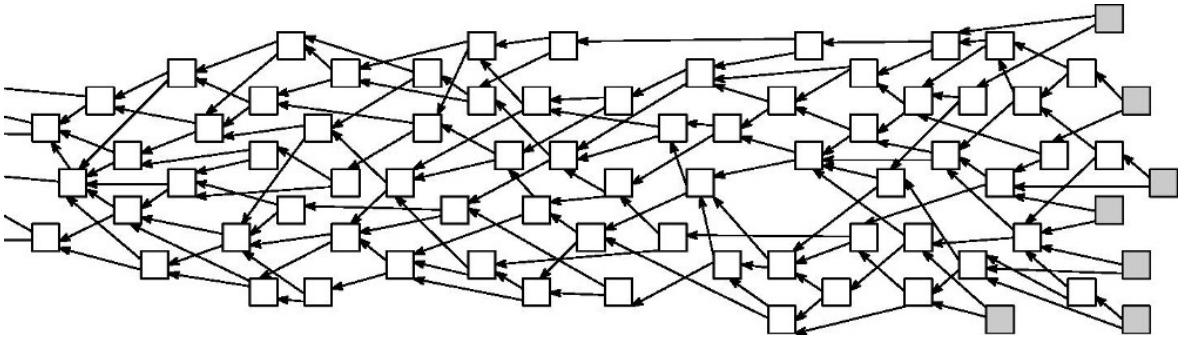


Figure 2.2: A partially representation of the Tangle where the vertices are transactions and arrows their approvals [79]

When a new transaction is issued, it chooses two existing transactions and link to them. Transactions yet without validation called tips.

The transaction just linked indirectly validates all previous transactions. This way transactions which are inside the Tangle can be considered as confirmed.

The structure imposes delays due to the information propagation through the network and the fact that the protocol does not force new transactions link to existent tips.

The tangle does not suffer from scalability problems as blockchain or Ethereum does. The tests showed that throughput of 1000 Transactions Per Second (TPS) can be achieved being 7 TPS current limit for the Bitcoin.

IOTA has many advantages as privacy, security, and scalability, but it does not suit for private business applications. Also, it imposes power burden on the nodes since each message requires a node to validate at least two other tips. It results in a tripled power consumption for data communication and, moreover, validations sum a significant additional computational burden.

2.2 FIWARE

FIWARE [17] is an open source cloud architecture suitable for IoT that was selected by the European Commission for accelerating SME in 2020. It was born in Europe from the Future Internet Public Private Partnership (FI-PPP).

This highly practical initiative project formed an ecosystem of developers, cities, accelerators, innovation Hubs, and more than 1000 SMEs. Expected revenue for 2020 is about 330 million euros.

FIWARE aims to boost startups and SMEs from multiple sectors:

- Smart Cities
- AgriFood
- eHealth
- Transport
- Energy & Environment
- Media & Content
- Manufacturing & Logistics
- Social & Learning

The architecture proposes a universal set of standards for context data management. The core component is FIWARE Context Broker called Orion [18]. The Orion is encompassed by various addtional platform components that provide context data from many sources (IoT sensors, social networks, Customer relationship management (CRM) systems, and so on) and support processing, analysis, and visualization of data. It makes possible creation of new revenue streams from access to context information.

In its communication core it uses NGSI open standard called FIWARE NGSI RESTful Application Programming Interface (API) that favours application logic portability and offers stable framework for future development.

Figure 2.3 shows generic architecture of solutions powered by FIWARE. The core is the Context Broker surrounded by a numerous additional FIWARE components. On the southern boarder the Context Broker interfaces with IoT devices, Robots or other systems and on the northern boarder with data analysis and visualization services.

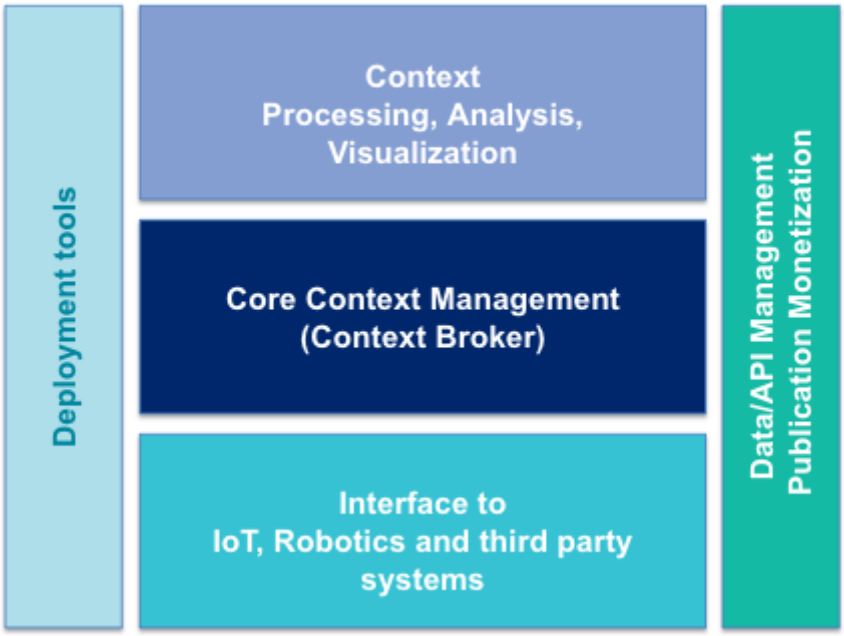


Figure 2.3: Generic architecture of powered by FIWARE platforms [18]

Though, there are not so much information about logic behind the implementation details, the core component, Orion Context Broker, is in charge to manage the entire lifecycle of information stored in a context. It offers FIWARE NGSI RESTful API for data queries, updates, subscriptions, and registrations.

FIWARE is a good candidate for SMEs to take advantage of IoT applications. Further, it is a context manager what gives an opportunity for companies to deeper contextualize their business structure and take more advantages. However, it is still in progress and such important features as security and privacy are not fully developed. It has a complex structure and requires a computationally powerful server. String-based protocols are supported only, and therefore, string based device data formats.

2.3 IoT Architecture in Industrial Processes

The industry 4.0 is a new industrial revolution [101] which partially consists in IoT incorporation into industrial environments. IoT for industrial environment is called Industrial Internet of Things (IIoT). Industrial processes are characterized by real-time behaviour, great precision, and predictability. With that said, an IoT architecture that aims to suit this field must face respective challenges [106] and meet mentioned constraints.

It is vital to note that industrial environment was historically provided with an abundant suit of standards on all levels. Networks for communication on device level are called fieldbuses and are rigorously specified and standardized with corresponding communication protocols. There are well established research groups as Industrial Internet Reference Architecture (IIRA), Reference Architecture Model Industry (RAMI), and others which work on defining standards for IIoT.

Authors of [21] proposed an IoT architecture in industrial processes which combines renown Fieldbus [94] standards, with well defined international ISO standards and intelligent use of MQTT [88] communication protocol.

They do not change the use of well-known industrial protocols and standards as Ole for Process Control - Unified Architecture (OPC-UA) and Field Device Integration (FDI) at low levels. However, they do propose architectural changes at higher levels maintaining rather the logical structure of ISA-95/88 standards. ISA-95 and ISA-88 describe hierarchical structure of the architecture and define the flow of information between the manufacturer and the operations management while regulating execution processes of control tasks.

The hierarchy introduced by these standards is reflected in MQTT topic composition strategy while the infrastructure is similar to the one proposed in the present document. Hierarchical topic composition is defined as following

`/enterprise/site/area/work_center/work_unit/tag/group/point/operation/value`

where:

- `enterprise` is a complete enterprise regardless geoposition
- `site` represents possible factories
- `area` stands for a specific area within the factory
- `work_center` can represent a production line, cell, or unit or even a warehouse
- `work_unit` depending on a `work_center` can be work unit or cell, or warehouse section
- `tag` is an entry for a control equipment or module
- `group` summarized a set of grouped variables
- `point` represents a concrete control variable

- `operation` is an operation that is intended to carry out over a variable, i.e. read or write
- `value` stands for the variable or `point` value

Figure 2.4 presents the proposed IIoT architecture scheme. It defines Industrial Environment based on Fieldbus standards and MQTT environment. Fieldbus protocols payload must be adapted before MQTT message can be sent to the Broker. For this purpose corresponding Gateways are included. Later, the Broker reports variables to the subscribers that can be a storage and supervision system among others.

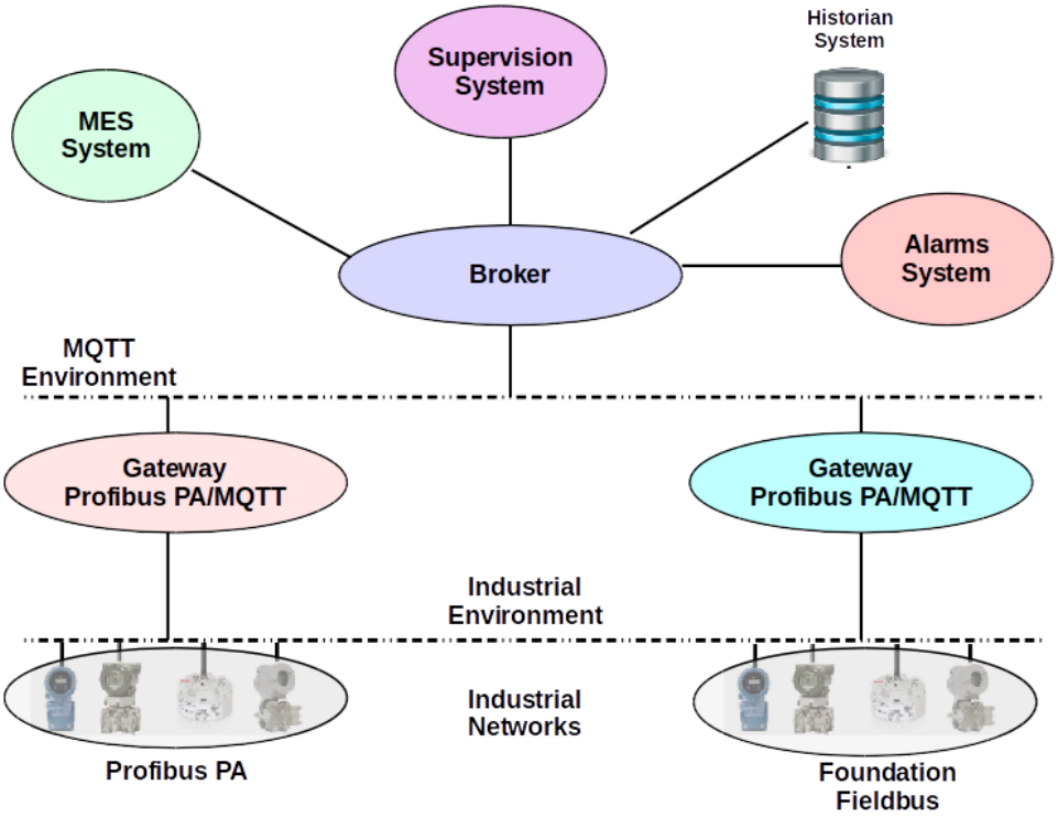


Figure 2.4: Proposed IIoT architecture scheme [21]

It is not usual to convert fieldbus protocol payload to an MQTT message format, thus, the programming of such gateways should be done from scratch to ensure its reliability and assurance. Another potential disadvantage is the real-time behaviour for automation due the supervision system location. For the best performance it must be installed near or on the factory site.

Industrial processes is a very specific application field that requires strictly configured real-time execution environment for running applications. It must be fault-tolerant and have redundant equipment that bring the highest reliability rates. This imposes high resource requirements on every level. The security is based on Secure Sockets Layer (SSL)/Transport Layer Security (TLS) what also brings privacy. As it is an MQTT-based architecture it imposes a certain network overhead [89]. However, this architecture is not low-power oriented as the nodes must be always on monitoring the environment, thus, the network overhead is not an issue.

2.4 Distributed Sensing Architecture

In order to reduce project development time programmers often try to minimize their time creating new code tending to reuse created projects. Such justified and efficient practice can be applied also for projects of larger scale as an IoT architecture.

The approach proposed in [12] called CoIoT takes advantage from existing middleware software called Execution Environment for Highly Distributed Applications (EXEHDA) [59] created for data acquisition routines and IoT devices management. It was used as the core of the proposed Distributed Sensing Architecture.

Authors point out that EXEHDA possesses important properties as being distributed, reactive to sensor data, mobile, and capable to express follow-me semantics. EXEHDA applications can adapt to user movements. Figure 2.5 depicts the architecture of the given middleware.

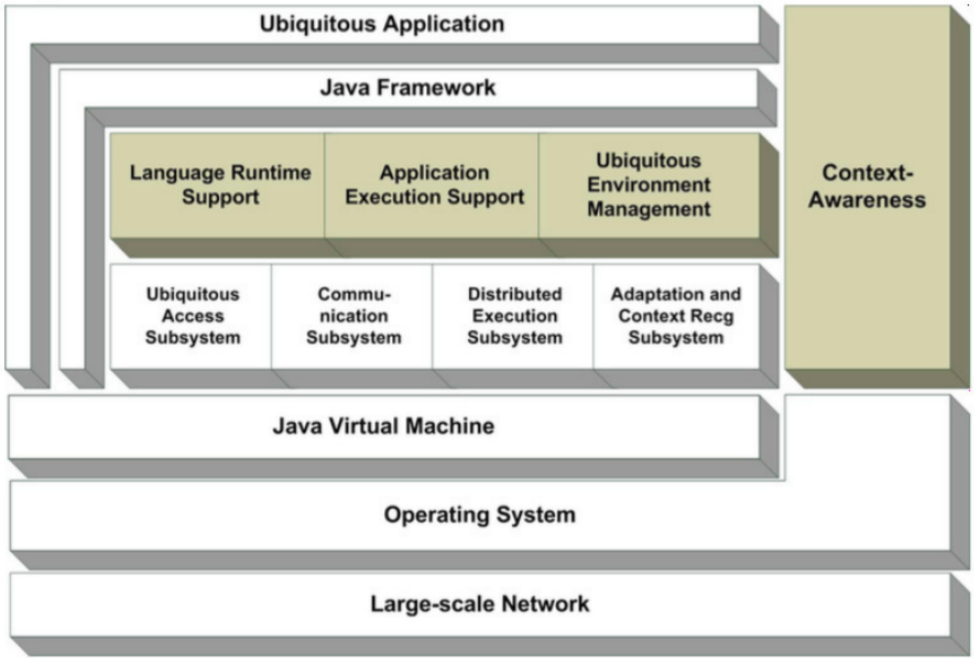


Figure 2.5: EXEHDA middleware software architecture [12]

It is based on various services which cooperate to provide functionality in three dimensions:

1. Ubiquitous environment management for IoT devices layer control
2. Execution environment for user applications with abstractions and services that implement follow-me semantics
3. API for IoT application development

The services are organized in subsystems: large-scale distribution, uncoupled temporal and spatial communication, code and data ubiquitous access, adaption, and context recognition.

The proposed IoT architecture is shown in Figure 2.6. It is quite complex infrastructure with many components and their interactions. This architecture is designed to act autonomously for data recollection and managing physical context treating all devices in an heterogeneous manner.

Data processing responsibility is shared between two servers: Border server and Context server. The Border Server is placed near the application location and is the first step of the data acquisition process. The Rule Engine employs priority scheme for reactive behaviour. Next steps interface the Border Server with the Context Server which receives and stores data on permanent basis. It further offers a user interface and is capable to analyze and visualize the data.

Despite of the complexity presented by the infrastructure is offers more reliability for devices operation. Even if the connection is lost between both servers, the devices will continue to operate without interruption or loss of data. The Context Server offers REST-styled API allowing external services to query existing data and, this way, improving cross-platform interoperation.

CoIoT is not an infrastructure designed for low-resource equipment since Java Virtual Machine (JVM) must be able to run and execute many services related to the infrastructure subsystems. Though the authors claim that a specific priority scheme provides a possibility for reactive behaviour, it is a subject of OS policy for priority and thread management. Standard JVM threads cannot impose their priority to an operating system unless they comply with the Real-Time Specification for Java (RTSJ). The use of such threads bears special conditions related to memory management that significantly complicate the implementation. With that in mind, on the other hand, Java Garbage Collector (JGC), the main cause of execution delays, will make impossible real-time applications execution.

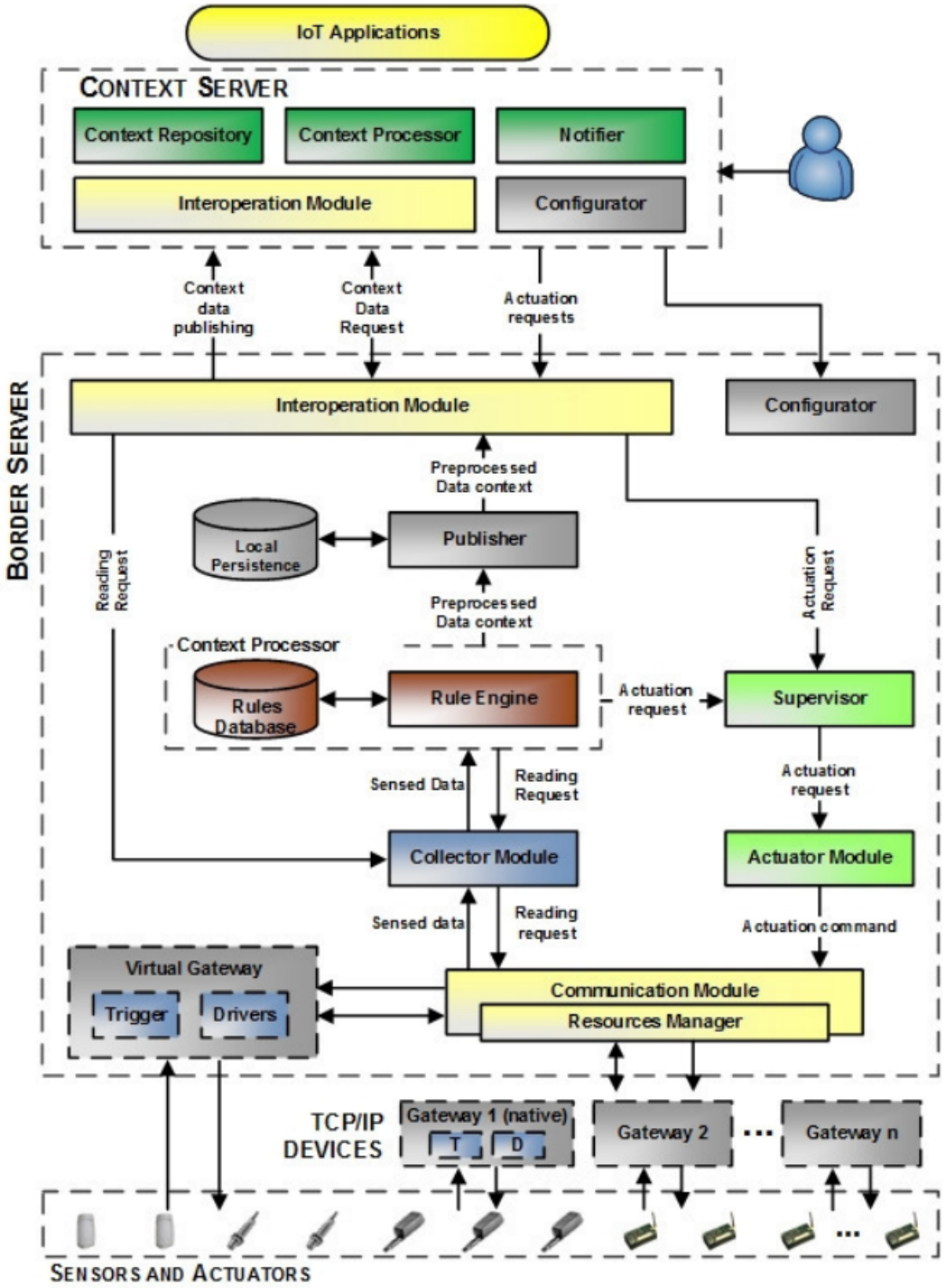


Figure 2.6: Architecture of CoIoT [12]

2.5 Web of Things (WOT) or Lab of Things at UNED

Researchers from Universidad Nacional de Educacón a Distancia (UNED) focus on introducing IoT into the academic world presenting its concepts to the students. In order to improve quality of the distance learning they introduced Lab of Things (LoT) at UNED (LoT@UNED) [76],

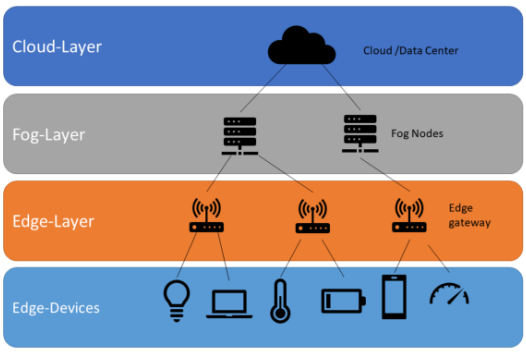
an environment that comprises device and platform layers. Device layer consists of a complete ready device infrastructure based on RPi SoC-boards cluster. The platform level is based on IBM Watson IoT Service [45] which is in charge of device data reception and processing. The storage system is a noSQL database called Cloudant [44]. The IBM service is capable to perform machine learning analysis of data stored in Cloudant. A room for flexibility was provisioned for the cloud platform layer allowing the use of Amazon Web Services (AWS) [2] or Google Cloud Platform [36] services alternatives.

Previously mentioned RPi cluster is separated into two logical groups with possibility for extension. One group incorporates various sensors for students tasks and experiments and another offers its resources without additional sensors due to physical structure limitation of the rack.

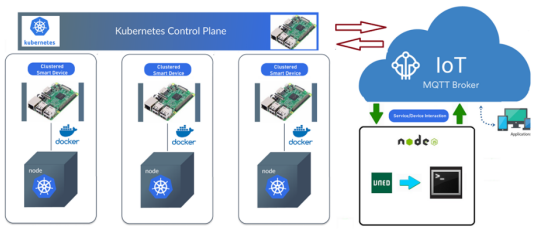
Each boards implements special services which allow the concurrent use of the hardware and sensors connected to the device. Connection to the IoT services in the cloud is seamless and carried out in batch. It simplifies the equipment deployment. The platform associates particular services with each device and the students can request the devices for their needs.

MQTT was selected as the communication protocol due to its popularity, simplicity, and suitability for the purpose. Students should have Python programming skills and be able to use standard libraries to access sensors data. They program data send messages publishing data to predefined topics.

The execution environment provided by virtualization technologies, mainly Docker containers. The containerization allows efficient use of resources and easy deployment. The existence of cluster architecture opens a challenge for orchestrating the containers and their services with redundancy, fault tolerance, and dynamic access management to the devices features. Kubernets [11] was selected as a tool for containers management the provides continuous service delivery with maximum availability. The architecture of WOT inspired by the fog computing model is presented in Figure 2.7a and its technical infrastructure in Figure 2.7b.



(a) WoT architecture model.



(b) Infrastructure scheme for LoT@UNED deployment

Figure 2.7: Diagrams related to the WOT architecture [76]

The current version of the infrastructure is employed using three containers:

1. **IoT** container is in charge of providing a run-time environment for device programming

and sensor access using Python programming language.

2. **Programming** container provides Python execution environment for basic programming activities.

3. **Security** container provides a shell where security commands such as `nmap` [61] , `Wireshark` [73], `route`, and so on are delegated for execution to individual boards.

Additionally, services for user authentication, command history, interaction with devices, and web portal for students and teachers were implemented to provide full-fledged infrastructure functionality.

WoT is an outstanding tool for IoT introduction into the academic realm. It fulfills its educational purpose, but it cannot be applied for custom applications or used by people outside of the university environment. The use of external services as IBM Watson IoT service or, possibly, AWS may seem advantageous, however it comes with corresponding costs and limitations that must be foreseen in the future.

2.6 Autonomous Sensor Network for Rural Agriculture Environments (ASNRAE)

Not all IoT applications can be generalized and often custom solutions are required. Designers more often try to provide dashboard flexibility. Even though, specific designs and architectures suit better for certain applications. [82] proposes a custom architecture solution for agriculture environment focusing on low cost and node energy efficiency through self-rechargeable nodes. It proposes a technical solution for small and middle-size fields.

Described system consists of a WSN connected to the cloud platforms. The WSN consists of a coordinator node working as a base station for the rest of the nodes. A set of nodes in charge of the data acquisition and actuating use the coordinator node to send data and receive actuating commands. Every node uses solar panels for recharging batteries and, thus, increasing energy autonomy. Proposed architecture is presented in Figure 2.8.

The coordinator node integrates ZigBee module for communication with the rest of the nodes, and WiFi for communication with the cloud platform. The coordinator node is based on Arduino Mega [5] board connected with ESP8266 [23] for WiFi integration. The sensor nodes are based on Arduino Nano [68] with integrated ZigBee module.

As was mentioned, ZigBee was selected as the WSN communication technology. The protocol for communication was well-known MQTT protocol. The platform for storing data and visualizing data, as well as, system configuration was selected the ThingSpeak [62]. It conveniently integrates MQTT protocol what allowed its smooth incorporation into the existing infrastructure.

The proposed architecture is a very good example of how energy harvesting can bring energy autonomy to the project and extend nodes battery life. Communication with the whole WSN

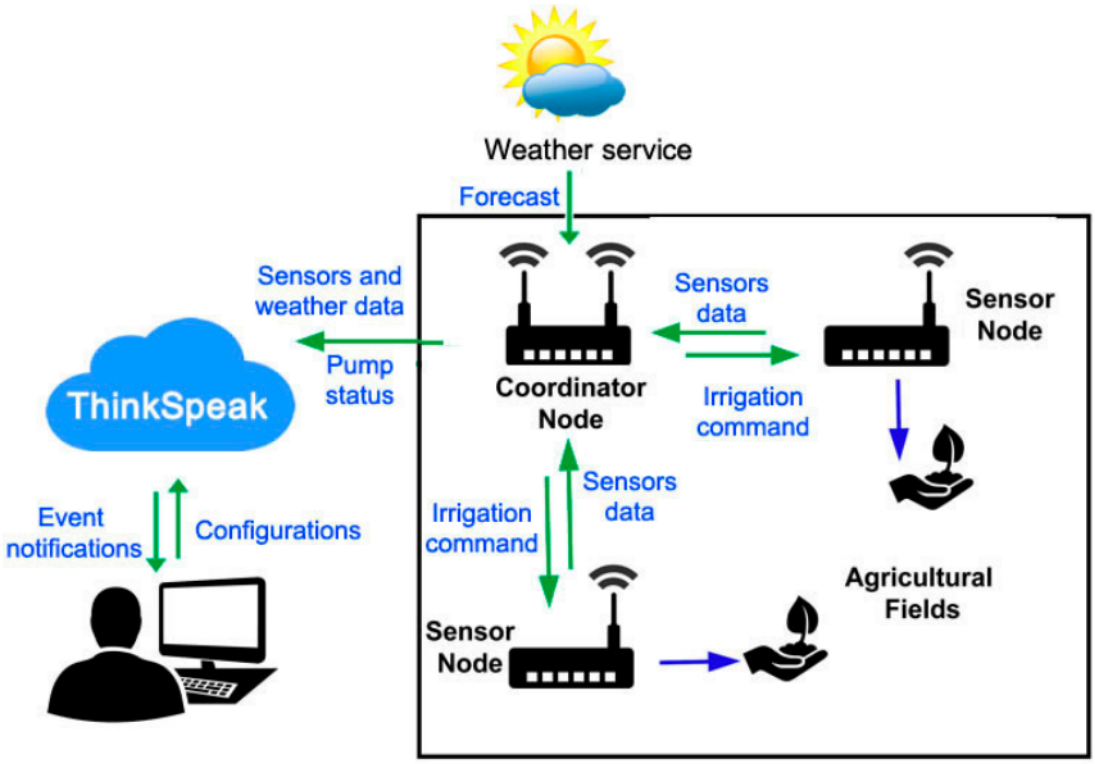


Figure 2.8: Rural Agriculture Environment WSN architecture proposed in [82]

relies on the coordinator node. Thus, it becomes the most fragile point of the architecture. Introduction of a redundant coordinator node would improve the reliability. As the cloud platform is a third-party solution it may become unsuitable for SME or students due to fees. The use of MQTT platform imposes only string-based device data format.

2.7 Discussion of Presented Proposals

Present chapter exposed different existing IoT architectures describing their principles, structure, advantages, and disadvantages. Once the insights from every architecture have been exposed, it will be useful to draw the bottom line indicating presence as well as absence of the most important features.

LSB architecture can be as adaption of the traditional blockchain strategy put into IoT rails. It inherits significant disadvantages from the historical approach, mainly high-resource requirements as cluster heads and the overlay layer of dedicated servers. On the other hand, there are complex distribution management mechanisms aimed to address the scalability and throughput issues. Given the mentioned requirements, it cannot be considered as energy-efficient oriented architecture. Flexibility of device data format was not specified in the paper, being an important feature for the overall efficiency. Neither was indicated whether the device driven workflow is present what would significantly reduce the power consumption of every node. Given the issues of latency, always present in the blockchain-based networks, the real-time features cannot be supported. Though, the end-to-end communication is provisioned; and privacy and security,

the inherent strengths of the blockchain are present.

IOTA is another IoT architecture which is founded on the blockchain philosophy. It does not require a complex system of servers and rules that should govern them. Thus, the required architecture resources are kept at low. As every device must compute at least two transactions for every added message, the architecture cannot be considered energy-efficiency oriented. Device data format was not specified anywhere in the documents. The workflow seems to be device driven as they are the agents who organize and maintain the workflow. The end-to-end communications mechanisms were not specified, but only how they push their data forward. Due to latency issues, IOTA does not provide real-time features. Security and privacy are straightforwardly present.

FIWARE is based on the Orion context broker, the central agent for context management. It requires high computational resources to provide running environment for the Orion and additionally database services for historical data. Since FIWARE has a flexible structure with the possibility of integration of different modules, it is possible to integrate different protocols as so-called IoT Agents which can bring energy efficiency and data-format flexibility. Devices are the agents that alter an application context, they push their data to the broker and it acts depending on its configuration. However, the top-down commands, e.i. configuration, called southern border communication, were not designed for sleeping devices, thus, the devices will easily lose their commands. Therefore, it does not have the device-driven workflow. However, end-to-end communication are fully supported. Further, real-time assures that it can manage real-time devices data, however features as alerts and automation were not specified in the documentation. Security and privacy are not fully implemented yet.

IIoT Architecture proposed in Section 2.3 clearly was not designed to require low-resources, neither to be energy efficient, nor to have device-driven workflow. High-resources is a must to be always in time handling sensors data and actuating over physical environment. Normally, those devices are connected to constant power sources and are always awake. Thus, there is no need for those features. End-to-end communication support was not explicitly indicated in the paper. On the other hand, real-time features must be necessarily supported by this architecture, and security and privacy must be provisioned to keep secure the industrial environment.

CoIoT requires high-resources availability, since it is written in Java , and thus, needs enough resources to effectively run JVM. There is no enough information to state that it is energy efficiency architecture as the paper focuses most on the high-level architectural details. No information about protocols used, neither payload type nor device-driven workflow presence was not indicated. However, the end-to-end device connection is supported in general terms. Java cannot assure the real-time behaviour for its applications unless RTSJ is used. Since the RTSJ was not mentioned, no real-time features are supported. Additionally, no information about security and privacy mechanisms was provided.

WOT was not initially designed to serve IoT applications from real environments, but rather as educational purpose infrastructure that didactically introduces IoT concepts to students. Taking into account that nodes are based on RPi boards connected into two clusters, and that it requires third-party cloud services it can be concluded that it has high-resource requirements. There is no need for energy efficiency, neither for device-driven workflow since the nodes must have high availability to serve student's requests. Real-time behaviour is not a concern in the WOT applications, thus real-time features are not necessarily supported. However, authen-

tication procedure and the use of third-party cloud services impose the security and privacy features.

ASNRAE, similar to WOT, requires the presence of third-party cloud services, therefore the overall resource requirements are high. However, the use of solar photo-voltaic panels enforces the energy efficiency of the architecture recharging devices' batteries in favouring conditions. Since the main protocol chosen for communication was MQTT, data payload from devices is limited to be string-based. The end-to-end communication is supported by ASNRAE, though it was not specified the assured delivery if a configuration message was sent from the platform with the destination device being in a sleep state. There are no on-demand mechanisms as implemented in the device-driven workflows. Real-time behaviour was not mentioned in the paper, however it should be a challenging task to implement it since in-field devices form a WSN. Finally, the security and privacy features were borrowed from the third-party platform.

Table 2.1 summarizes the discussion and matches the main features of the architecture proposed in the present document against the discussed in the current chapter.

Features\Authors	LSB [30]	IOTA [79]	FIWARE [17]	IIoT [21]	CoIoT [12]	WOT [76]	ASNRAE [82]	Author
Low-resource requirements		✓						✓
Energy-efficiency oriented			✓				✓	✓
Device data format flexibility			✓					✓
Device-driven workflow		✓						✓
End-to-end communications	✓		✓		✓	✓	✓	✓
Real-time alerts				✓				✓
Real-time automation support				✓				✓
Security	✓	✓		✓		✓	✓	✓
Privacy	✓	✓		✓		✓	✓	✓
Third-party cloud services required						✓	✓	

Table 2.1: Comparison table of architectures discussed in the present chapter

Chapter 3

Description of the Project

Contents

The idea behind the proposed IoT infrastructure has certain level of technical complexity. However, following its rigid architecture structure it can be clearly explained.

On the practical level, the idea is to facilitate for the developers and businesses an economical way to revolutionize their businesses and solutions. Many businesses can take advantage of the IoT optimizing their operations and reducing expenses, but often it becomes too expensive, especially for small companies, due to popular platforms per-device or per-connection monthly costs. On the other hand, individual developers, hobbyists, and students form the users segment that has constant problems and limitations to find a free or cheap IoT platform for their personal projects. The designed IoT infrastructure is completely free and can be used right 'out of the shelf'.

The installation and hosting of the IoT Platform and gateway do not require special costly equipment. The development was organized in such a way that the user shares computational burden with the platform as much as possible and, thus, the platform does not require much resources. The computational burden for the device data formatting, processing, and representation lays on the user browser. Since modern personal computers and laptops are used to have provisioned enough hardware resources, this fact is taken advantage of.

On the technical level, as was mentioned, the idea is more complex. This chapter will reveal it and put the complex simple. In order to do that, it was structured in a special manner.

Section 3.1 explains the terms architecture and infrastructure in the context of the present document and introduces them accordingly. Since they are structured in layers and encompass numerous details the following sections address every layer. Starting from the device layer, it goes through the gateway layer and finishes with the IoT platform layer. Next, the communication technologies are introduced along with the communication protocol. A crucial part of the infrastructure is the database agent. Thus, its selection and the criteria that were used are explained in Section 3.4. Finally, the software used for the project development are listed explaining how they were used at different development phases.

3.1 IoT Architecture and Infrastructure

First of all, a distinction between the architecture and infrastructure should be defined for the sake of clarity. The architecture refers to an abstract highest-level scheme that reflects the structure in an understandable way. The infrastructure can be considered as a scheme of a particular architecture implementation. It includes more implementation details while hiding specific ones as communication protocols or employed libraries.

The IoT ecosystem consists of a multi-layer architecture. Each layer encloses a complex system o a set of systems with their proper hardware and software architectures that supply necessary functionality. Present IoT architecture is schematically depicted in Figure 3.1

The structure of the architecture clarifies its building blocks. However, it does not provide enough information about the character and nature of layers interactions and how the building blocks become the unified solution.

In order to explain the working mechanisms deeper insights into the architecture ecosystem must be provided, that is, the infrastructure must be presented. Central point of the infrastructure is the data storage. It contains data about users, applications, and devices; and, at the same time becomes the center of communications between the platform and devices. Specific database technology and reasons of its choice will be unveiled later in this chapter.

Looking from the top to the bottom, the users interact with the platform for data visualization, downloading, or device configuration. The platform relies completely on the database. Inverting a view perspective, devices interact only with gateways sending data or receiving configuration messages. The gateways are connected to the database for inserting data or querying the pending messages for the devices. Described infrastructure can be observed in the Figure

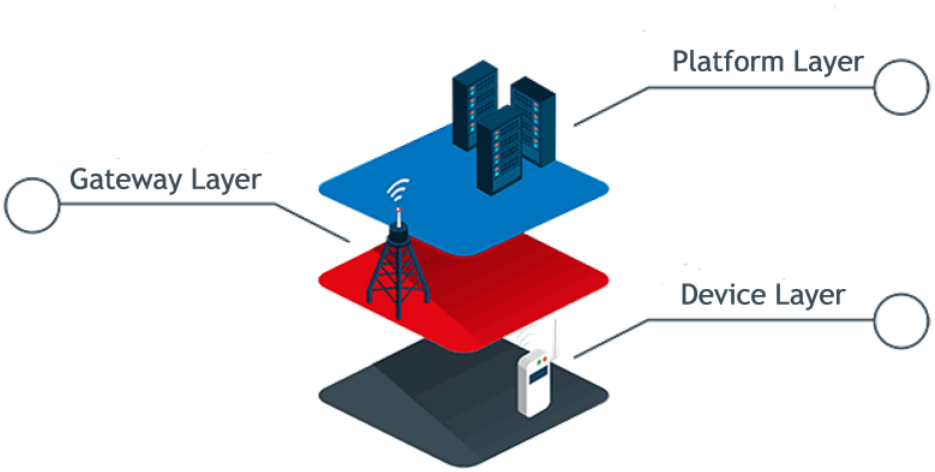


Figure 3.1: IoT Architecture of the project

3.2.

In this section three IoT layers related to the designed architecture will be described: device, gateway, and platform. An additional section related to the communication protocol will be added.

3.1.1 Device Layer

In this layer devices that sense and actuate over their physical environment are located. They are the "Things" part of any IoT project. The devices interact with the physical environment and recollect data of interest about an object under measurement or the environment itself and turn them into useful, analyzable data.

Though, the project itself does not require any specific device type and devices are considered as homogeneous nodes it is good to give an example that perfectly fits into the architecture and give several tips on how to select the right device for a project.

Microcontroller

This is the core component of each sensor/actuator device. It performs all the most important operations and orchestrates the data recollection forwarding them to the next architecture layer.

Selecting the right microcontroller is a challenging task. The designer should not only carefully consider a list of technical features, but also business case issues like costs and lead-times that can potentially hinder the project [72].

There are three major steps that are to be taken usually before the microcontroller has been selected for a particular project. First, a list of required hardware interfaces are to be

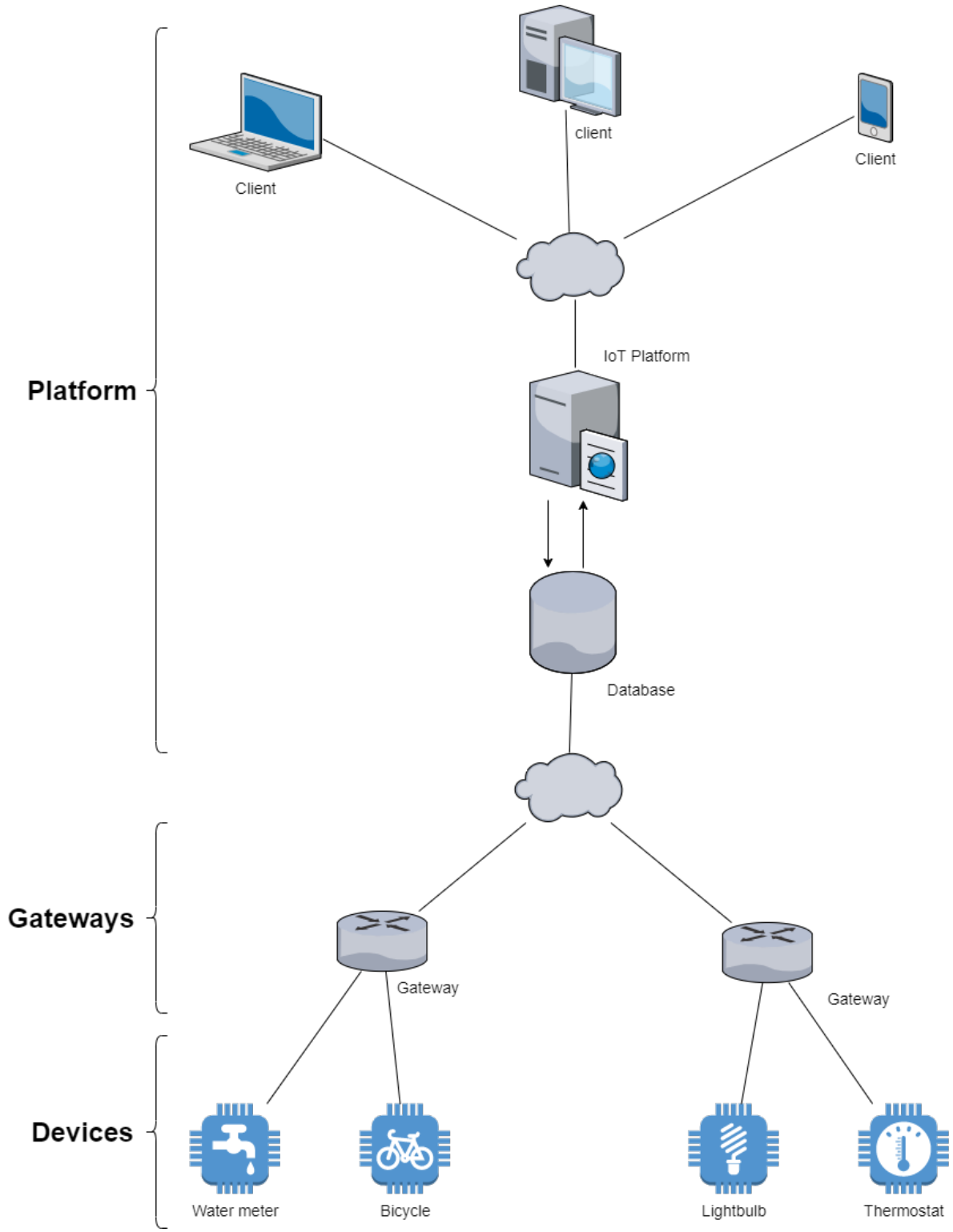


Figure 3.2: IoT Infrastructure of the project

composed. Required communication interfaces usually fit into well-known standards list as Inter-Integrated Circuit (I^2C) [49], Serial Peripheral Interface (SPI) [48], or Universal Asynchronous Receiver/Transmitter (UART) [63]. It should be taken into account how much space each

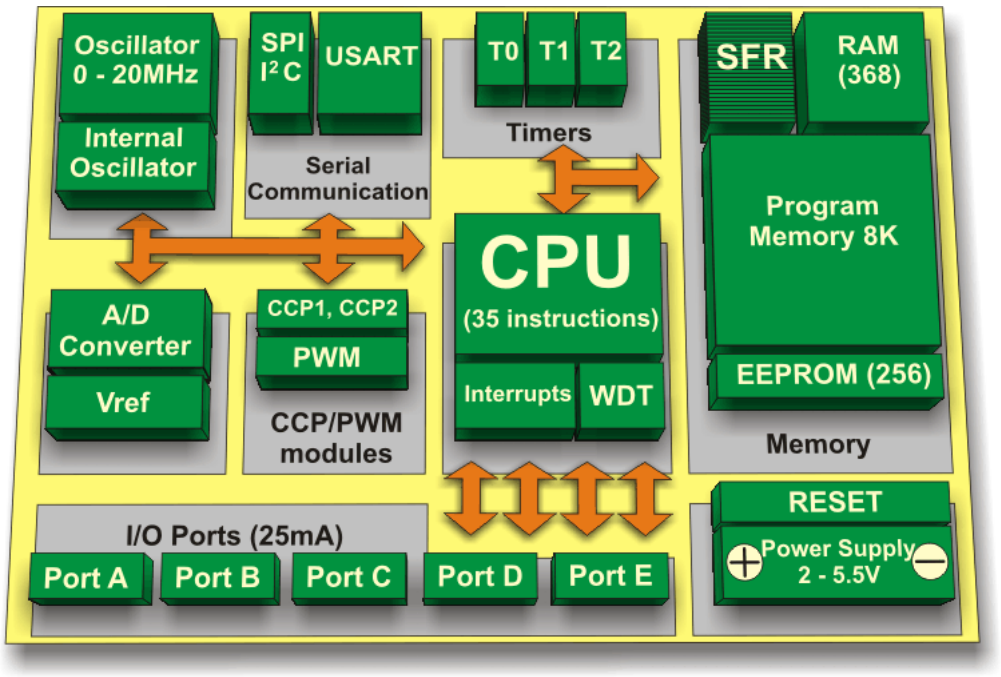


Figure 3.3: A typical microcontroller block diagram

interface driver occupies since microcontrollers are limited in power, memory, and functionality systems. Other kinds of interfaces that should be considered are digital inputs and outputs, Analog-to-Digital Converter (ADC), Pulse-Width Modulation (PWM) [41] and so on. Necessary number of General-Purpose Input/Output (GPIO) pins are to be considered for the required application as well.

All important features related to a microcontroller architecture are, by a common convention, depicted using a block diagram. Typical block diagram can be seen in Figure 3.3.

Additional features can also be present as deep sleep modes, active Random Access Memory (RAM) memory for sensors control in low-power modes, internal precise Real-Time Clock (RTC), or an embedded communication module.

Sensors and Actuators

Sensing is a technique used to gather information about a physical object or process [22] including the detection of occurring events. A sensor is a technical device that transforms events or characteristics from the physical world into electronic signals which can be analyzed. Another word for a sensor is a transducer, term related to conversion of energy from one type to another. A sensor also can be seen as a kind of transducer that transforms energy from physical world to electrical energy suitable for computers and electronic systems analysis.

Actuators are the devices that allow electronic systems directly control the physical environment. It can be motor for opening a door or relay that activates a thermostat. It would be appropriate to illustrate both processes, sensing and actuating in a schematic way as in Figure

3.4.

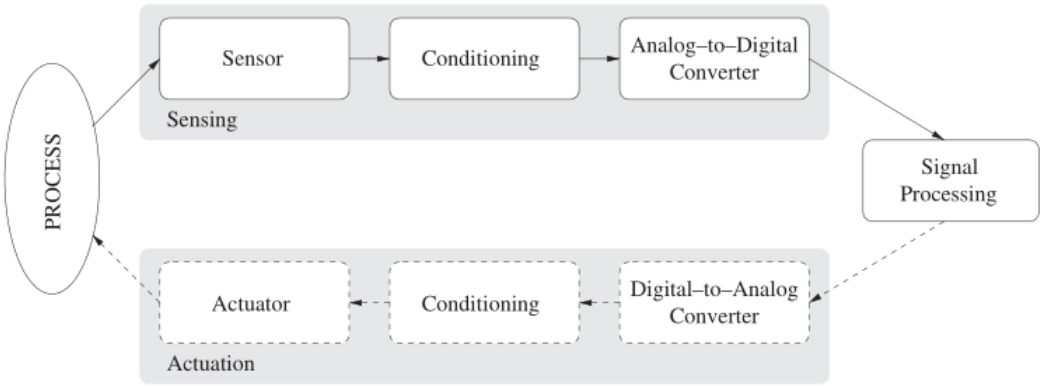


Figure 3.4: Sensing and actuating processes.

Processes in a physical environment are captured by a sensor device. The captured signals then pass through a conditioning phase where they are filtered or amplified, and, finally, converted into digital realm with an ADC. On this stage a digital value can be analyzed and, thus, the sample will be characterized. The process for actuators is straightforwardly inverted.

Real Time Clock

In order to reduce power consumption in the uttermost way or to maintain time synchronization for particular project purposes an external RTC is required. In spite of possible availability of an internal RTC inside the microcontroller the use of an external one is advisable. In practice, when the internal RTC is used, its circuitry, which can be shared with other peripherals, has to be powered on what increases its power consumption and does not allow the Microcontroller (MCU) to go to the deepest sleep states. The RTC, configured usually through I^2C or SPI serial bus, will generate interruptions to wake up the device for an event while the microcontroller has been in the power saving mode.

Flash Memory

The flash memory is supposed to store the most important event records which can be sent to the server at any moment and other log information. It will be the important piece of hardware when the firmware over-the-air update functionality will be integrated. The downloaded firmware will be stored in this memory and loaded by the bootloader after the update finished.

By convention, flash memories communicate with MCUs through the standard SPI interface.

3.1.2 Gateway Layer

After the data have been sampled, digitalized, and processed they are prepared for further data transfer. The gateways receive the aggregated and processed data. It is important to note that

the communication with the devices is realized using a specific communication technology as LoRaWAN [87], Sigfox [107], ZigBee, Wi-Fi [20], Bluetooth [64], or custom fixed radio frequency. Then, the gateway routes the data to a database or other networking processes. This time the transmission is carried out usually through another network interfaces which possess higher bandwidth. Ethernet or General Packet Radio Service (GPRS) [8] are typical examples of such interfaces.

Gateways often sit close to the sensors and devices. However, it depends on the chosen communication technology. It is often a matter of the characteristics and requirements of the project. Basically, there are several factors that are considered when a communication technology is being chosen:

1. **Devices dispersion**: If devices are located in the same area like a building, an industrial plant or even a neighborhood, short-range technologies like Wi-Fi, ZigBee or even Bluetooth would be a good choice. Otherwise, if they are widespread in a city or country, long-range technologies like LoRa [7] or Sigfox may be the right choice. In other cases median-range technologies based on radio frequency might be suitable.

2. **Required Data Rates**: Some networking protocols are not suitable depending on the amount of data that the devices send. For instance, Sigfox and LoRa do not provide enough bandwidth when a sensor must send a room temperature or state of a parking lot every minute, whereas Bluetooth or Zigbee fit for more data-intensive applications.

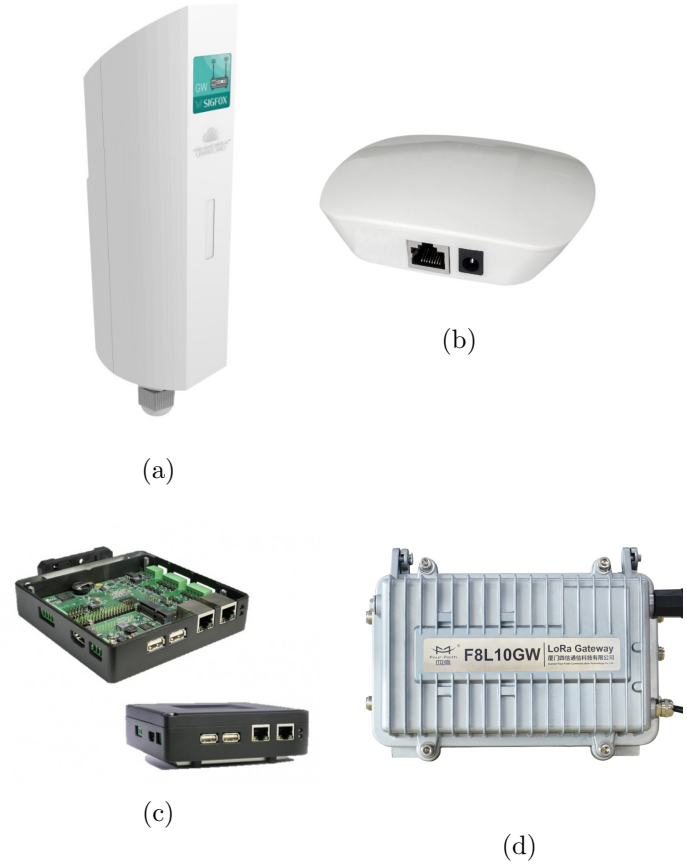
3. **Network coverage**: For wide coverage, it is possible to deploy a LoRa-based private network or use The Things Network (TTN) [69] as public LoRaWAN network provider. Alternatively the wide-range network infrastructure can be provided by mobile operators (2G [77], Narrow-Band IoT (NB-IoT) [10]) or private companies as Sigfox. For short coverage, ZigBee and conventional Radio Frequency (RF) transmitter are efficient options.

It is also worth to note that gateways can have three or four communication technologies integrated to aggregate data from multiple networks. Samples of different gateways are presented on Figure 3.5.

Detailed description of the gateway developed for this project will be presented in Chapter 6.

3.1.3 IoT Platform

The software platform of any IoT project will be in charge of managing applications, devices, and visualizing data to a limited extent. It supervises devices onboarding processes and monitors the current internal configurations and functionality. Using the cloud platform it is possible even to update device firmware when the device is appropriately programmed and such functionality is supported by the platform.



(a) (b) (c) (d)

Figure 3.5: Examples of gateways employing different communications technologies. 3.5a Sigfox, 3.5b ZigBee, 3.5c multiple technologies, 3.5d LoRa

Management Service

Data processing requires unambiguous identification of each datum. API for reading and gathering data must be provided. All these details comprise a management service system upon which reigns concrete applications.

When a particular device reports a change of state, it transmits the data according to a designed protocol. These data are processed according to the provided API and visualized for the corresponding application.

Applications

All IoT projects are carried out for a purpose. IoT applications are just software systems which use the data that are received from the devices and the functionality that they provide. Depending on the level of IoT project customization, three categories can be defined:

1. **IoT vertical applications** which provide out-of-box functionalities for a specific application domain like smart waste management, smart building monitoring, smart water metering, smart watering, or smart parking.
2. **Toolboxes and frameworks** for building a private dashboards, reports, alarms, and graphics. These can be independent products which integrate with external data sources or they can be provided as a part of the IoT software platform.
3. **Custom software applications** which are developed from the ground up using standard software development technologies. These applications will use the IoT software platform APIs as the foundation for building their functionality.

The designed IoT platform and the infrastructure is capable to offer solutions from the first and second categories and can be extended to offer the third category as well. Its detailed description can be found in Chapter 7.

3.2 Communication Technologies

Communications is one of the most important part of any IoT infrastructure without which the concept of IoT would make no sense. Communications cover each step of system functionality, from the architecture to data visualization.

MCU is the core of commanding and orchestrating the system work. It should communicate with each Integrated Circuit (IC) according to the specified protocol and govern them sending commands and receiving responses. After the data have been prepared to send another dimension of communication starts, wireless data transmission which involves completely different technologies and protocols. Thus, there are two dimensions of communications: horizontal or inter-IC communications and vertical or communication with gateways.

Horizontal Communications

This section will provide a brief introduction into inter-IC communication protocols used in the project.

In order to achieve a particular functionality, it is indispensable to coordinate operations performed on the device. Each IC carries out its functions, and it must be directed by some means. For this purpose there were engineered inter-IC protocols.

The protocols typically used in every IoT project are listed bellow.

I^2C The Inter-integrated Circuit Protocol is a protocol intended to allow multiple "slave" digital integrated circuits ("chips") to communicate with one or more "master" chips. Similar to the SPI protocol it is intended only for short-distance communications and like UART it requires only two signal wires to exchange information that can support up to 1008 slave devices.

I^2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can't talk to each other over the bus and must take turns using the bus lines). Most I^2C devices can communicate at 100kHz or 400kHz [49].

Each I^2C bus consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master; some slave devices may force the clock low at times to delay the master sending more data (or to require more time to prepare data before the master attempts to clock it out). This is called "clock stretching" and is described on the protocol page.

The detailed description of I^2C protocol is not in the scope of this project.

SPI

Serial Peripheral Interface is an interface bus commonly used to send data between microcontrollers and small peripherals such as common ICs, shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to [48].

It's a "synchronous" data bus, which means that it uses separate lines for data and a "clock" that keeps both sides in perfect sync. The clock is an oscillating signal that tells the receiver exactly when to sample the bits on the data line. This could be the rising (low to high) or falling (high to low) edge of the clock signal; the datasheet will specify which one to use. When the receiver detects that edge, it will immediately look at the data line to read the next bit. Because the clock is sent along with the data, specifying the speed isn't important, although devices will have a top speed at which they can operate.

SPI advantages: it's faster than asynchronous serial, the receive hardware can be a simple shift register, and it supports multiple slaves. The disadvantages are: it requires more signal lines (wires) than other communications methods, the communications must be well-defined in advance, the master must control all communications (slaves can't talk directly to each other), and it usually requires separate SS lines to each slave, which can be problematic if numerous slaves are needed.

It is noticeable that the I^2C possesses more advantages and is more attractable during the design process.

Vertical Communications

When it comes to vertical communications, it should be thought as a way to communicate or report collected data to a gateway which is normally located at a long-distance range from the device.

All wireless communications are based on radio frequency (RF). Microcontrollers have on-chip or on-Printed Circuit Board (PCB) RF module which is used to transmit and/or receive radio signals between the device and a gateway.

RF used for data transmission must comply with the Frequency Spectrum Allocation (FSA) standard bands. The unlicensed Industrial, Scientific and Medical (ISM) and Short Range

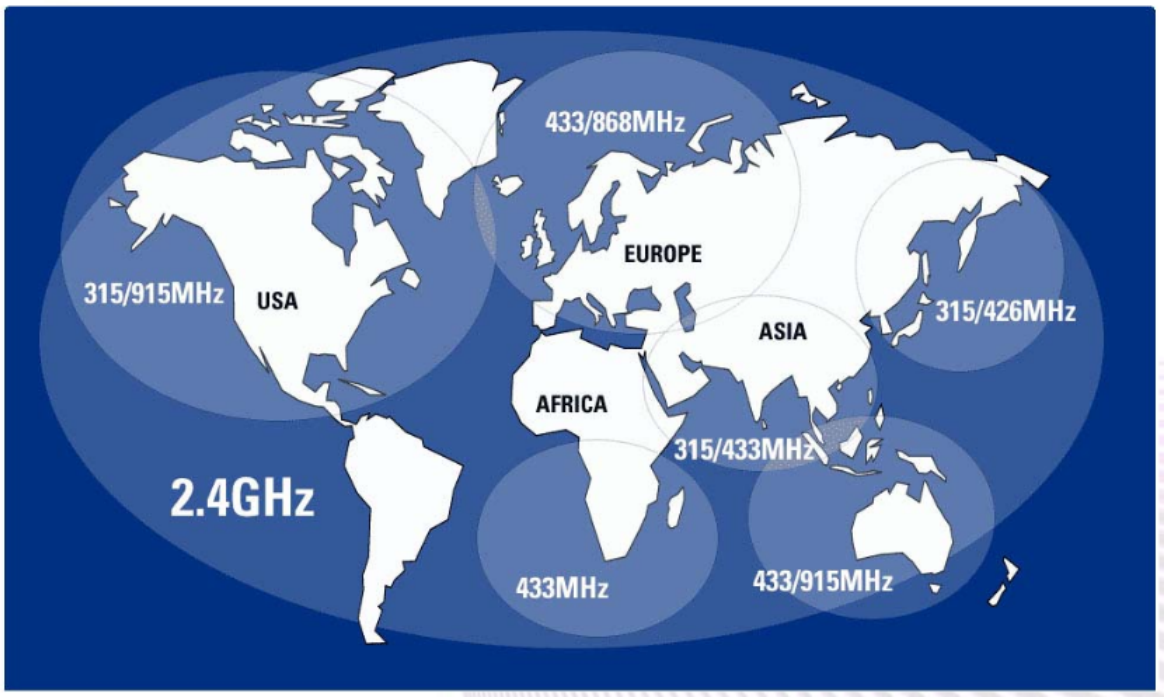


Figure 3.6: ISM/RSD License-Free frequency bands

Devices (SRD) bands are used for this aim. The license-free band frequencies according to geolocations can be observed on the Figure 3.6.

In Europe the allowed frequencies are listed bellow:

- 433.050 – 434.790 MHz (ETSI EN 300 220)
- 863.0 – 870.0 MHz (ETSI EN 300 220)
- 2400 – 2483.5 MHz (ETSI EN 300 440 or ETSI EN 300 328)

The availability of unlicensed RF bands opened possibility to develop different communication standards many of which are used today in IoT applications. Table 3.1 presents the most widely used technologies.

Name	Frequency	Range	Data Rate
3G	1.6-2.0 GHz	Long Range	Up to 2Mbps
4G	2.0-8.0 GHz	Long Range	Up to 1Gbps
5G (soon)	3-300 GHz	Long Range (network)	¿ 10Gbps
LoRa	Licence-free RF bands	15 km range	LPWAN, up to 50 Kbps
SigFox	Licence-free RF bands	15 km range	LPWAN, up to 100 bps
Ingenu	2.4 GHz	Long Range	LPWAN, up to 2.5Gbps
ZigBee	ISM	10-100 meters	up to 250Kbps
Wi-Fi	ISM	10-100 meters	standard-dependant
Bluetooth	2.402-2.480 GHz	30 meters	up to 2Mbps (version 5)

Table 3.1: Most prominent RF IoT communication standards [31]

3.3 Communication Protocol

It is very insightful to link the IoT architecture functionality with communication protocols. Devices exchange messages with gateways using one of the technologies presented in Table 3.1. All of them define specific protocol stack. For instance, it is known that everything in the Internet is related to protocols and Transmission Control Protocol (TCP)/Internet Protocol (IP) is de facto Internet protocols stack. It means that all information is transported encapsulated in standard packets as IP and TCP or User Datagram Protocol (UDP). Similarly, device data is encapsulated into lower-level protocols which are technology dependant. Depending on selected communication technology packets of Data Link Layer (L2), type of encoding and nature of transmission signals on Physical Layer (L1) will be different.

Among all these diversity there is a constant top layer, that is, the Application layer. It is where application-specific protocols take place. Figure 3.7 illustrates an example where a message with custom IoT communication protocol is sent over using the Internet stack.

In case when an IoT application is deployed over ZigBee or similar RF technology, the application layer in the protocol stack will be placed immediately after the Data Link Layer since these technologies were not designed for the Internet transmission.

Apart from the syntactical aspects, protocols have semantics, that is, certain meaning conceived for every message type. This is the key point that brings a desired functionality to a particular IoT infrastructure.

The syntax and semantics of the communication protocol specifically designed for the present project will be described in Chapter 5.

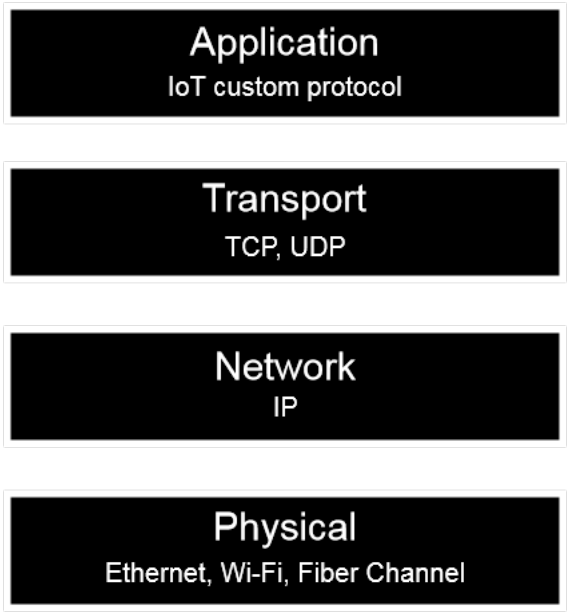


Figure 3.7: Appliction-specific IoT custom protocol on top of the Internet stack

3.4 Database Technology

Database technology is a software that stores, organizes, and processes information in a manner that end-users can always conveniently find stored information employing database query language. There are databases of different forms and shapes, less or more complex, for small or large amounts of data. During last 20 years of database development many different technologies were created, but not all of them were embraced by developers and widely used. Commonly used databases for the time of the project development are separated in two groups relational Structured Query Language (SQL) and non-relational noSQL databases.

SQL databases were primarily used for data storage more that forty years. MySQL [66], PostgreSQL [65], SQLite [75] are widely-used databases. NoSQL databases were created in 1960s but recently grown in popularity. MongoDB [15], Redis [13], and CouchDB [4] are the most popular NoSQL databases.

SQL databases store information in related data tables where rows represent records and columns record attributes. A database model with exhaustive definition for every table, column types and relations among tables called SQL Schema is required for its creation. In contrast NoSQL databases store JavaScript Object Notation (JSON)-like key-value pair documents. NoSQL allows complete freedom on a way a developer can store application data. An important difference is that NoSQL documents can store other documents or objects embedded. This is now allowed or a cumbersome way to store data in the SQL databases.

There are many dimensions on which both families can be compared and its comparison is expressed in Table 3.2.

Dimension	SQL	NoSQL
Nature	Relational	Non-relational
Design	Based on the concept of table	Based on the concept of document, key-value, column-oriented, and graph-oriented
Scalable	Tough task to scale big data	Easy scaled for big data
Model	Detailed database schema is required	No need for any database model
Community	Vast and expert	Very wide
Standardization	SQL standard language	Lack of standard query language
Schema	Rigid	Dynamic
Flexibility	Not flexible design	Extremely flexible

Table 3.2: SQL and NoSQL database technology comparison [24]

The question of which database technology is more suitable to the IoT infrastructure is not trivial. It requires a deep project analysis and clear purposes. The questions to answer are all related to the data. They are presented in the following listing with the answers which make basis for the final choice.

- What processing and decision making will be required?

  The current version of the platform does not aim to extensively process device data neither apply any machine learning techniques to introduce special insights or predictions. These features can be applied in future versions.

  The platform will visualize data of every device for reasonable period of time and provide comfortable access to explore them and download as a Comma-Separated Values (CSV) format file.

  Devices will be able to send data in three formats: JSON, MessagePack [35], and binary formats. Thus, the database must be able to store devices data in a common format, binary, and every application will retrieve and decode the data according to defined devices data models.

  The decisions will be required to analyze devices behaviour for sending alarms and enqueuing automation messages.

  Publish/Subscribe functionality will be required for real-time platform notifications.

- What volume of data will be transmitted by the devices?

  The platform was not initially thought to be deployed as a geo-distributed service with millions-of-devices connection capability, but rather as an efficient server that every small business can economically deploy and quickly prototype and use its IoT solution.

  The limitations rise from the active database connections capability. Thus, the database technology of choice should be able to manage about 50-100 gateways and every gateway should manage between 100-1000 devices. Maximizing the amount of devices, the overall number should turn about 100000 devices per complete infrastructure.

  Every application was thought to manage up to 256 devices, thus, the platform will be capable to hold about 392 application.

Of course, those estimations are not always the case, as it unusual to have 256 devices connected to one application. Usually this number is much lower. How often devices will contact with gateways is yet another parameter to take into account.

The estimations were made for data-intensive applications where devices data send periods are of 1 minute, what is not always the case, with the payload size is 100 bytes. Thus, if 100000 devices will send 100-bytes-payload messages every minute, then every day overall data augmentation will be about 13.4 Gb.

- Will devices be controlled or configured remotely?

  Yes, an efficient and simple form of device control will be required for remote configuration and automation messages.

- May the machine that will hold the database be limited in resources?

  The platform is thought to be lightweight and capable to run on a RPi or similar Linux-based SoC board.

The possibility of having support for different database technologies was seriously considered. Using software patterns as Data Access Object (DAO) described in detail in Section 7.2.2 it is possible to adapt the infrastructure for different database technologies.

Taking into account all questions, as the best initial option was considered PostgreSQL database. It effectively runs on resource-constrained devices and possesses the publish/subscribe mechanisms required for real-time features. Section 7.3.6 explains the mechanisms and routines used to support publish/subscribe functionality. Connections are constrained by a host operating system and, for Linux the maximum is usually set to 100 what suffices the needs. Last but not least, it provides a room to implement mechanisms for device automation and remote configuration. Additionally, in case of high-resource availability, DAO for any noSQL database can be implemented, i.e. MongoDB or Redis, and thus, the infrastructure extended.

3.5 Development Software

All information technology practical projects require a set of computer programs that facilitate their implementation. This section will describe which software was used on each step of development giving a brief insight into a corresponding contribution.

There were three main phases of development:

1. Communication Protocol. Device Firmware
2. Gateway Firmware
3. IoT Platform

Though the device firmware is not a direct part of the project, it will be useful to describe the software used for its development since the communication protocol was implemented, debugged, and improved during the device firmware development.

3.5.1 Communication Protocol. Device Firmware

As it is known, firmware is a specific software that provides the low-level control for the device's specific hardware. Particularly, when it comes to embedded systems, it is the only program that will run on the system and provide all its functionalities.

Depending on a project, a particular device will require firmware which should implement its functionality, diligently care about power consumption, and manage network communications.

The way of firmware programming strongly depends on the MCU manufacturer chosen for the design. Each MCU has its method of being programmed and each manufacturer supplies for this compilers, frameworks, and toolchains.

The software which was used during the communication protocol and device firmware development and the functionality it brought are listed below:

Visual Studio Code [47] is a source code editor developed by Microsoft for Windows, Linux and macOS. It becomes vary developer-friendly when it comes to features it has. Some of them are support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is free and open-source, although the official download is under a proprietary license.

There is one important plugin which converts Visual Code Studio into a great IDE for IoT systems development, mainly PlatformIO [90]. Since all firmware development lies under C/C++ programming languages it introduces Intelligent Code Completion and Smart Code Linker for rapid professional development and multi-projects workflow with Multiple Panes.

Special advantage of PlatformIO is Multi-platform Build System without external dependencies to the OS software: more than 400 embedded boards, 20 development platforms, and 10 frameworks. Arduino and ARM embed [46] compatible.

Since during the communication protocol and device firmware development the ESP32 MCU was used, it was a perfect IDE for rapid and efficient programming and loading the source code into the device memory.

HTerm [39] is a terminal program for the serial interface that runs under Windows and Linux. Its features make it a great debugging tool. Support of connection with any kind of serial ports; all baud rates available on the target hardware; input and output transmissions in ASCII, hexadecimal, binary and decimal formats via those ports; sending and saving files; parity for send and receive; copy any received formats to the clipboard are distinguishable features of the program.

It served as a powerful debugging tool in the project. Continuous system monitoring was possible and used for error correction during the cyclic process of device firmware development.

The device firmware was written in C/C++ programming languages.

3.5.2 Gateway Firmware

Gateway is one of the key points of the infrastructure. It must, on one hand, efficiently implement routines to interact with the devices what depends upon a selected communication technology and, on the other hand, communicate with the database.

Gateways are usually Linux-based systems, though not necessarily, which are placed remotely respect to a developer. The firmware developed for the present project is intended to effectively run on a Raspberry Pi board or any other Linux-based system. It was not implemented on the developer's system, but the developer connected to the gateway through a Secure Shell (SSH) [105] service where the development took place.

The software which was used during the gateway firmware development and the functionality it brought are listed below:

PuTTY [92] is a free open-source terminal emulator, serial console that supports SSH, Telnet, and SCP network protocols and can connect to a serial port.

With this well-known piece of software the connection to the gateway and all necessary actions for file management and operating system navigation were made possible.

Vim [19] is another well-known text editor for command line interface software development. It is a layer of additional plugins that empower de facto Unix editor 'vi'.

It is highly configurable text editor that, once a developer learn to use its features, enables efficient text editing.

It was used as an IDE for the gateway firmware development.

The firmware was written in pure C programming language (C11).

3.5.3 IoT Platform Development

The platform development is the heaviest part of the project. It consisted of a back- and front-end server development process.

As the platform intended to be lightweight and resource-efficient, the development initially took place on a Raspberry Pi board but later it moved to a virtual machine environment since the board suffered lethal breakdown. The virtual machine used Raspbian, a Debian-based operating system with the adjusted correspondingly hardware resources.

VirtualBox [100] is a powerful general purpose cross-platform open-source virtualization software. It gives a hosting operating system (OS) an capability to run different OSs inside it. With its use a user can easily run Linux-based OS on a Windows machine.

It is very simple and incredibly powerful tool that millions of developers use worldwide.

It hosted the IoT platform during its development phase.

Flask [38] is lightweight Web Server Gateway Interface (WSGI) web application framework. It offers simple and powerful interface to quickly develop web applications from trivial up to complex level of complexity.

It allows a complete freedom on what external libraries and tools are used to integrate functionality to a project. User management, flexible database connection, incredibly useful wrappers and decorators, easy file upload and download, and easy to understand and develop Jinja2 template system are some the features that enriched the present project.

It was chosen as the best candidate for the platform development to maintain its resource efficiency and future scalability.

Vim and PuTTY were used for the platform development identically as during the gateway firmware development phase.

Firefox is a well-known web browser and was used for platform troubleshooting and testing.

The IoT platform was implemented mainly in Python programming language, and additionally in HTML5, CSS3 and JavaScript.

3.6 System Requirements and Installation

The development of the project was completed for the moment of writing the document. Thus, it is possible to provide information about the OS minimum requirements, current running deployment of the platform, links to the public repositories with the source code, and installation instructions. This section provide these details.

3.6.1 Minimum System Requirements

The platform was designed to run on a Linux-based system. The efforts to keep the hardware requirements as low as possible providing real-time features and quick IoT Platform web response were carried throughout the whole development. The resulting minimum systems requirements are as follows:

- **OS :** Debian-like (Debian, Raspbian, Ubuntu)
- **CPU :** ARM Cortex-A53, 1.2GHz
- **RAM :** 1GB
- **Storage :** 16GB

From the above technical specification it can be observe that the whole infrastructure can run on one Raspberry Pi 3 board.

3.6.2 Running Deployment and Source Code

Once the developments were finished the first deployment version of the IoT Platform was ready. It was installed and is available on `http://51.254.120.244:8080/`. Current platform configuration allows autonomous user registration. The reader can access and download the source code of the IoT Platform at `https://lorca.act.uji.es/gitlab/vrykov/thso.server/-/tree/dev`.

For the moment of writing the document there is only one developed version of the gateway. It serves requests in the Communication Protocol format described in Chapter 5. It listens on `54345` UDP port on `51.254.120.244` IP address. The gateway firmware and communication protocol source code is available at `https://lorca.act.uji.es/gitlab/vrykov/thso.gateway` and `https://lorca.act.uji.es/gitlab/vrykov/thso.gateway_protocol` repositories accordingly.

3.6.3 Deployment Instructions

1. Create folder for the IoT Platform

```
$ mkdir hpca_iot
$ cd hpca_iot
```

2. Clone the project

```
$ git clone https://lorca.act.uji.es/gitlab/vrykov/
      ↪  thso.server
$ cd thso.server
$ git checkout dev
```

3. Run preinstallation script which will create virtual envoronment, install all necessary C libs and python dependencies, and export environment variables.

```
$ sudo ./preset.sh
```

Now the server is installed and ready for configuration. There are 3 types of configuration: environment, application and server. The environment can be production, development and testing. The server can run completely differently for each environment depending on your configuration respectively. The default environment is 'production'. ('production' environment requires for https and certificate presence for secure session management. If the certificate is not available, change to development environment to one that does not use encryption for session data).

```
$ export FLASK_ENV=production
$ export FLASK_ENV=development
$ export FLASK_ENV=test
```

The application configuration basically describes database connection and other parameters as secure cookies or debug mode. On the other hand, the administrator performs server (uWSGI) configuration which is located in `app/server.ini`. In this file he can define concurrency parameters, that is, how many processes and threads will be used by the server, or on which port it will be listening.

Finally, there are two ways of executing the platform and gateway. The first and the easiest way is to run directly the next command.

```
thso.server/app $ uwsgi server.ini
```

Alternatively, the administrator will have to add two `systemd` scripts for describing the IoT Platform and gateway firmware tasks. After the unit description and installation details were specified in the scripts, the administrator enables and start them using `systemctl` command.

For the gateway installation and deployment, the administrator will have to execute the following commands:

```
$ mkdir iot_gateway
$ cd iot_gateway
$ git clone https://lorca.act.uji.es/gitlab/vrykov/thso.gateway
$ cd thso.gateway
$ make
$ ./gateway
```

Or, alternatively, he can employ the `systemd` script as described previously.

3.7 Chapter Summary

Present chapter described the proposed IoT architecture considering device, gateway and platform levels with their insights. The range of possible communication technologies that can be used in tandem with the proposed architecture was also mentioned. Such important parts of the infrastructure as the communication protocol and database technology were introduced to certain extent. All necessary information about its deployment, minimum OS requirements, and installation instructions were provided. Finally, the software and programming languages used during the development phase were listed.

Chapter 4

Project Planning

Contents

4.1 Methodology

HPC&A is a research group that maintains the startup spirit, thus many projects which are being developed use agile methodology. This project is not an exception, therefore during the communication protocol, gateway firmware, IoT platform, and testing agile methodologies were applied.

Preferred methodology for the project development was SCRUM [50]. It forms part of the Agile Method and was created in 1993 by Jeff Sutherland.

Briefly explained, the Scrum Method works according to the following guidelines:

1. A product creator makes a prioritized task list called a product backlog.
2. A Scrum Team is created and a "sprint planning" meeting is called. The team decides on the first priority of the product backlog and decides how to implement those pieces.
3. The team has a specific amount of time called a "sprint" to complete the work – usually two to four weeks.
4. The team meets each day to assess the progress. These meetings are called "daily Scrum".

5. The ScrumMaster supervises and keeps the team focused on its goal to be sure deadlines are met.
6. At the end of the sprint, the work should be ready to show to a customer or project creator to evaluate the progress.
7. The sprint ends with a sprint review and analysis.
8. Then the next sprint begins and the team chooses another part of the product backlog to begin working again.

As was mentioned, the development of the project took place during the COVID-19 quarantine. It was conceived and developed individually by the author. Therefore, he fulfilled the roles of the product creator, ScrumMaster, and a team composed of 1 member.

Every week there was a session during which a state of the project and relative progress were considered. Once a current state was considered and previous progress analyzed, further decisions have been taken for the next steps or corrections to apply if necessary.

4.2 Planning

Every project must have a detailed planning, thus it was created. It described the tasks which should have been carried out to finish the project and time that should have been dedicated to each of them.

The project duration lied in a period of time between the 1st of April 2020 and the 30th of June 2020. During this period a complete development of the project was expected, from the communication protocol to the testing and validation. The work was organized taking into account the general and specific objectives. Though, there were many tasks during the development, for the sake of clarity, in this section will be exposed only workpackages which covered the objectives. For individual tasks and more detailed progress, please, refer to the Appendix A.1.

The development process consisted of the workpackages completed in the following order:

1. IoT infrastructure design
2. IoT Platform development
3. Gateway firmware development
4. Device firmware development
5. Communication protocol development
6. Unit testing of developed components
7. Real-time alerts and devices automation features development

The actual projection of the tasks defined in the previous section is visualized in Figure 4.1.

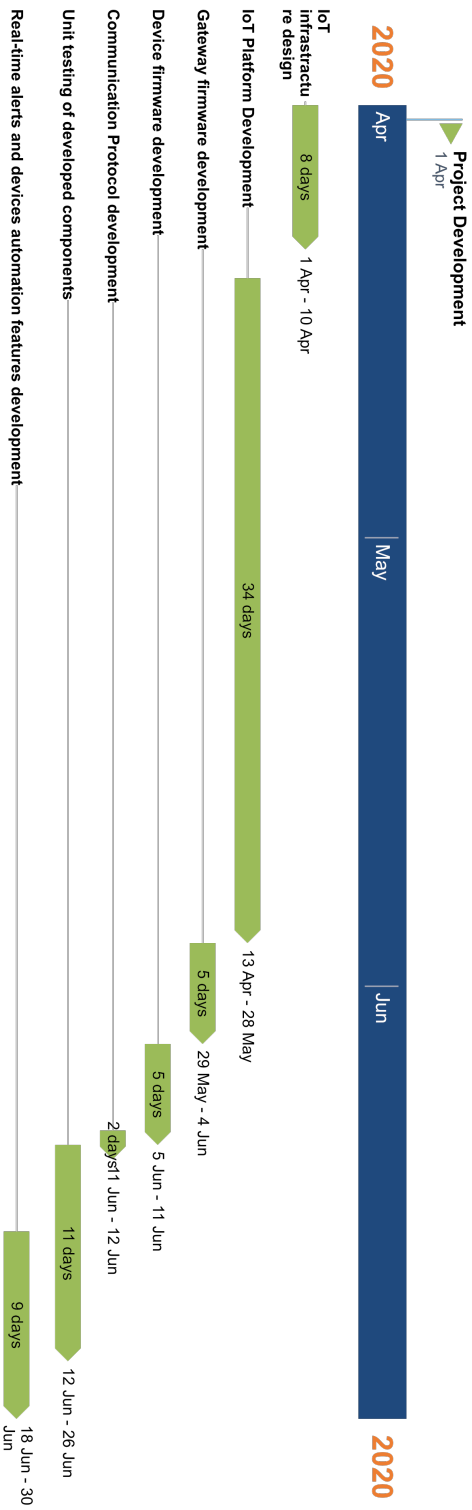


Figure 4.1: Gantt chart for the project development

4.3 Resources and Project's Costs Estimation

The project duration was estimated to be 370 hours which extended over 16 weeks including day offs and weekends, 5 hours per working day. If a cost of the working hour of a software developer can be esteemed around 12€ per hour [57], therefore the human costs will be of 4.440€.

All software used during the project development was open source with no additional expenditures. However, there needed various MCUs and a SoC Linux board for development purposes.

During the device firmware and communication protocol development an ESP32S, Wisen Whisper Node [104] (based on ATMega328p), and Ardunio MKR1000 [58] boards were used. Their costs were 10€, 20€, and 31€ respectively plus additional costs of cables and batteries of 10€ to bring autonomy. On the first phases of the gateway programming a RPi 3 board with the cost of 45€ was used.

The rest of the tools and required elements for the successful project completion were found available on the workplace at the university and summed up no additional costs.

The summary of the project expenses can be found in the Table 4.1.

Table 4.1: Cost summary

Item	
Human costs	Cost (€)
Project development	4440
Device Firmware Development	
ESP32S	10
Wisen Whisper Node	20
Ardunio MKR1000	31
Accessories	10
Gateway Firmware Development	
Raspberry Pi 3	45
Total	4556

This table summarizes all the costs required for the successful project development.

4.4 Project Monitoring

Normally, the projects of this kind and others do not follow initial planning and suffer many changes during development. Fortunately, the present project was not the common case and the development went according to the initially arranged priorities and through the steps sketched during the first working week with tiny deviations.

4.4.1 Sprints

The development was pivoted by 6 sprints. In this section each of those sprints are described.

1st Sprint

During the first sprint the IoT infrastructure design was determined and sketched.

It was important to think carefully about the purpose of the project, and, in the light of the purpose, conceive the right architecture and, consequently, infrastructure. The required mechanisms, capabilities, and limitations were considered.

Not easy point was to opt for the most suitable database technology. SQL and NoSQL paradigms were considered and PostgreSQL selected. Based on the selected database, a user-device interaction mechanisms were engineered.

An important point was to decide a format for device messages. This decision affected the design and a manner in which the data should be retrieved and visualized. It was crucial to give a user freedom of choice minimizing the use of transmitted data.

Finally, the responsibilities of every element in the infrastructure were delimited.

After these preparation steps, actual implementation of the project started.

2nd Sprint

The first part of the second sprint was dedicated to the structure of the IoT platform project. The project structure was decided and a platform configuration scheme implemented. On the other hand it was necessary to decide a URL scheme and a sitemap.

Actual database schema development took place during the 2nd sprint.

The rest of the second sprint was spent implementing DAOs for every database entity.

3rd Sprint

Just after the DAOs were developed, the platform logic implementation has started.

During this sprint, application, device, user management, and device data model were developed. Device data visualization was a struggle since the author had to select data visualization library. The choice was Google Line Chart [37]. Additionally, a lot of new to the author JavaScript features have been learnt.

Finally, the devices remote configuration was implemented to a point where the platform was ready for the next steps in the user-device interaction. However, it would depend upon the communication protocol development which has not been implemented yet.

4^{th} Sprint

During this sprint the platform development continued and more features were added to the platform.

As every platform requires supervision and monitoring, different user privileges and roles were introduced for users and administrators. As applications must be secured from intruders the security mechanisms were implemented. Also log for event monitoring and, most of all, errors, was implemented.

At the end of the fourth sprint the platform has been deployed over uWSGI.

5^{th} Sprint

This sprint was dedicated partially to the gateway firmware implementation and partially to the device firmware.

The gateway implementation started as a simple UDP server and was being developed into a powerful service for network packet managing. The communication protocol was integrated and routines for every function developed. Everything were prepared for further troubleshooting and testing.

During the second part of the sprint a device firmware development took place. Although, it was not a direct part of the project it was necessary for a harmonious infrastructure development and testing. After the sensing, time management, and interruptions routines were developed, the communication protocol development continued.

Right after a simple connectivity with the gateway was achieved, the communication protocol were integrated step by step. During this phase the protocol was debugged and its development finished.

6^{th} Sprint

While project complexity grows, it becomes indispensable to be sure that all functions are still working when new features added. To ensure this kind of reliability during last weeks of the project development the unit testing for every component was developed.

First the tests were developed for the platform, then for the gateway and device firmware, and finally for the communication protocol.

On this step the infrastructure was prepared for thorough testing and validation. Applications, devices, data everything were tested and validated.

However not all planned platform features were implemented yet. Real-time alerts and automation were still not implemented. First, a publish/subscribe functionality of PostgreSQL was explored since upon it based the real-time behaviour.

Once it was achieved, the email alerts and automation was straightforward to implement.

4.5 Chapter Summary

Planning methodology was plainly explained in detail, and how it was applied to the carried out work. The general planning of the work and how the general and specific objectives were achieved in that was presented in a Gantt-chart [16] form. More detailed planning for every specific objective was decided to put into Appendices section. Finally, the required resources costs estimation and detailed sprints monitoring were exposed.

Chapter 5

Communication Protocol

Contents

5.1 Introduction

The term protocol applied for data communications was first applied by K.A.Barlet and R.A.Scantlebury in 1967 [84]. The paper was titled A Protocol for Use in the NPL Data Communications Network and elaborated at the National Physical Laboratory in England.

A protocol can be seen as a mutual agreement about how information is exchanged in a distributed system [42]. A whole protocol is similar to a language definition as it defines:

- strict format for valid messages,
- rules and procedures for data exchange,
- and, formally, a vocabulary of valid messages.

Every distributed system might be created for a precise purpose. If network communications take place for achieving this purpose, then a suitable protocol must be designed to provide

seamless efficient means for the system functioning. IoT infrastructure is a kind of multilayer distributed system.

Due to the multilayer nature of the IoT architecture, there must be formally defined procedures for inter-layer communications. In practice, well-known protocols, such as MQTT, XMPP [83], or CoAP [86] are often used for device-gateway communication. However, custom protocols are also frequently designed to bring more flexibility and efficiency to a concrete IoT infrastructure. Higher layers are used to offer APIs that ensure levels interconnection.

Considering the MQTT and CoAP, there can be found certain issues that can be overperformed by a custom-designed protocol. The MQTT structure requires the inclusion of an additional layer in the protocol stack where the broker resides. This layer adds extra headers and affects the gateway power consumption since more processes should remain in execution. [99] analyzed different scenarios and concluded that increased number of publishers adds more power burden on the device where the broker resides.

Another disadvantage is the use of TCP. As MQTT runs over TCP, it will, on one hand, increase firmware binary file in terms of space. On the other hand, it uses multi-step handshakes before the communication starts what increases device wake-up time. Then, TCP can use persistent sessions for the communication what is not energy efficient. If session persistence is not used, a device has to establish connection every time it communicates with the broker what again decreases device's battery life. Those issues are not present when a UDP custom protocol is designed.

CoAP is running over UDP, but still has issues pertaining to the TCP. It substitutes the mechanisms used in TCP using additional DTLS [54] layer that provides the management mechanisms, but now run over UDP. So, in effect, it comes with the same inconveniences described previously for the MQTT but with less network overhead.

[89] performs an insightful comparison of UDP, TCP, CoAP, and MQTT the network overhead that poses each of them. The comparison summarized in Table 5.1. The columns correspond to the connection setup (Setup), message transmission (Trans), keep-alive signaling (K-A), and connection termination (Term) overhead.

Protocol		Setup		Trans		K-A		Term	
		↑	↓	↑	↓	↑	↓	↑	↓
UDP		-	-	8	0	-	-	-	-
TCP		44	24	20	20	20	20	20	40
CoAP	UDP	16	16	8	0	0	0	8	8
	CoAP	67	32	12	0	0	0	13	8
MQTT	TCP	84	64	20	20	40	40	20	40
	MQTT	34	4	8	0	2	2	0	0

Table 5.1: Minimum network overhead comparison for the most popular IoT protocols [89]

Though the presented material does not directly address the issue of power consumption, yet reasonable arguments can be made on respect. The network overhead is inevitably reflected in the power consumption. It is well-known that communication modules carries the heaviest power burden on the IoT devices. From Table 5.1 can also be seen that a custom protocol

design based on UDP may result in an advantageous alternative to the standard options.

In this solution a custom protocol was designed. The current version of the protocol is quite simple, yet it is scalable and easily extensible for new functionalities.

Additionally, the decision to design a custom protocol was based on the current Internet structure. IoT devices cannot have public static IP addresses, at most, private ones. It makes them unreachable for external networks. Existing solutions, as MQTT, somehow maintain active connections with gateways what greatly degrades their power efficiency. It is well-known how important it is for a device to save power. Thus, with that in mind, a new device-driven protocol was designed that favours devices power efficiency.

The first axiom of the protocol states that all events are initiated by the devices which are supposed to sleep most of the time.

5.2 Implementation Goals and Details

There are several capabilities that must be provided by the infrastructure to support working environment for the IoT applications. First and foremost, communication means should be provisioned. The most important part of them, on logical level, is the communication protocol.

It is highly important for devices to be synchronized and have access to the current time at any moment. The time can be important for application-dependant reasons, i.e. to provide sensor data with precise timestamp or for devices collective synchronization. The data must safely travel from devices to the gateway. Also, there must be mechanisms for communication with devices. On the other hand, communications must be fault tolerant employing acknowledgement mechanism.

The best way to represent network communications is historically through Message Sequence Flow Chart (MSFC). In this chapter, all protocol scenarios are illustrated by MSFCs.

5.2.1 General Packet Format

All packets are strictly shaped according to the packet format definition. The format includes all minimal necessary information for IoT application identification, device identification within the application, and grammar for packet type and content semantics. The general packet form is presented in Figure 5.1.

Application is uniquely identified by 8-bytes field called `app_key`. Thus, theoretically one platform can house up to 2^{64} applications. Devices are uniquely identified by 1-byte field called `dev_id`. Thus, one application can manage up to 256 devices. `packet_type` defines which content carries inside the packet; it is 1-byte field. `packet_length` defines the length of content, 255 bytes maximum. And, finally, `packet_content` represent the very content of the packet.

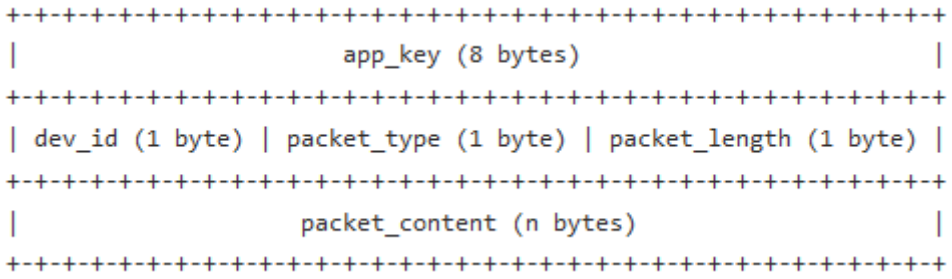


Figure 5.1: General Communication Protocol packet format

5.2.2 Time Request

Time synchronization is one of the vital requisites for many applications. The MSFC for that scenario is pretty simple and shown in Figure 5.2.

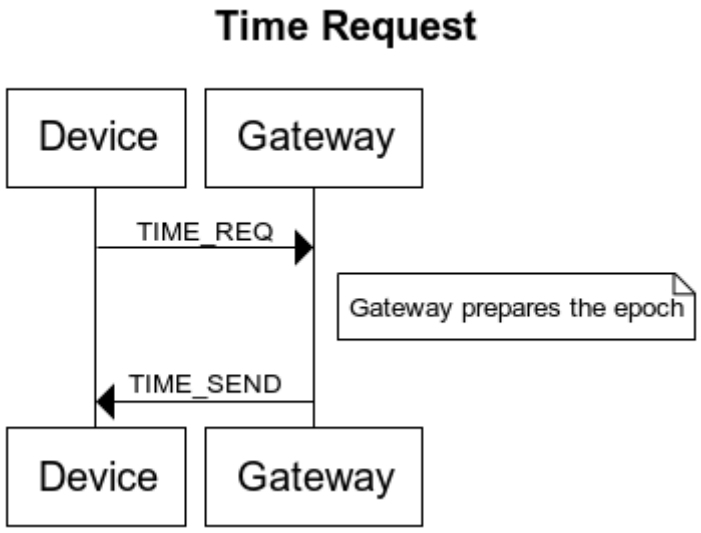


Figure 5.2: MSFC for Time Request event

Device initiates the communication with TIME_REQ packet and gateway responds with TIME_SEND packet that contains current Coordinated Universal Time (UTC).

Next subsections present packet formats for both packet types.

Time request packet formats

TIME_REQ packet has the following format

TIME_SEND packet has the following format

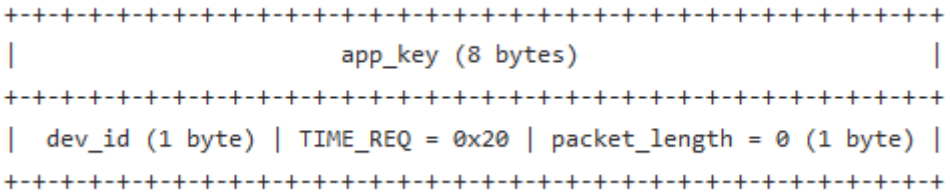


Figure 5.3: TIME_REQ Packet Format

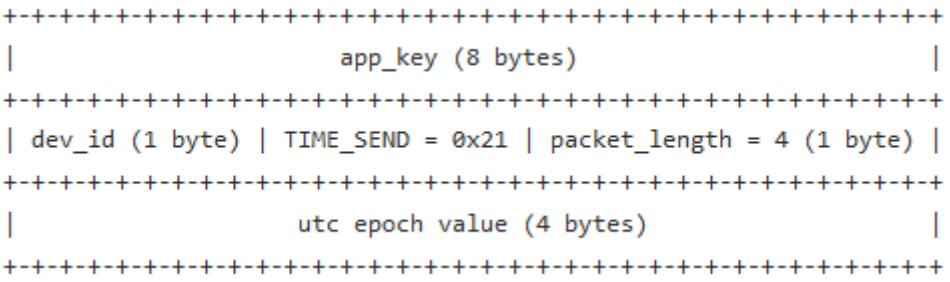


Figure 5.4: TIME_SEND Packet Format

5.2.3 Data Send without Pending Messages

Data from a device to a gateway is moved according to Data Send scenario. Every time data packet is received by the gateway, the data are extracted and inserted into corresponding database table. Then, the gateway queries the database for pending messages for the device. Pending message has a special format as defined in Figure 5.10.

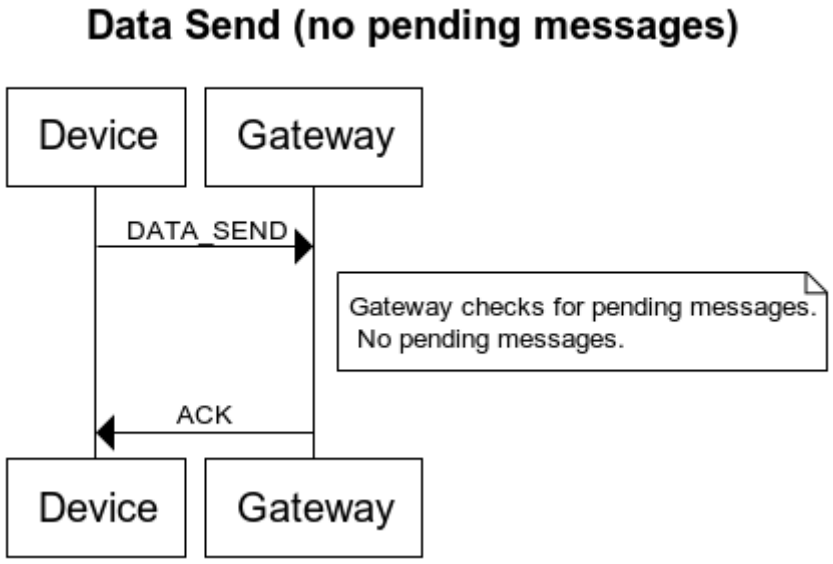


Figure 5.5: MSFC for Data Send without Pending Message event

If the database query indicates that there are no pending messages, then a simple confirmation ACK packet is returned from the gateway as the message acknowledgment.

5.2.4 Data Send with Pending Message

The previous scenario can easily vary depending on pending messages presence. If there is a pending message stored for the device, then the scenario will as illustrated in Figure 5.6.

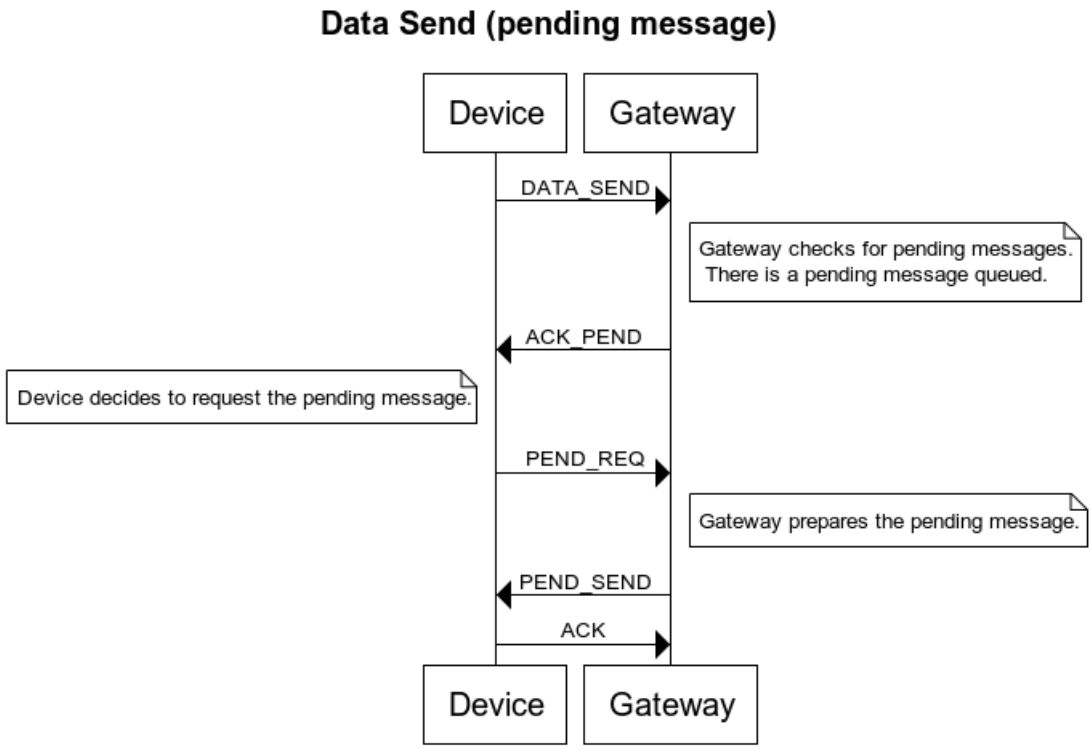


Figure 5.6: MSFC for Data Send with a Pending Message event

After the data is received by the gateway and database query performed, a special acknowledgement packet, the ACK_PEND is sent back to the device. This way the device knows that there is a message waiting for it and can decide the moment when to request it.

When the device decides to request the pending message, it sends the PEND_REQ packet. The gateway queries again the database for the pending messages, picks one in a FIFO fashion and sends it inside the PEND_SEND packet, and waits for the acknowledgement from the device. Once the device received the PEND_SEND packet and processed it, it sends the corresponding acknowledgement.

Data Send packet formats

DATA_SEND packet has the following format.

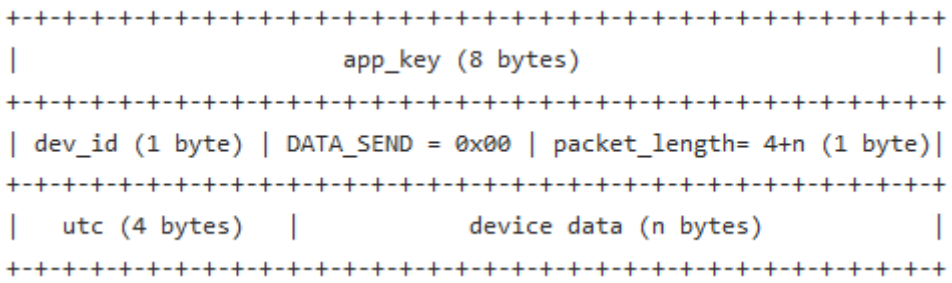


Figure 5.7: DATA_SEND Packet Format

PEND_REQ packet has the following format.

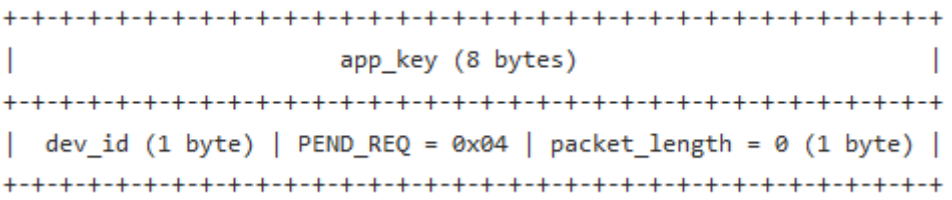


Figure 5.8: PEND_REQ Packet Format

PEND_SEND packet has the following format.

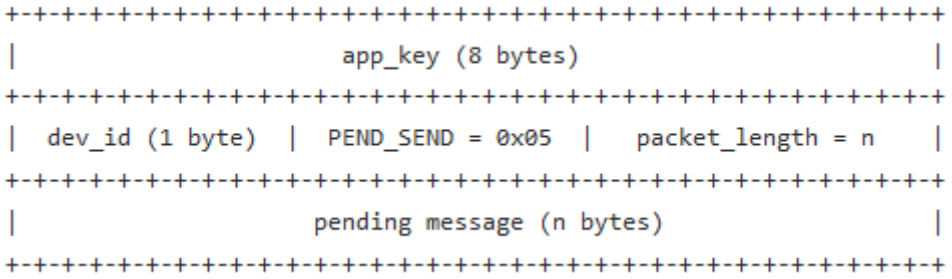


Figure 5.9: PEND_SEND Packet Format

The PEND_SEND payload contains a device control packet which has the format as shown in Figure 5.10.

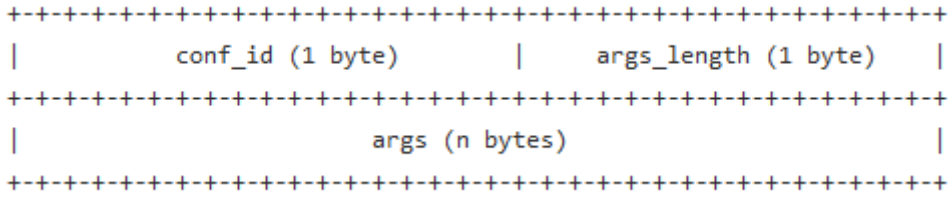


Figure 5.10: PEND_SEND Packet content format, Device Control packet format

STAT packet has the format as shown in Figure 5.11.

Figure 5.11: STAT Packet Format

5.2.5 Check for Pending Messages

A device can periodically check for the pending messages availability, especially to approach real-time behaviour for automation applications.

As usual, a device initiates the communication with PEND_REQ packet. Here, two possible scenarios can take place. If there is a pending message, then everything is identical to the Data Send with pending messages scenarios. Otherwise, a negative acknowledge NACK is sent to the device indicating absence of pending messages. MSFC for this scenario is illustrated in Figure 5.12.

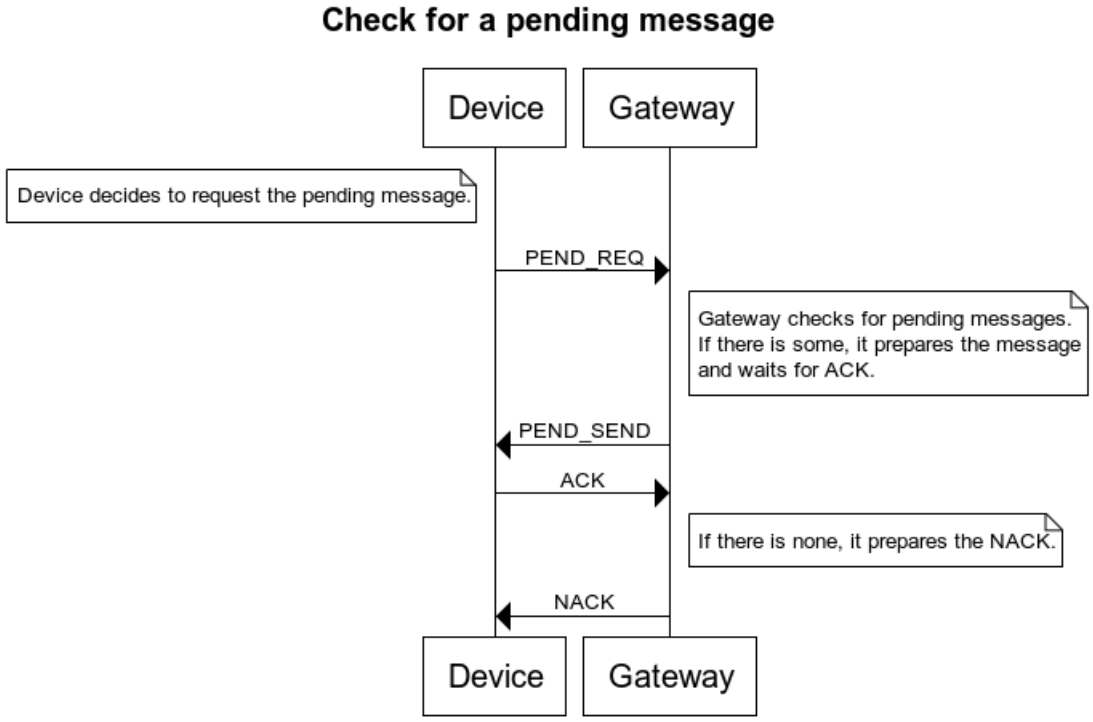


Figure 5.12: MSFC for Check for Pending Message event

5.3 Future Improvements

The proposed protocol is very simple in its nature and functionality. However, it is sufficient to provide the running environment for applications.

Data Send with pending messages scenario can be improved for more efficiency. After the first database query, gateway may directly send the pending messages and device message acknowledgement simultaneously. The downside would be that it obliges the device to handle the message what in some cases may be inconvenient.

`TIME_SEND` packet may also include information about pending messages presence.

One important feature frequently employed in commercial production environments is Firmware-Over-The-Air (FOTA) Update. It is not simple to add this feature, but possible. The protocol must be extended with new packet types for firmware transmission control and device firmware identification. The database scheme will become more complex since the firmware version control will be incorporated. Message integrity checks as Cyclic Redundancy Check (CRC) or checksum will be required for every packet. It is of highest importance to ensure that firmware packet was not damaged and defective bytes or malicious instructions were not written into the device Read Only Memory (ROM).

A great advantage would be to provide mechanisms for better automation, not polling-based as in the current version.

5.4 Chapter Summary

Infrastructure-specific communication protocol brings special advantages to the overall solution. It significantly reduces network overhead compared to its competitors, MQTT and CoAP, and provides mechanisms for end-to-end communication oriented towards energy efficiency. Scenarios that can take place during the IoT applications execution were described along with the involved packet formats. Finally, future improvements were briefly mentioned.

Chapter 6

Gateway Development

Contents

IoT architectures are used to be significantly more complex than typical enterprise infrastructures. The main difference is that the scope of IoT solutions is bigger. In contrast, a data center is just a portion of the architecture. There may be millions of devices being sensing and actuating over their environments and periodically reporting data.

Instead of taking a necessity for presence of the gateway as is, some reasons will be plainly exposed in the following listing.

1. Limited capability of sensor network connectivity.

   Sensors usually transmit information using LoRa, Bluetooth, Zigbee, or any other communication standard which cannot send packets through the Internet. Gateways becomes for them as access points to the Internet.

2. Device packets preprocessing.

   Gateways are not that simple to just forward packets from one network to another, though it can be (i.e. LoRaWAN gateways). That would result in a waste of resources and efficiency losses. More often, they filter packets, take routing decisions if there are more than one database location, and perform aggregation functions.

3. Operation area monitoring.

   Checking for sensors presence and monitoring of an operational area can be done having only one connection point. Communicating periodically with devices, gateway is able to recollect statistics about devices activity. It is not necessary to connect with every device, everything is done on go in one place.

This chapter will unveil gateway firmware architecture and implementation details.

6.1 Firmware Architecture

The gateway and sensors portion of the proposed IoT infrastructure diagram shown in Figure 3.2 is more precisely presented in Figure 6.1.

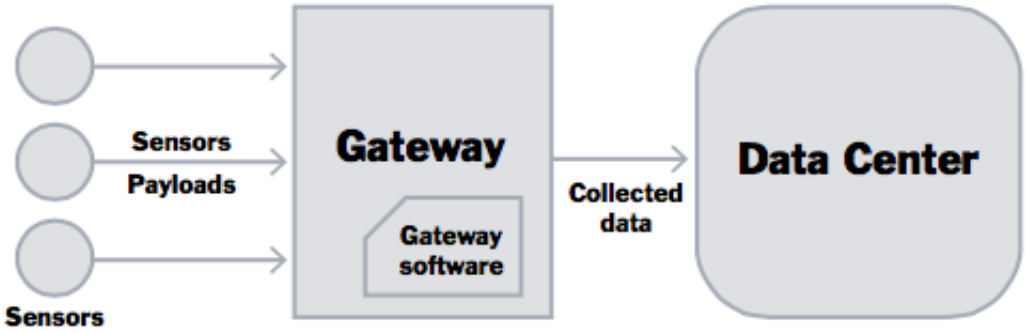


Figure 6.1: IoT Architecture, devices and gateway

Crucial importance has the gateway software referred to as firmware which is responsible for data recollection, preprocessing, and forwarding to the data center, where the database is installed.

For the development failures and, generally, disaster scenarios must be designed recovering strategies. Power down or other situations that may interrupt normal workflow must be foreseen. The firmware must be launched as soon as the gateway is back on.

Logging system also might be designed. Connections with devices, platforms, all error events should be captured and some of them sent to the platform. To find the right balance of which log events to report is useful and important.

Device data normally comprises a set of small values as temperature, humidity, different counters, or other application-dependant values. Sizes for each variable normally lay in the range from 1 to 4 bytes. Thus, devices send tiny messages, normally from 12 to 50 bytes. As the gateway potentially serves many devices, it means that it will have to process a lot of small messages.

Two key characteristics are its capability to understand devices and process as much messages as possible per second. Both are explained in further sections.

Actual implementation of the gateway firmware was designed to run on a Linux-based Operating System (OS). It has an advantage to run over the real-time Linux-kernel (versions greater than 4.0.0). For immediate availability, in case of power down event, the firmware was configured to run as an OS service which is automatically started on system power on, of course when network services are available (System-V runlevels 3, 4, and 5; and systemd multi-user.target and graphical.target).

Firmware structure starts with the initial setup and follows with an infinite loop which is serving devices' requests. Initialization consists of OS signal handler setup which ensures correct functionality after in-loop breakdowns, database connection setup, gateway socket opening and successive port binding. Finally, the thread pull that brings multithreading functionality is created, and a mutex initialized.

The mutex is required to control concurrent accesses to the database. As multiple threads can simultaneously use the database connection, it becomes a critical section. The mutex provides the mechanism for atomic operations over the critical section.

The infinite loop main role is to listen to the predefined by the developer port and receive messages from devices. Once a message is received a new task is dispatched for the message processing. Hopefully, the Unix UDP socket `send` and `recv` functions are atomic by definition and the developer does not have to consider them as critical sections. Otherwise, it would be not possible to listen and respond through the same port simultaneously and concurrent gateway operation could not be realized resulting in huge efficiency losses.

6.2 Gateway Protocol Integration

For each incoming message from a device a new task is enqueued to the alertue, and a new thread is dispatched for its execution. The work that has to be done is the packet processing. Every packet has a strictly defined form and can be of a variety of types. Packet types and corresponding formats along with the protocol scenarios are described in Chapter 5.

The packet processing begins with the packet decoding since the packet, on arrival, represents just a set of bytes without meaning. Packet decoding phase is similar to putting everything it its own box. It extracts from raw bytes protocol fields as `app_key`, `dev_id`, `packet_type`, `payload_length`, and `payload` using standard libraries of the C programming language. Following steps depend on a concrete packet type. Next listing describes actions performed by the gateway depending on the received packet type.

- `TIME_REQ`

  If `TIME_REQ` packet received, the gateway uses `gettimeofday` function from `<time.h>` C standard library to obtain UTC epoch. The epoch represents the number of seconds elapsed from 00:00:00, 1 of January 1970, the moment from which a computer measures system time [102]. It is widely used due to its suitableness. Being a 4-bytes integer, it is easily transmitted and handled by any embedded device. On the other hand, it is the simplest form to leave a timestamp which can be translated later to the human-readable time form on any computer platform.

  Once the epoch is extracted, it is packed as the content of a new `TIME_SEND` packet and sent back to the device.

- `DATA_SEND`

  `DATA_SEND` packet contains the sensor data and a timestamp. It is necessary first to separate the sensor data from the epoch. If a device does not implement time management

functionality and cannot transmit the epoch, then it has to leave this field set to zero. If it is left as zero, the gateway will take packet reception time and consider it as the timestamp.

The next step is the database `INSERT` query preparation. After its successful execution the gateway checks if there are pending messages for the device and depending on that it will respond to the device with `ACK` or `ACK_PEND` packet.

In case of any error a `NACK` packet is sent to the device.

- `PEND_REQ`

  If `PEND_REQ` is received, then a device knows about existent pending message or it is just a polling operation. In any of these cases the gateway queries the database and, if there are no pending messages, it sends back to the device the `NACK` packet. Otherwise, it prepares the `PEND_SEND` packet with the pending message, establishes the response timeout and sends the packet.

  The response timeout is necessary for packet loss events. If there is no timeout set and the `PEND_SEND` packet is lost, then the gateway will fall into a deadlock waiting for the confirmation. In order to avoid this scenario, 300ms timeout is setup what is enough for one transmission over the most of RF technologies. If the timeout shots, then the `PEND_SEND` packet is resent. After 5 retries, the gateway ceases `PEND_SEND` sending operation.

This way the gateway performs packet processing and integrates the entire Communication Protocol.

6.3 Multithreading

The capacity to process as much packets as possible per unit of time is always highly desired for distributed reactive systems. When a program is executed, the OS environment creates a new process with associated memory heap, stack, and other relative attributes. Finally, the process starts to execute developed code using OS-allocated resources.

Process execution is sequential by its nature. It means that every process execute its code line by line. In terms of network connectivity it implies that only one incoming message can be processed at a time. For a single-process gateway the scenario would be as following

- Gateway listens for incoming packets.
- Packet received by the gateway.
- Packet processing started.
- Packet processing finished.
- Response packet prepared.
- Gateway sends the response to the device.

- Gateway listens for incoming packets again.

This kind of implementation has two downsides. First, packets that could have been received during packet processing are lost. Event if OS network buffer can store those packets, if there are multiple devices sending messages, the buffer will be overflown soon and packet loss inevitable. Second, as a consequence of the first downsize, gateway throughput will be significantly low.

Such implementation of gateway would be extremely inefficient and unsuitable for real-time applications and applications with numerous devices. Scalability would be impossible. How this situation could be tackled? Is it possible to receive and process packets, and then respond back simultaneously? The answer is yes.

There are two strategies to achieve it: using processes or using threads. Creation of a new process is a heavy burden for the OS as there are a lot of attributes that must be created and, most of all, a separate memory address space needs to be allocated. Hopefully, in contrast to the processes, threads share the address space with the parent process and are easily created. They are also called the lightweight processes. Moreover, one process can have many associated threads. Thus, the best choice is to use threads.

Multithreading is the best technique to increase throughput of network applications. A thread pool is a strategy to maintain continuous high throughput maintaining certain number of created threads in memory. It was not considered suitable by the author and a decision to create a new strategy was taken.

It is not always required to maintain a certain number of threads in memory. On low demand, zero or one thread would be enough. Therefore, the best choice would be to maintain a dynamic-size thread pull which changes its size depending on demand.

Thread pull library contains a task queue, a linked list where each node is a task to be executed or, in terms of the gateway, a packet to be processed. When a new packet arrives, it is enqueued by the parent process to the task queue and, if possible, passed to the thread pull. The enqueue routine automatically launches a new thread or multiple threads if the maximum amount of allowed threads is not reached and there are tasks waiting to be executed. The new thread takes a pointer to the enqueued task and moves the current task pointer further to the next task to be executed. As soon as the packet is processed and respective response prepared, it is sent back to the device and the thread releases task's and its own resources. This scheme is illustrated in Figure 6.2.

Completed tasks in reality do not exist since when the response is sent back to the device task's resources are released by the thread.

Working with memory allocation and releasing, it is crucial to avoid memory leaks and control that all memory is managed correctly. In the C programming language it is not trivial task and a use of special tools is required. In the present project Valgrind tool suit was used for this purpose. It comprises a number of debugging and profiling tools for improving quality of applications [98]. The most popular tool is called `memcheck` which is designed to detect memory-relative errors as memory leaks, segmentation faults, or unexpected program crashes.

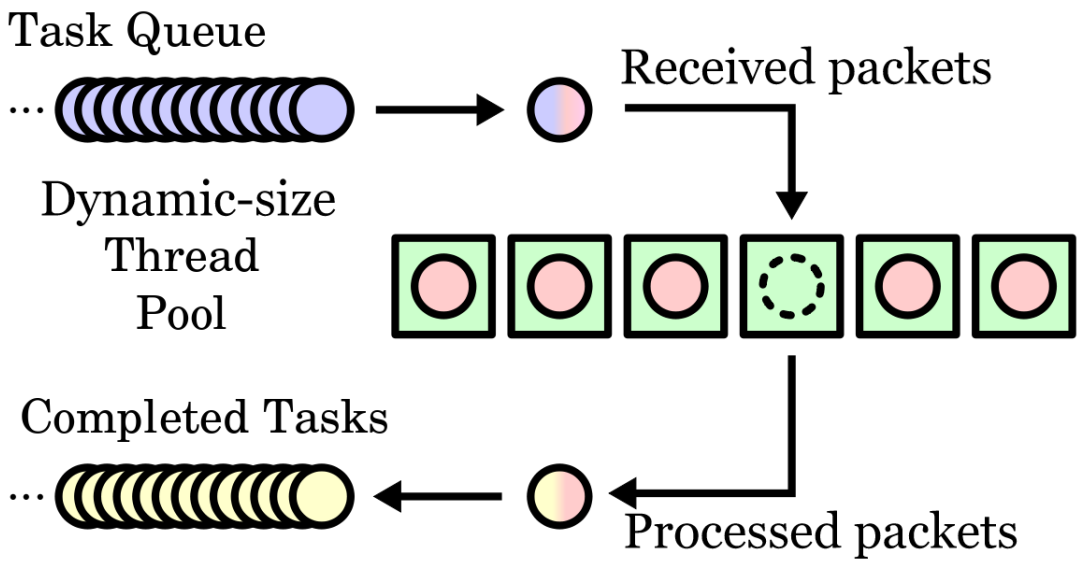


Figure 6.2: Gateway task queue and thread pull

Various kinds to tests were performed and all of them were successfully passed. Tests employed various tasks with different pool sizes, and several task queue maximum sizes.

Also critical demand tests were performed on the gateway, but they will be described in Chapter 8.

6.4 Chapter Summary

The insights into the gateway firmware architecture were given. The way of how were achieved high availability support, device protocol implementation, and high throughput was explained. A point that deserved major importance, mulithreading, was explained separately.

Chapter 7

IoT Platform Development

Contents

One of the most important elements of the infrastructure is the IoT Platform. It is the point of connection for users with their applications, devices, and data. It encompasses many structural as well as front-end and back-end aspects harmonically collaborating to offer intuitive user interface and responsive IoT infrastructure functionality.

Structural aspects cover project structure from file system point of view. Since the platform is considerably complex project, it is important to maintain order and coherency for files and directories in order to make possible easy support for new features and extensions.

Back-end aspects are in charge of internal platform logic. All user requests must be received by a server and processed, and then a consistent response must be constructed and provided back to the user. Internal logic is much closer to the devices and infrastructure than the user

interface. Database connectivity is extremely important in this context. New applications, devices, alarms, automation are created through the use of vital database basic and advanced features.

Front-end aspects have to do with the user interface. It is of crucial importance to make easily comprehensible and intuitive interface. In fact, there is an overwhelming amount of studies about the importance of user experience and how it can improved.

This chapter discusses in detail all mentioned features making arguments for the practices used.

7.1 Project Setup

Before the actual platform implementation has begun, a time was spent to decide the structural facets of the project. They cover a broad scope of characteristics from the file system to the web site.

7.1.1 Structure and Configuration

Depending on a kind of programming language and framework used for development there is always a set of best time-proven practices for project organization. Those practices assure to avoid development process running into issues when the project grows.

The programming language used for the development was Python. Python is a high-level, interpreted, and general-purpose programming language which has strong popularity and is considered the third most popular programming language according to [80].

Within Python arouse many popular frameworks for web applications development as Django [40], TurboGears [81], web2py [26], and Flask. The framework chosen for the present project was Flask due to the ease of deployment and development process. Actually, Flask is a microframework which is based on Werkzeug engine and Jinja2 template engine. It includes development server, unit testing support, RESTful request dispatching and WSGI compliance.

The most convenient way to structure Flask application is using Python package method, that is the application is defined as a package which can be imported as any other Python package. This method allows exceptional flexibility as the application can be split into multiple logical files. As a result, the project structure is clean, easy to navigate and extend.

There are three established directory levels. The next listing describes them in a descending order.

1. External Dependencies and Environment Level

   On this level files and scripts which define the external dependencies as PostgreSQL, libpq and libssl libraries, environment variables, scripts that define initial database structure

are located. Virtual environment folder is recommended to place on this level.

Generally, it encompasses external dependencies on which the application and its core functions are based.

2. Internal Dependencies and Configuration Level

   Inside the previous level there is a folder called `app`. It represents the gate to the next level. On this level the application log file and configuration script are located.

   The configuration script defines settings for development, production, and testing environments. Among the settings database server, email server and other application configuration parameters can be found. This file only defines configurations, but the run script actually apply them depending on the user-defined OS environment variable `FLASK_ENV`.

   The file with internal application requirements is placed here. It greatly eases the installation process since all necessary packages can be installed using only one command.

   On this level is also placed the uWSGI server deployment configuration file. It defines IP address, TCP port, the platform launching script, number of processes and threads used and is capable to define numerous useful features as described in [97].

   Finally, a script which launches the whole platform is also placed on this level.

3. Internal Platform Implementation Level

   This is the core level of the platform. It is separated in 4 parts: static files, dynamic front-end and alert messages templates, views scripts, and additionally required modules as DAOs and decorators.

   Static files are basically JavaScript and CSS files which are required for satisfactory front-end functionality and page rendering. The templates folder is separated into public and administration templates that are dynamically rendered depending on user privileges, applications, or devices. They will be mentioned in subsequent sections. Views scripts represent internal logic of the platform. The HTTP requests are received and processed using functions defined there. The platform URL scheme are also defined in these files. They are also split into administrative and public views. Finally, DAOs are placed in `dao` folder while decorators and helper modules are located in the `helpers` directory.

The described structure is depicted in Figure 7.1.

7.1.2 Use Cases

Use cases are small documents which help to keep clear project objectives and maintain focus on them separately.

There are three levels on which use cases can significantly help. Of course, they are mostly taken advantage of when different teams work on a project. However, they are very helpful even for single developers making clear a project scope and details.

They help during analysis making easy the team communication and converging the understanding of a system for every team member. System requirements are made clear as every use

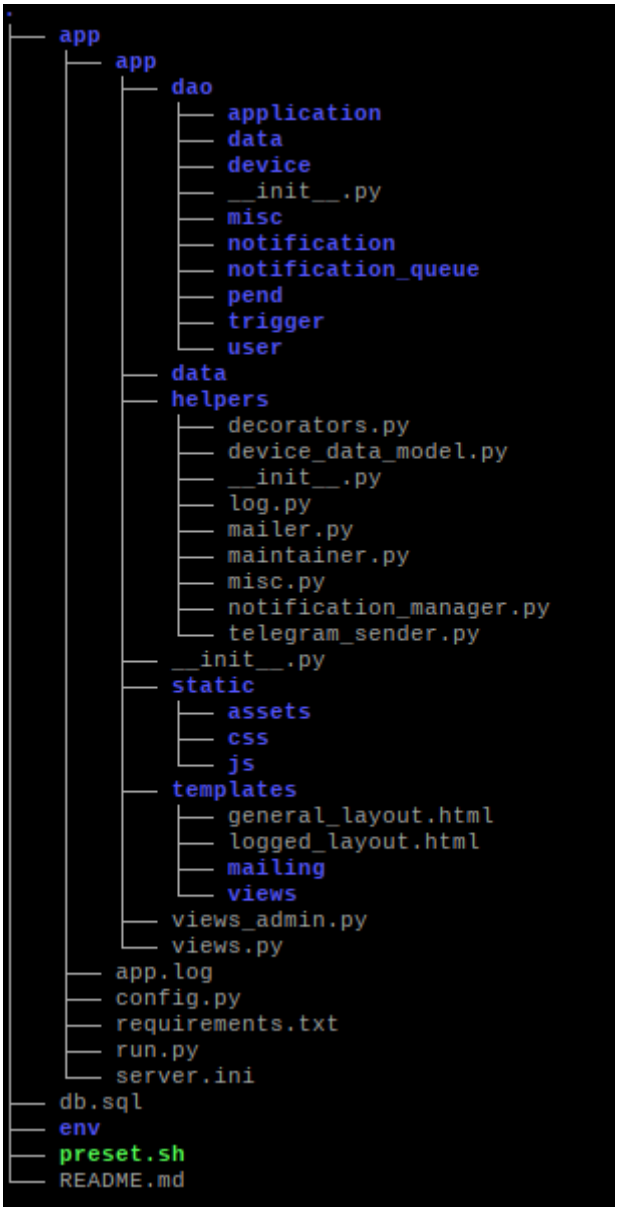


Figure 7.1: IoT Platform project structure (4-level depth)

case is a portion of clarity, consensus, and commitment. Designing use cases unveils possible alternatives, helps to find exceptions beforehand, reveals hidden terms, unnecessary steps, and other issues. It is also remarkable that they help to delimit what enters the project scope and what is outside.

The development team will greatly take advantage of them better understanding the system working processes, having a deeper look into functional requirements recognizing patterns and contexts. And finally, the team will better prioritize the work making the coding phase more efficient.

Testing team will be helped by them seeing more clearly gaps between delivered software

and its requirements. Understanding the system workflow and purpose they will ensure that it works properly faster.

The overall platform purpose in not achieved by a single process, but rather by a set of processes. Each of these processes must have a separate use case. Diagram for all of them is presented in Figure 7.2. Next sections will develop the use cases for every scenario presented in the diagram.

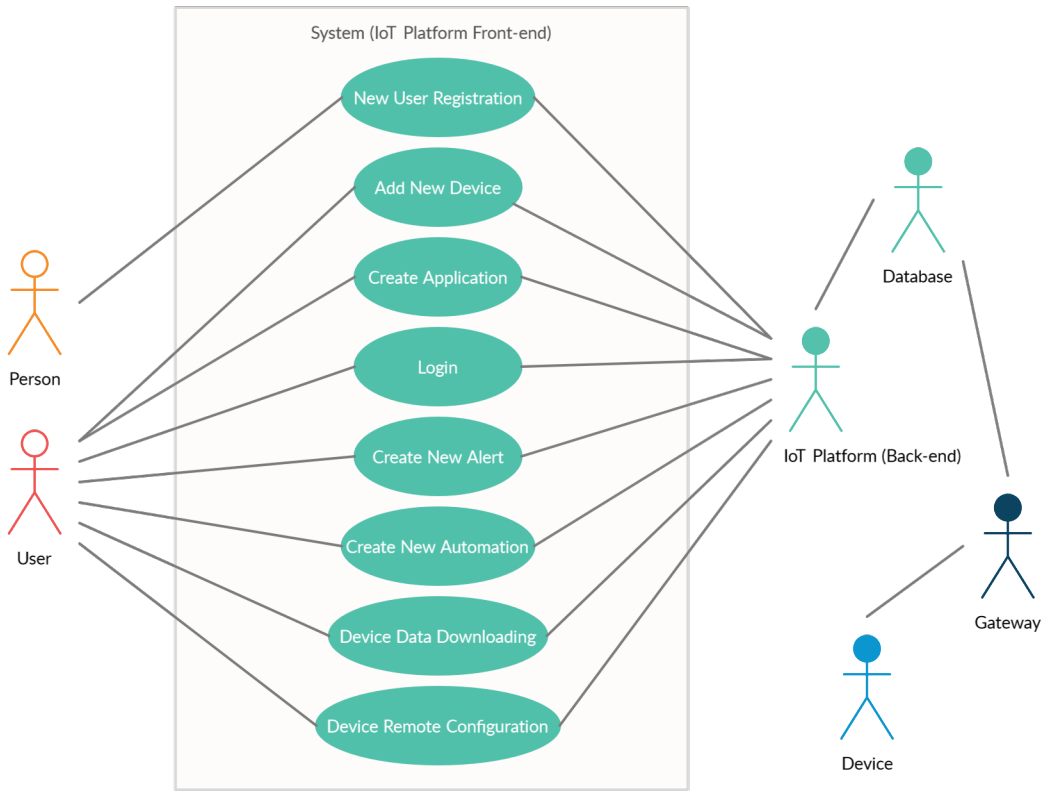


Figure 7.2: Use case diagram for the New Application and Add New Device scenarios

New User Registration

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database

Triggers:

- A person indicates that he wants to create a new account.

Preconditions:

- The administrator configured the platform to allow new users registration.

Post-conditions:

- New user record will be stored in the database.

Normal flow:

1. A person enters the registration page.
2. The person introduces the username which should be unique.
3. The person introduces the password, repeats it for confirmation, and clicks the Create Account button.
4. The user is redirected to the associated dashboard.

Alternate flows:

3A1: If the passwords do not match, the user cannot click the Create Account button. Once they do match, the alternative flow returns to the normal one.

New Application

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database

Triggers:

- The user indicates that he wants to create a new application.

Preconditions:

- The user logged in the system.
- The user has the permission to create applications.

Post-conditions:

- New application record will be stored in the database, and other relative tables and entries created.

Normal flow:

1. The user opens the applications pane.
2. The user clicks the New Application button.
3. The user writes the name, optionally description, indicates if the application will be secure, and clicks Create Application button.
4. The IoT Platform receives the POST request to create a new application.
5. Required fields are checked to be not empty.
6. New database entry is created.
7. New table associated to devices for new application is created.
8. The user is redirected to the applications pane where the new application appears.

Alternate flows:

5A1: If some of the required fields are found empty, the user will be redirected to the New Application view from the point 3, and corresponding feedback message will be shown. From this moment on, the normal flow will be restored.

Add New Device

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database

Triggers:

- The user indicates that he wants to add a new device into an existing application.

Preconditions:

- The user logged in the system.
- The target application has been created beforehand.
- The user has permission add new devices into applications.

Post-conditions:

- New device record will be stored in the database, and other relative tables and entries created.

Normal flow:

1. The user opens the Applications view.
2. The user clicks for details about an existing application.
3. The user clicks the Add Device button.
4. The user writes the name, optionally description, indicates the device id, registers the device data model and clicks Add Device button.
5. The IoT Platform receives the POST request to add a new device.
6. Required fields are checked to be not empty and the device id to be not used.
7. New database entry is inserted into the devices list table.
8. New table associated to the device data is created.
9. The user is redirected to the Application view where the new device appears.

Alternate flows:

6A1: If some of the required fields are found empty, the user will be redirected to the Add Device view from the point 3 and corresponding feedback message will be shown. From this moment on, the normal flow will be restored.

6A2: If the introduced device id was occupied, the user will be redirected to the Add Device view from the point 4, and corresponding feedback message will be shown. From this moment on, the normal flow will be restored. Note that the range of unoccupied device ids was indicated right above the device id input field.

Login

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database

Triggers:

- A user indicates that he wants to log into the platform.

Preconditions:

- The user had created an account or received credentials for a created for him account.

Post-conditions:

- New user logged into the platform and is able to explore its functionality.

Normal flow:

1. A user enters the IoT Platform login page.
2. The user introduces the username and password.
3. The user is redirected to the asscialted dashboard.

Alternate flows:

2A1: If the username or password are not correct or do not match, the user will not be allowed to log into the platform and will be redirected to the login page with the corresponding feedback provided. Once the user enters the right combination of his username and password, the alternative flow returns to the normal one.

Create New Alert

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database

Triggers:

- A user indicates that he wants to create a new alert.

Preconditions:

- The user logged into the platform.
- The user had created an application.
- The user had added a device that will trigger the alert to the application.
- The user has permission to create alerts.

Post-conditions:

- New alert identification record will be stored in the database.
- New associated to the alert database function will be created.
- New trigger will be created, the function will be bound to the trigger, and the trigger will be linked to the device table.

Normal flow:

1. The user opens the Application view.
2. The user clicks Alerts button.
3. The user, in appeared Alerts view, clicks New Alert button.
4. The user introduces new alert name.
5. The user selects the device that will be associated with the alert

6. The user selects the device variable that will trigger the alert.
7. The user selects a conditional operator for the variable
8. The user introduces a value that will be used with the selected conditional operator against the selected device variable.
9. The user introduces the email to which the alert should be sent.
10. The user clicks Create Alert button.

Alternate flows:

4A1: If the alert name field has been left empty, the user will not be able to click Create Alert button.

5A1: If the device has not been selected, the user will not be able to click Create Alert button.

6A1: If the device variable has not been selected, the user will not be able to click Create Alert button.

8A1: If the value has not been introduced, the user will not be able to click Create Alert button.

9A1: If the email has not been introduced, the user will not be able to click Create Alert button.

Unless all fields have been filled and options selected, the Create Alert button will not be activated.

Create New Automation

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database

Triggers:

- A user indicates that he wants to create a new automation.

Preconditions:

- The user logged into the platform.
- The user had created an application.
- The user had added a device or devices that will constitute automation.
- The user has permission to create automation.

Post-conditions:

- New automation identification record will be stored in the database.
- New associated to the automation database function will be created.
- New trigger will be created, the function will be bound to the trigger, and the trigger will be linked to the device table.

Normal flow:

1. The user opens the Application view.
2. The user clicks Automation button.
3. The user, in appeared Automation view, clicks New Automation button.
4. The user introduces new automation name.
5. The user selects the device that will be associated with the automation trigger.
6. The user selects the device variable that will trigger the automation.
7. The user selects a conditional operator for the variable.
8. The user introduces a value that will be used with the selected conditional operator against the selected device variable.
9. The user selects the device that will be associated with the automation target.
10. The user introduces the Configuration ID and arguments related to the message that will be sent to the automation target device.
11. The user clicks Create Automation button.

Alternate flows:

4A1: If the automation name field has been left empty, the user will not be able to click Create Automation button.

5A1: If the device has not been selected, the user will not be able to click Create Automation button.

6A1: If the device variable has not been selected, the user will not be able to click Create Automation button.

8A1: If the value has not been introduced, the user will not be able to click Create Automation button.

9A1: If the target automation device has not been selected, the user will not be able to click Create Automation button.

10A1: If the Configuration ID and argument fields have been left empty, the user will not be able to click Create Automation button.

Unless all fields have been filled and options selected, the Create Automation button will not be activated.

Device Data Downloading

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database

Triggers:

- The user indicates that he wants to download device data.

Preconditions:

- The user logged into the platform.
- The user had created an application.
- The user had added a device which data will be downloaded.

Post-conditions:

- The user has a CSV file with the data sent by the device, if any, stored in his operating device.

Normal flow:

1. The user opens the Applications view.
2. The user clicks the Details button corresponding to the application of interest.
3. In the Application view, the user clicks on the card associated with the device of interest.
4. In the Device view, the user clicks the Download CSV button, then, the server responds to the user CSV file with the device data attached.
5. After the file transmission has been finished the user has the CSV file in his operating machine.

Device Remote Configuration

Actors:

- IoT Platform User
- IoT Platform (Back-end part)
- Database
- Gateway
- IoT Device

Triggers:

- A user indicates that he wants to configure or change configuration of an IoT device.

Preconditions:

- The user logged into the system.
- The user had created an application.
- The user had added a device that will be configured.
- The device is appropriately programmed and consistently follows the Communication Protocol scenarios.
- Provided transmission medium is of high quality such as errors consideration can be omitted.

Post-conditions:

- The device will change the intended configuration.

Normal flow:

1. The user opens the Applications view.
2. The user clicks the Details button corresponding to the application of interest.
3. In the application view, the user clicks on the card associated with the device of interest.
4. In the Device view, the user clicks the Configure button.
5. In the Configure view, the user fills the Configuration ID and argument fields.
6. The user clicks the Configure button.
7. Next, device sends `DATA_SEND` packet to the gateway.
8. The gateway inserts the device data to the database and queries for pending messages for the device.
9. The gateway finds the pending message and prepares `ACK_PEND` packet.
10. The device receives `ACK_PEND` packet and sends `PEND_REQ` packet to the gateway.
11. The gateway responds to the device with `PEND_SEND` packet that contains device configuration information (Configuration ID and argument).
12. The device receives the `PEND_SEND` packet, processes it, changes the configuration, and sends `ACK` packet to the gateway.

Alternate flows:

5A1: If the Configuration ID and argument fields have been left empty, the user will not be able to click Configure button. Unless all fields have been filled and options selected, the Configure button will not be activated.

7.1.3 Platform as a Web Site

Internet platform is defined as a set of technologies that are used as a foundation upon which other applications, processes or technologies are developed [93]. Platform is a place where one party is in touch with another, i.e. users are in touch with their devices.

Apart from all mentioned above, online platform is also a web site. According to the best practices, it must provide a coherent and intuitive web interface and employ consistent URL scheme. Another aspect is a sitemap which is, simply put, a list of pages of a web site.

URL Scheme

The World Wide Web exists about 30 years already. There has been written a lot about URL and especially Universal Resource Identifier (URI) design. Tim Berners-Lee and Roy Fielding have written various guidelines for their design. The former wrote the design Axioms [9] which focus on sameness and identity issues. On the other hand, once decided the URL scheme must be permanently the same since it cannot be known who already has bookmarked a page or if it has been scrawled. When a link is followed by a potential user and it turned out that it was broken, human mind tends to lose confidence in the owners of the web site, and the user frustration is inevitable.

URL scheme it is not only the design aspect, it has to do with the web site reputation and is a point of user confidence and emotional satisfaction.

Roy Fielding was the creator of REST philosophy, and currently the best practices for URL design follow his ideas. The basic rule allows using nouns and verbs which identify actions, objects, and places.

```
/application/create - is used for Create Application view
/application/<appkey> - is used for Application view
/application/<appkey>/settings - is used for Application Settings view
/application/<appkey>/device/<devid> - is used for Device view
```

The complete URL scheme is presented in the next section.

Another important rule is to use HTTP methods as GET, PUT, POST, and so on, matching them with intended actions nature. Next listing explains the the meaning for the used HTTP methods.

GET is used to get resource as application or device.

PUT is used to update resource.

POST is used to create resource.

DELETE is used to delete resource.

Thus, the combination of consistent URL scheme along with accurately used in a REST-manner HTTP methods are combined into clear, scalable, and transparent web interface.

Sitemap

The sitemap designed for the present project according to the described in the previous section rules is exposed in the following listing.

```
Scheme for all users
```

```
/
/register
/login
/logout
/chart-update
/recent-activity
/settings
/account/delete
/applications
/application/create
/application/<appkey>
/application/<appkey>/settings
/application/<appkey>/delete
/application/<appkey>/device/add
/application/<appkey>/device/<devid>
/application/<appkey>/device/<devid>/configuration
/application/<appkey>/device/<devid>/configuration/delete
/application/<appkey>/device/<devid>/variables
/application/<appkey>/device/<devid>/download-csv
/application/<appkey>/device/<devid>/settings
/application/<appkey>/device/<devid>/delete
/application/<appkey>/alerts
/application/<appkey>/alert/create
/application/<appkey>/alert/delete
/application/<appkey>/automation
/application/<appkey>/automation/create
/application/<appkey>/automation/delete

Scheme for administrators

/administration
/administration/users
/administration/<username>
/administration/<username>/chart-update
/administration/<username>/recent-activity
/administration/<username>/settings
/administration/<username>/account/delete
/administration/<username>/applications
/administration/<username>/application/create
/administration/<username>/application/<appkey>
/administration/<username>/application/<appkey>/settings
/administration/<username>/application/<appkey>/delete
/administration/<username>/application/<appkey>/device/add
/administration/<username>/application/<appkey>/device/<devid>
/administration/<username>/application/<appkey>/device/<devid>
    ↪   /configuration
/administration/<username>/application/<appkey>/device/<devid>
    ↪   /configuration/delete
/administration/<username>/application/<appkey>/device/<devid>
```

```
    ↪   /variables
/administration/<username>/application/<appkey>/device/<devid>
    ↪   /download-csv
/administration/<username>/application/<appkey>/device/<devid>
    ↪   /settings
/administration/<username>/application/<appkey>/device/<devid>
    ↪   /delete
/administration/<username>/application/<appkey>/alerts
/administration/<username>/application/<appkey>/alert/create
/administration/<username>/application/<appkey>/alert/delete
/administration/<username>/application/<appkey>/automation
/administration/<username>/application/<appkey>/automation/create
/administration/<username>/application/<appkey>/automation/delete
```

This section shed light on the time-proven web-design practices used during the development of the IoT Platform web site structure.

7.2 Database Integration

Every Internet platform must have a place where to store its data. There are many storage architectures implying various databases where each architecture corresponds to a concrete solution type, for instance enterprise platform or online marketplace. However, when it turns to the IoT architecture there is no a concrete recipe. It rather depends upon the purpose and objectives that establish particular requirements.

Be it as it may, when a developer writes the code, the database-related aspects must be decided and issues solved. Often web frameworks already offer a concrete set of databases integrated. Flask is an exception, it does not originally include databases, and it was done intentionally. A complete freedom of choice about which database to use is given up to the developer.

As was described in Section 3.4 the preferred database selected for the project was PostgreSQL. The next step in the development was to search a high-quality, well-documented, and continuously-maintained PostgreSQL adapter for Python programming language. It was not a long search as there is only one the most popular open source adapter called `psycopg` [25].

`psycopg` was written mostly in C programming language and in its essence represents a wrapper around `libpq` library used in the gateway development. `psycopg` offers support for all Python versions, old and new ones, up to 3.8 and supports PostgreSQL versions from 7.4 to 12. It is thread-save, asynchronous and non-blocking and adapts database datatypes to many Python objects as tuples, lists, and dictionaries. There are many other features that make this adapter to be outstanding candidate for many Python applications with the database integration based on PostgreSQL.

7.2.1 Database Schema

In order to provide a deep insight into the architecture functionality it is required to make clear the database role. The role of database is better explained when its schema was described and understood. This section introduces the database schema and explains its design reasons.

Once the IoT Platform has been installed, there is an initial set of tables required for its start up. Among those tables are `users`, `applications`, `notifications`, and `pend_msg`. The rest of the tables related to the devices and their data will be added and deleted dynamically. The database schema is depicted in Figure 7.3.

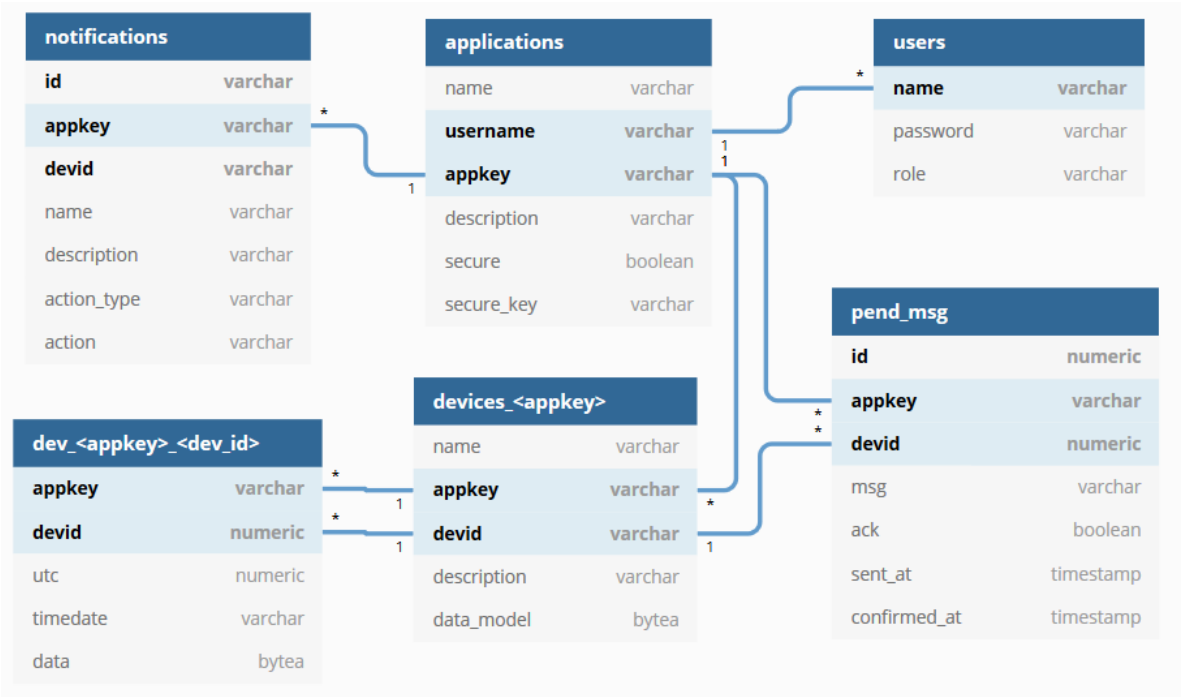


Figure 7.3: Database schema for the IoT Platform

As a generic example tables `device_<appkey>` and `dev_<appkey>_<devid>` were added to store a list of devices for a specific application and to store data of a particular device which belong to a particular application respectively. In practice, in order to avoid duplicating information columns `appkey` and `devid` are not present in the tables `device_<appkey>` and `dev_<appkey>_<devid>` and were shown only to demonstrate logical links. For their referring concrete values of `appkey` and `devid` were used in the table names.

On user creation one new recorded is added to the table users. The passwords are saved in its SHA256-hashed forms with adding a block with salt. This is a standard methods for storing the passwords which protects them against rainbow lookup tables.

When the user creates new application, apart from adding one record into the `applications` table, a new table to store the list of devices for that application is also created. The table to store a particular device data is created akin, that is, when new device is created.

When the user decides to remove, a particular device, application, or personal account, those tables are dropped in a cascade manner.

7.2.2 Data Access Objects (DAO)

Software engineers tend to face similar kind of problems and solve them in similar ways. Some experienced developers noticed this fact and tried to extract those problems with their contexts and solutions. General solutions for such problems were called Software Design Patterns. It is also referred to as formalized best practices [103] using which a developer is able to solve common problems designing software.

There are various design patterns related to different aspects of software development: creational, structural, behavioral, functional, concurrency, and architectural.

There are several software-design problems related to the database access. It is always necessary to maintain coherency between an object in computer memory during application execution and one stored in the database. On the other hand, database technology can change with time and it may imply rewriting significant part of the application. Both problems can be solved applying the DAO software pattern.

It is the architectural pattern that allows to access persistent storage for maintaining data coherency, to separate a concrete database adapter interface from application data access mechanisms, and to make generic database client interface valid for any persistent data source allowing larger teams to work on different parts of a project having clearly separated concerns.

This section will present an example of this pattern applied to an IoT device. Structure for this pattern applied to the present project is shown in Figure 7.4.

This way the data access, coherence, and future database technologies are supported.

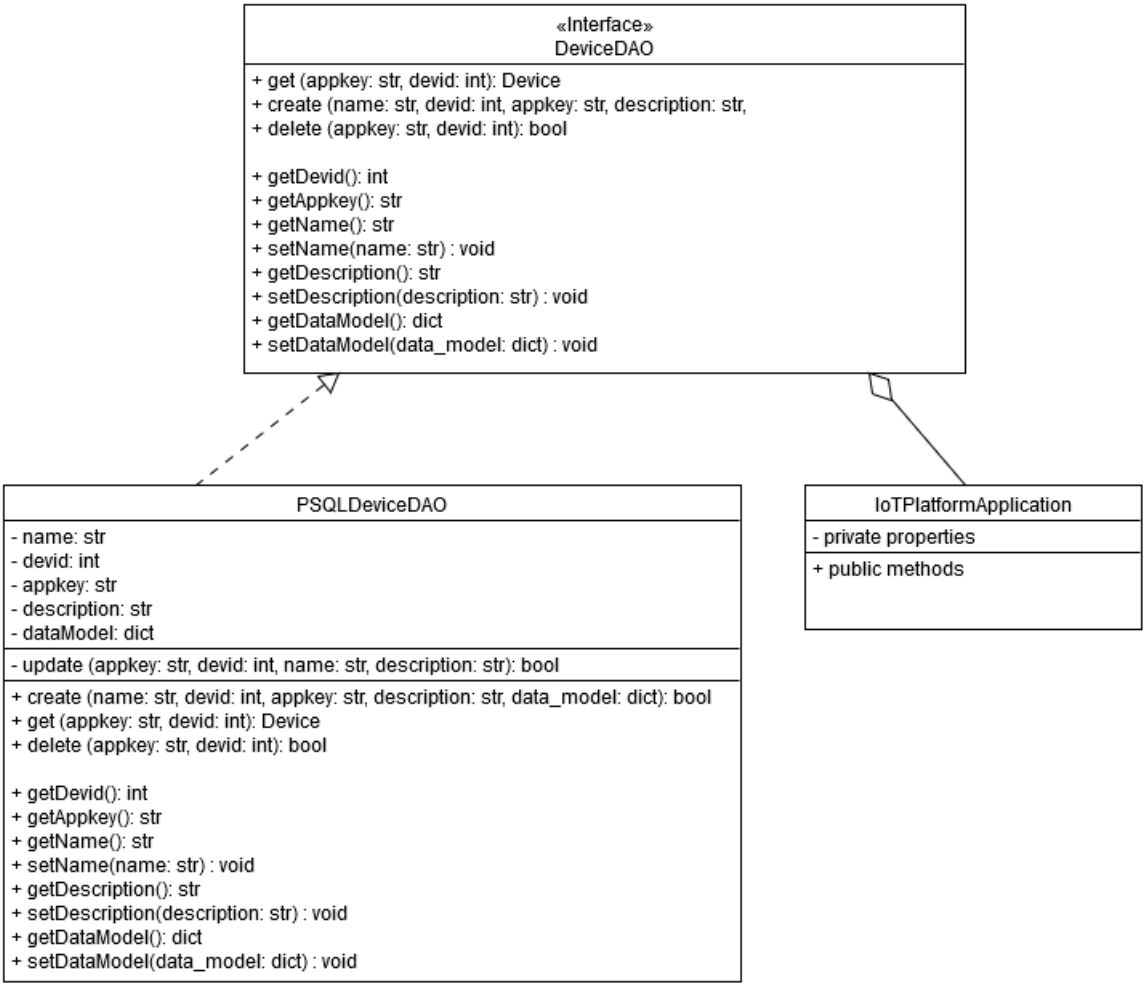


Figure 7.4: DAO software design pattern applied to the present database scheme

7.3 Platform Internal Logic

The infrastructure presented Figure 3.2, though is a practical implementation of the architecture presented in Figure 3.1 it lacks mechanisms that make it work as a whole. The core element that sustains the user information about created applications and devices and provide mechanisms for the communication with devices was introduced in Section 3.4 and deeper explained in Section 7.2.1. However, there is a set of tasks that carries out internal platform logic and which were not described yet.

There are procedures and crucial details that make the platform work and which are implementation-specific. This section will address those features and provide information about them to a sufficient extent.

7.3.1 Application Management

Application is a basic platform unit that moves the gears of the infrastructure and makes it work. Without it will be no devices, no data, neither alerts, nor automation.

The most important attribute of an application is the `appkey`. It is an 8-byte string which along with a user name uniquely identifies every application. It is generated using a pseudo-random number generator that provides one of the standard Python libraries.

When a user creates alerts or automation they are stored referring the application for which they were created.

An application can be secure or insecure. It means that the user can choose to send ciphered payload or transmit data using plain bytes. This functionality is governed by `secure` attribute stored in the associated database record. It must be mentioned also, that independently of user choice to create secure or insecure application, the key for AES-128 (CTR mode) ciphering algorithm is generated and stored in the `secure_key` application attribute.

Application security is another important concern that deserves special attention. As was mentioned, the encryption algorithm used is AES-128-CTR which is considered secure and used in IoT communication standards such LoRaWAN [95], [1] and ZigBee [74].

When a gateway receives a messages from a device, it checks whether it belongs to a secure application. If it does not, payload is sent as is to the database, otherwise the `secure_key` is extracted and the payload deciphered, after what it is sent to the database. It is highly important that the connection with the database is carried out only using TLS or SSL protocols. It must be explicitly indicated in the database configuration file.

The key sharing problem is solved in a simple way hard-coding the `secure_key` during the device programming. For better security and against reverse engineering schemes the firmware can be secured using Flash Encryption feature similar to ESP32 [28] or STM32 [3].

7.3.2 Device Management

Application without devices made void since the devices is what brings senses and action to an application physical environment. This section is dedicated to the device management aspects on platform and architectural levels.

The most important attributes of a device are `appkey` that identify the application to which a device belongs and `devid` that uniquely identify the device within the application.

For the user convenience the selection of `devid` was left up to the user. Similar to networking addressing schemes, the user can exercise possible id assignment strategies to segment the application logically.

This section will uncover such aspects as device data model, data visualization, download,

and remote configuration.

Device Data Model

All devices are designed to send and/or receive data. Data comprises a set of variables stored in memory using a certain format. Further, every variable has its own name and type. How devices should send data to gateways that the platform can understand and visualize them? This question introduces the problem of the device data model.

There are many protocols using which data can be sent, but all of them can be divided in two groups: string-based and binary-based. Depending on a case one can be more efficient than the other in terms of size. Examples of string-based protocols are HTTP [34], UltraLight 2.0 used by FIWARE [29], or Ubidots protocol for TCP/UDP communications [27]. Undoubted advantage of string-based protocols is the cross-platform portability, that is, it does not matter a machine architecture, string representation is the same on every machine and defined by ASCII table [91]. However this representation is not always efficient as can be binary-based protocols.

Examples of the binary-based protocols are TCP/UDP, IP, and Datagram congestion control protocol (DCCP) [52] since those protocols packets are treated as binary information. They treat a packet payload as raw bytes and employ completely different decodification (parsing) methods to the string-based protocols. Of course, strings can be transmitted over such protocols because their final instance consists of bytes as well however it does not affect how they process packets. String-based protocols cannot carry binary payload, it must be encoded into base64 format for example.

Thus, sensor data can be expressed in string or binary formats. The most popular example of string based data format is JSON. There are other formats, but they are not universal and usually depend on a particular protocol implementation as [29] and [27]. On the other hand, sensor data can be expressed as bytes with specified types, i.e. IEEE 754 for Floating-point Arithmetic [51]. But it must be noticed that binary data are subject to architecture representation and architecture-related information must be known for their correct decodification.

In order to adapt for user needs and provide certain flexibility, the IoT Platform offers three possible data formats: JSON, MessagePack [35], and binary.

JSON format is a simple string-based dictionary-like format with keys and values where keys correspond to variable names. MessagePack is an efficient binary format inspired by JSON used for data serialization. It was enabled due to its efficiency compared to JSON. It is faster and smaller. For example, small integers are encoded as a single byte, and strings require only one additional byte. Binary format does not require any explanation. A simple efficiency comparison of all three data formats is shown in Figure 7.5.

It can be observed how strong the difference is. Using binary format instead of JSON in this example optimizes data compression efficiency up to 13 times what correspondingly reduces mobile data transmission costs if GPRS communication takes place.

When the user adds new device to an application, respective device data model must be

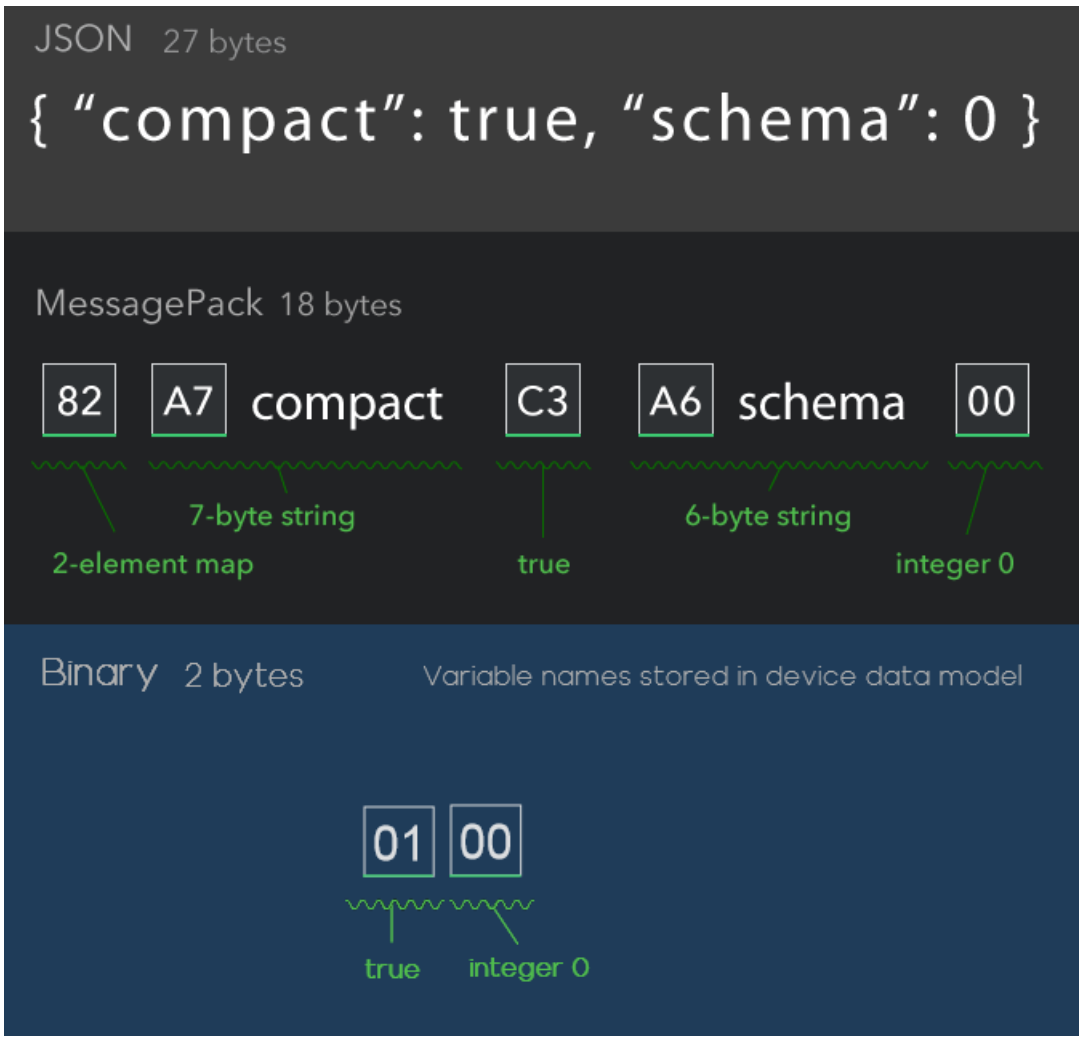


Figure 7.5: JSON, MessagePack, and binary formats efficiency comparison

specified. If JSON or MessagePack format is specified, then the user must to introduce variable names and types. Both formats hold relatively simple data formats which can be a number, string, or boolean. In contrast, binary format requires more exhaustive specification. First of all, the user must introduce the architecture endianness (little or big endian). Then, similar to JSON and MessagePack, the user introduces variable names and types. Available types are float, boolean, 1-,2-,4-,8-byte signed and unsigned integers, and string. String size must be explicitly introduced.

The IoT Platform will use this information when device data decodification is required. For instance, when the user wants to download CSV file with the device data, visualize them, or alert/automation condition needs to be checked, special data decodification routines will be applied.

If the binary format is selected, then it is important to preserve variables order as they were introduced during the device addition. It is also always possible to edit variables order or change data model in the device settings view.

Device Data Visualization

The presentation of data in graphical format brings special insights to users, be they business leaders or students. It enables rapidly analyse the visual information, understand difficult patterns and nature of the processes or objects under consideration. Moreover, if the visualization is interactive, concrete values observed at certain moments can be seen in detail.

As the principal tool for visualizing data on the IoT Platform Google Charts was selected. It is powerful, highly customizable, simple, and free. Google Charts is designed as JavaScript classes and rendered using HTML5/SVG technology. Web-based rendering tools allows its cross-browser and device presence. There are no required plugins.

Typical device data chart is visualized in Figure 7.6.

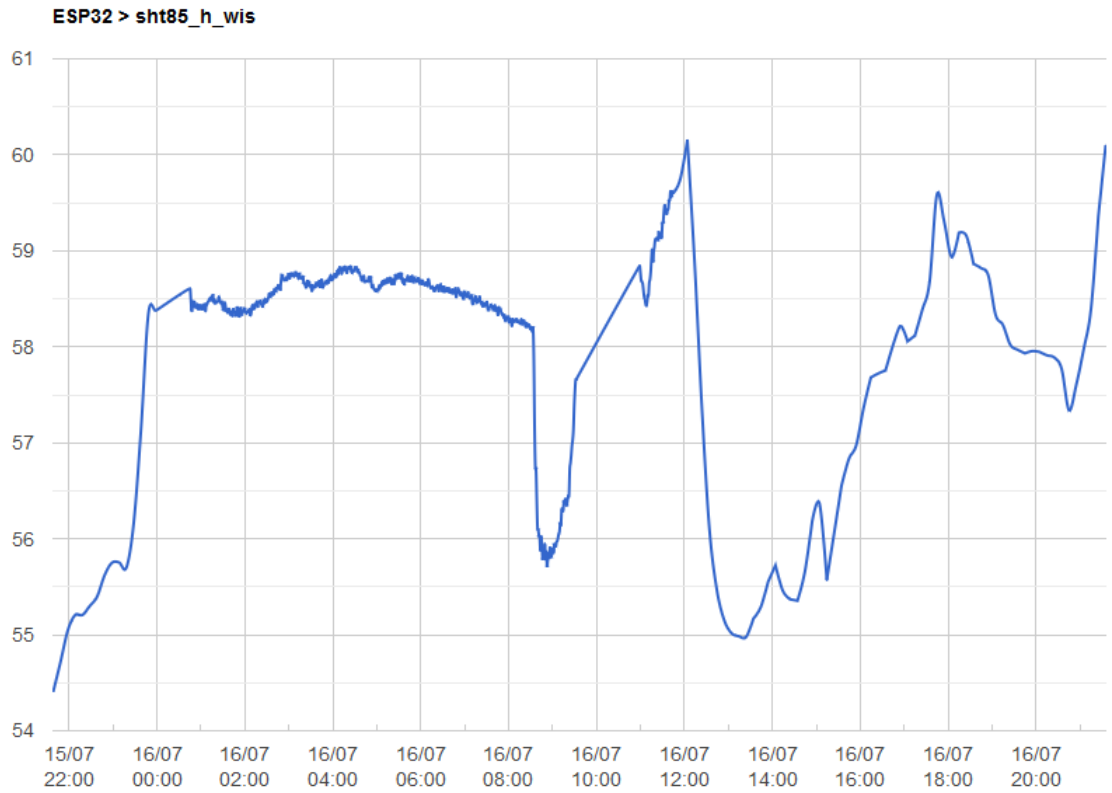


Figure 7.6: Device data visualization based on Google Charts. Humidity data from SHT85 sensor.

The Device view includes tabs which correspond to device variables. When the user click on a tab, the variable data is dynamically loaded from the server and nicely visualized. The chart shows data for last 24 hours. Additionally, under the chart, a table with timestamps and values is presented. Moreover, it can be extended upon user request. This way the user can observe device's behaviour and easily explore concrete values. The data visualized in Figure 7.6 were borrowed from the real research application that aims to compare behavior of many temperature/humidity sensors.

Device Data Download

The IoT Platform allows the user to download data as a file of CSV format. Since the platform has limited capabilities and does not offer special AI and Machine Learning (ML) involvement, this option opens the door for special user-enabled applications.

The platform uses device data model to decode binary data stored in the database. Once the user decided to download device data, a new file is created, and, line by line, it is filled with the device data. CSV file is then sent attached to the user response. After the request is served, the file is immediately deleted.

Device Remote Configuration

Among the most important features of the presented infrastructure is its capability to communicate with IoT devices. Reader might remember that all communications are initiated by the client. Thus, a strategy for fitting this capability within the frame delimited by this condition was designed.

Section 3.1 introduced the notion of importance of the database on the architectural level. It is the connection point between the IoT Platform and connected devices. That is, there is no direct communication between the platform and devices. All communications go through the database layer.

For this purpose in the database was created a table called `pend_msg` shown in Figure 7.3. The device configuration cycle begins when the user sends a configuration message to a device. The message is inserted into the `pend_msg` table. Then, at some moment, the device sends `PEND_REQ` message to the gateway which queries the database for pending messages. Once the message is sent by the gateway and acknowledged by the device, the configuration cycle finishes.

This strategy is simple and convenient since it satisfies the main condition of the device initiative and assures that it will receive the message without loss.

7.3.3 User Management

The IoT Platform proposes the user management model with four possible roles: interface, user, administrator, and superuser. Each role has its own privileges and restrictions. Next listing plainly exposes the user model describing privileges, access permissions, and administrative features. The order of importance is preserved from low to high.

1. Interface

   When a user is assigned this role (it only can be assigned by administrators), nothing can be created or added. It will allow only to watch created applications, alarms and automation, added devices and their data.

It was thought for business cases when a company just would like to share certain application with its customers without the right for modification. For instance, an IoT startup may serve its applications for particular client companies.

2. User

   Everything that is pertaining to a normal user can be carried out with this role. The user can create new applications, alarms and automation, add new devices.

3. Administrator

   This role assumes much more responsibility, privileges, and extensive access. Administrator, apart from having the same privileges as user has access to user information. Administrator can create, update, and delete other users' applications, alerts and automation, and devices.

   Additionally administrators have user management privileges. They can create, update, and delete accounts with interface and user roles.

   Administrators cannot delete neither access to other administrators profiles and content. They are assigned by the superuser.

4. Superuser

   The superuser is the only role which has total control over the platform. Everything what can be done by normal administrators can be done by the superuser. The superuser is the only one who assign administrators and can withdraw their privileges and delete their accounts. Since it is the role of major control, there can be only one superuser per platform.

   When the platform is installed it is the default created user.

Administrative features and privileges system can be improved for being more flexible. These improvements will be mentioned in Section 9.2.

7.3.4 Logging

The logging information is important on various levels. Every serious system must implement a logging strategy. Generally, three segments of users can be mentioned which take mostly advantage of the logging: developers, system administrators, and security specialists.

Developers use logging as a tool for finding bugs in code during and after implementation phase. Normally, logging is enabled where other debugging tools cannot be used, it helps to identify problematic places. Different logging levels can be employed to reach more or less verbose information output about the program procedures.

System administrators monitor activity of numerous computers assuring their reliable running conditions. They often access logs to check that everything works as expected. Suspicious activity as administrator user management out of working hours can be noticed and reported.

Another beneficial group is the security analysts. This group accesses log information to analyze it from authentication to resource accesses.

IoT Platform implements logging at different levels recollecting information at every functionality aspect. All authentications, resource management as application and device creation and deletion events are recorded.

7.3.5 Deployment Over uWSGI

uWSGI which stands for Web Server Gateway Interface is a full-fledged HTTP server that, at the same time, is a deployment option on servers as Flask, nginx [70], lighttpd [56], and cherokee [14]. It is both, a protocol and an application server.

Back in the day, when a developer wanted to design a web application in Python the options were very poor and limiting, with security and performance issues. uWSGI appeared in time when the Python community was considering new architectures and options for web applications development.

In essence, uWSGI is a spec for an interface between a Python application and a web server. Its point was to allow a web server to deploy any Python application as long as it follows the spec. The spec is defined in official PEP 333 [78] standard for Python Web Server Gateway Interface. The whole standard can be roughly reduced to providing to the server a callable Python object which launches the application. The application receives requests from the web server, processes them, and returns back to the web server which manages network connections.

Figure 7.7 depicts a typical network request flow from a client to a Python application and returning the response back to the client. A client initiates the process sending a request to the server. uWSGI server receives the request, manages the HTTP connection, and passes the request to the Flask framework. Flask selects a corresponding to the request handler and passes the request to the handler which represents a portion of actual application implementation. Once the response has been prepared, the reverse process is straightforward.

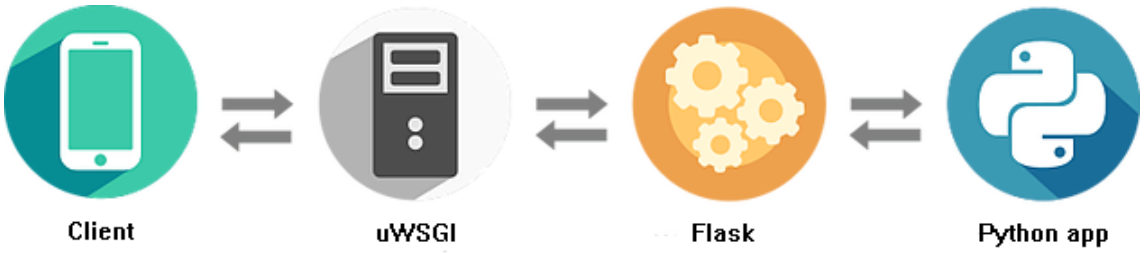


Figure 7.7: Network request flow from a client to a Python application and response flow back to the client.

The server uses different configuration formats among which `.ini` files is a de facto standard. It consists of `[section]`s and `key=value` pairs and provides an incredible flexibility with numerous configuration options.

For the configuration file of the IoT Platform the next options were used

```
[uwsgi]
```

```
http = :8080
wsgi-file = run.py
callable = app

master = true
processes = 2
threads = 7
```

`http` option determines the server listening port. `wsgi-file` and `callable` options define the callable Python object that launches the application, as PEP 333 indicates. `master` option enables a master process for better resource management. The rest of options define the number of processes and threads that the server can dispose for the tasks.

7.3.6 Real-time Alerts and Automation Development

Availability of real-time alerts significantly raises the platform value as users become rapidly aware of important changes in their applications behavior. The support for such functionality requires development of a certain strategy that will permit device data reception, processing, and alerts dispatch.

In the context of the project architecture the core element is the database. Thus, a certain database functionality must be explored and adapted to support the necessary workflow.

Publish-Subscribe Database Paradigm

Section 3.4 introduced database technology used in the project, and Section 7.2 brought insights into how it was adapted and used. This section will describe database features used for achieving support for real-time alerts and automation.

The publish/subscribe paradigm is implemented in PostgreSQL using `listen` and `notify` functions. These functions provide a simple form of inter-process communication for, possibly distributed, processes connected to the PostgreSQL database. It means, once a connection to the database is established, it can send messages to clients.

Clients must implement routines for processing database messages. The server sends a string-based payload that can contain any text message with up to 8kB length. A conventional and often suitable way of communication is using JSON-encoded payload. When Python application receives such payload, it automatically converts it in a standard dictionary which is comfortably treated from the code.

The mechanism for event publication is called a Trigger. Trigger is a function automatically invoked when a specified event happens, e.g. `INSERT`, `DELETE`, or `UPDATE`. Triggers are associated with database tables.

In order to create a trigger developer must define a trigger function first, and then attach

the trigger function to a table. The defined trigger is the mechanism to invoke the defined function. For instance, if the trigger is attached to the `INSERT` event, then, on every `INSERT` the trigger will be invoked and call the defined function.

The idea is to execute `notify` routine inside the trigger function. The notifications are delivered through channels. The channels are dynamically-defined message-passing interfaces. A client that is interested in receiving messages from a channel has to start listening to the specific channel. It is the way of how subscriptions are performed. Using publish-subscribe terminology, triggers publish messages to a defined channel, and clients consume those messages subscribing to the channel, while the database core works as the broker.

Standard `psycopg` library extensively used in the development process provides mechanisms for using publish-subscribe functionality, but the syntax is odd and difficult to read. Instead `pgpubsub` library was used that provided a more convenient way to listening to PostgreSQL channels.

When the platform user creates a new alert or automation, a function is created and linked to the trigger. The trigger is attached to the device data table. On the other hand, the IoT Platform runs several threads that listen and process the incoming database messages. The database does not interpret device data by itself it just forwards incoming messages from devices to the defined channel. The platform receives all messages, checks corresponding conditions, and, once met, specified actions take place.

Device automation uses identical to the alerts strategy. The only difference is when the alert or automation condition is met, the alert invokes mailing services, and the automation enqueues specified messages to the automated devices.

The main limitation associated to this mechanism is absence of message queuing mechanism. If there are any connection problems and the client could not receive the message, it means the message is lost forever. Thus, it is crucial to have a reliable database connection.

Email Service Integration

Many web applications require the ability to send emails to their users. Flask does not provision out-of-box functionality to send mails. However, there is an extension, called Flask-Mail, that can be installed and easily used for establishing a connection with any email server.

Special `Jinja2` template for sending an email message was created. It allowed customization of alert messages maintaining fixed message format. Figure 7.8 presents an example of fired alarm email message.

Figure 7.8: Alert email message example

7.4 User Interface (UI)

A good UI possesses the usability quality. Basically, the usability is a quality attribute that assesses how easy user interfaces are to use [71].

Usability is compound of 5 quality components:

- Learnability

  It has to do with the ease or difficulties users face using the interface the first time.

- Efficiency

  Efficiency defines the speed of completing tasks once the design is learnt.

- Memorability

  After a period of not using the design and getting back to it, users tend to find difficulties of reestablishing their proficiency. The memorability addresses this aspect.

- Errors

  This aspect covers everything related to errors the users make using the interface.

- Satisfaction

  The aim of good user interface is to provide satisfactory user experience.

Usability is important due to simple fact that when a web page or platform is difficult to use, users leave it. If they are confused or lost using the platform, they will leave. Of course, there can exist user manuals for using a platform or a web site, however, a good design never requires additional information, it is intuitive.

The best has been done during the user interface design to make it intuitive and simple.

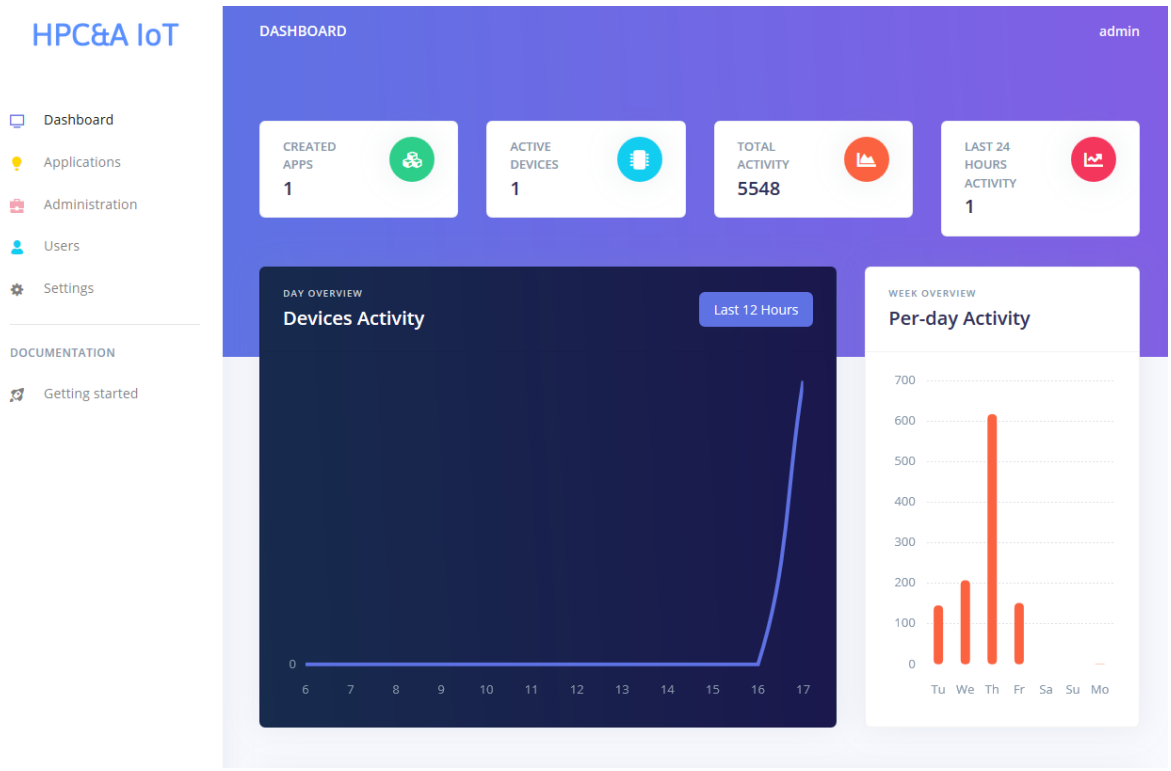


Figure 7.9: IoT Platform user dashboard, upper part

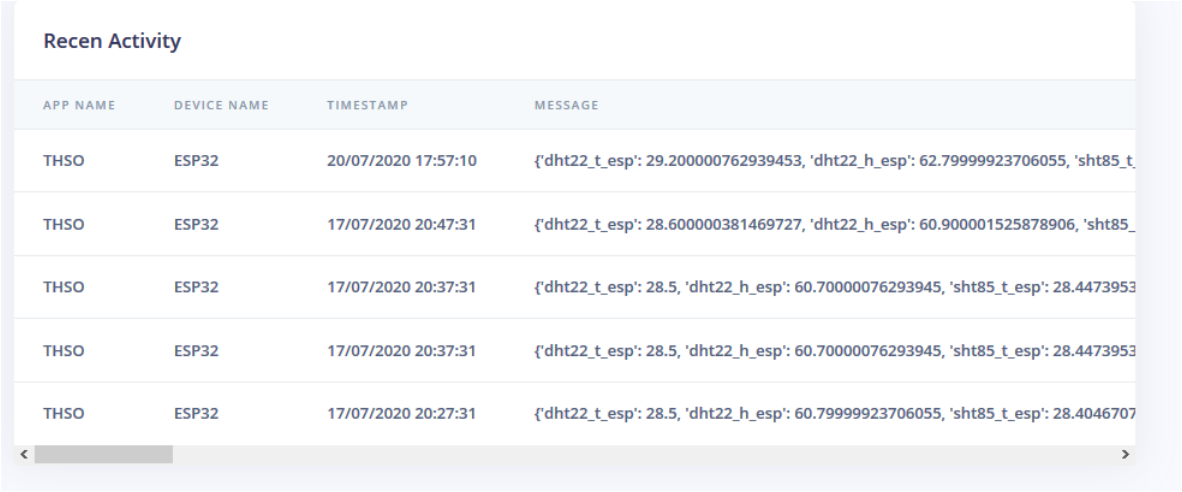


Figure 7.10: IoT Platform user dashboard, bottom part

The purpose of using an IoT Platform is to monitor and manage IoT applications. Thus, every user must have a dashboard which will provide an overview for devices activity. With that in mind, the dashboard shown in Figures 7.9 and 7.10 was designed.

On the left part, Figure 7.9, the user has the navigation panel where access to applications and settings is provided. If the user has administrator privileges, additional administrative

options such as platform Administration and Users list will be available.

On the top part there are four cards that summarize user and devices activity. Number of created applications, devices, total devices activity, and devices activity during last 24 hours is provided. The activity is represented as a number of received from devices messages.

In the central part, on the left, the user has a curve chart for per-hour devices activity during last 12 hours, and, on the right, a bar chart with per-day activity during last week. On the bottom part a recent activity table is constructed with 5 last messages indicating applications and devices names, timestamps, and content.

The rest of the UI views will be presented in Appendix B.1.

7.5 Chapter Summary

The current chapter provided a number of details about the IoT Platform architecture, development, and other internal details. How the project organized from the structural point of view as well as its design process was described first. As was mentioned in Section 3.4 the author used DAO software design pattern to provide flexibility and easy extension changing the database technology. The details of this process was discussed in this chapter. Another important aspect was the platform internal logic that was related in application, device, and user management, logging, real-time features implementation, and deployment details. Finally, the UI design was partially presented in this chapter, and the full design was included in the Appendices section.

Chapter 8

Tests and Validation

Contents

Verification and validation took place at different phases of the infrastructure development. This process was intentionally overlapped with the development of another project called Temperature/Humidity Sensors Observance (THSO). The project consisted in gathering temperature and humidity data from numerous cheap as well as expensive sensors for analyzing their behaviour. Thus, communication protocol, gateway development, and the IoT Platform were validated within the THSO frame.

The THSO system configuration is shown in Figure 8.1. It is compound of three MCUs each connected with certain number of temperature/humidity sensors. They communicate over UART with specially designed master/slave protocol. The central node, master, is ESP32 MCU and located in central part. It periodically polls sensor data from Wisen WhisperNode (on the left) and Arduino MKR1000 (on the right) boards. Once sensor data are gathered, it prepares a packet with binary data and sends it to the gateway over WiFi. The sampling period can be remotely adjusted using the IoT Platform manually, or employing automation.

The validation process must preserve certain order since the IoT Platform cannot be validated without appropriate operation of the gateway firmware. And the gateway, in turn, cannot be thoroughly validated without the Communication Protocol correct operation. This chapter describes the validation process during different phases of the IoT Platform development.

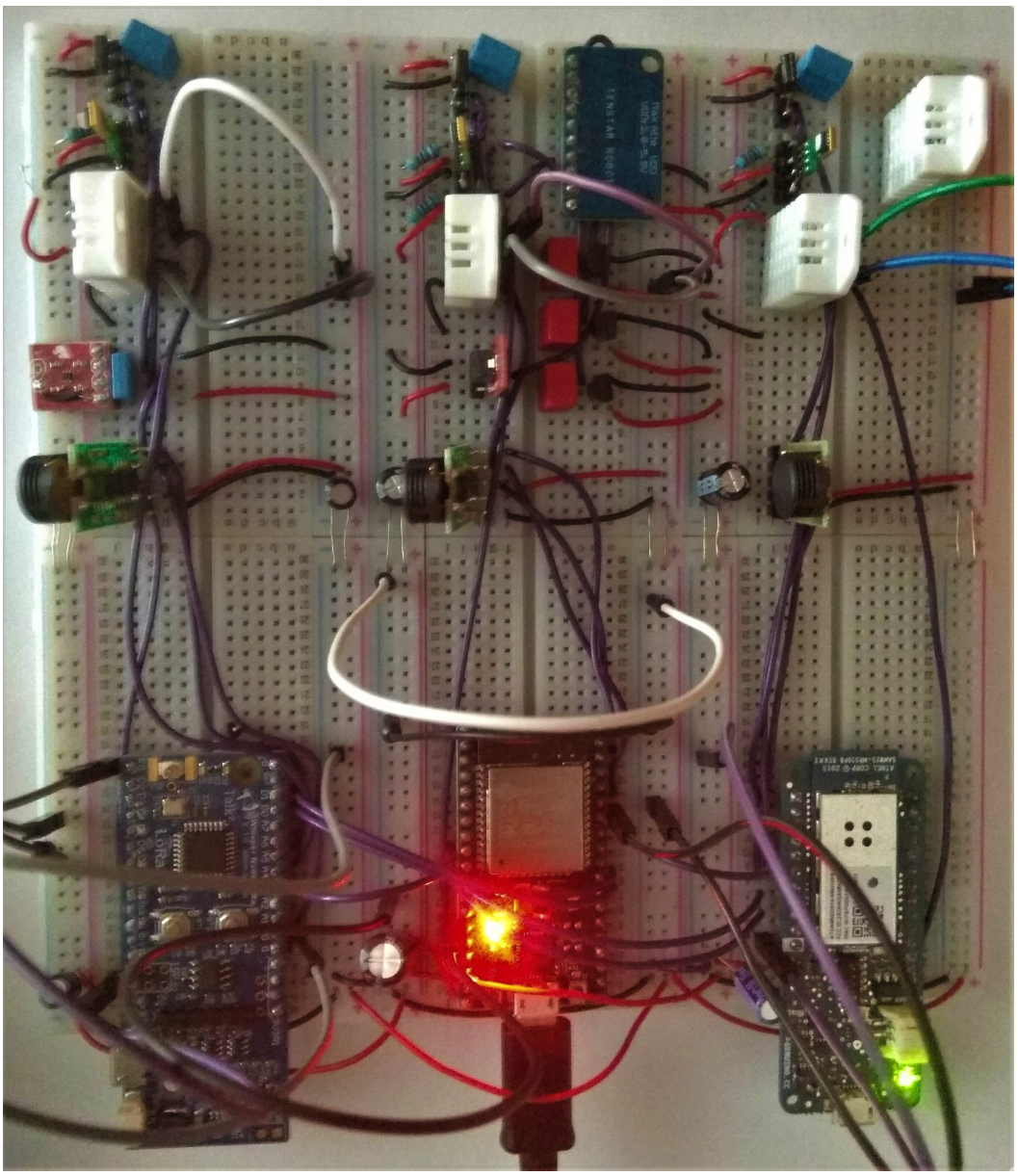


Figure 8.1: Temperature/Humidity Sensor Observance (THSO) system configuration

8.1 Communication Protocol and Gateway Validation

The validation of any computer protocol must be done having at least two points of communication. This is why the gateway was involved in the communication protocol validation.

The validation consisted of several steps:

1. Protocol implementation on device side
2. Protocol implementation on gateway side

3. Device-gateway interaction tests
4. Use of Packet Sender tool

Implementation on both, device and gateway, sides was required due to their functional difference. That is, the device does not require gateway functions and vice versa. On the other hand, it is better to separate the implementation for memory optimization reason. Implementation of only necessary routines on the device side will reduce the memory usage which is often scarce resource.

Every required function was written and tested on both sides. As the protocol is designed to be device-initiated, gateway just must return corresponding responses and perform associated actions.

`TIME_REQ` packet was sent from the device and corresponding `TIME_SEND` response containing current UTC was returned. The same test was performed using Packet Sender. It is an open-source tool that allow sending and receiving UDP and TCP packets [67]. It significantly helps to test APIs, protocols, and even to automate tests processes. Employing this tool consistent results were always returned.

`DATA_SEND` packets were also sent from the device as well as from the Packet Sender. Gateway was confirming every message, and, in effect, all device data were inserted into the database. It was confirmed by the data visualization through the platform and additionally exploring the database.

`DATA_SEND` with pending messages scenario was successfully tested as well. The device was programmed with `Conf_ID=0` associated to the data send period. Therefore, from the platform was enqueued several configuration messages with `Conf_ID=0` but different arguments. The arguments had to change the data send period to 1 minute, then to 5 minutes, and finally to 10 minutes. The test was successfully performed and checked monitoring the log via debugging UART on the device, monitoring log on the gateway, and observing the database changes of the column corresponding to the message acknowledgement.

Checking for a pending message scenario was not required to test since it was embedded into the previously described case.

The described series of tests has shown that the communication protocol was functioning according to its specification and was ready to be employed in user applications.

After the multithreaded version of the gateway was implemented it was additionally tested for the throughput capability. The tool for the testing was Packet Sender used previously. Testbed condition was set programming the tool to send 100 packets every second as shown in Figure 8.2. The gateway has shown unexpected results successfully handling every message. Average latency between sequential gateway responses was approximately 10ms. Log file with the complete information about sent and received packets was delivered along with the present document.

The protocol validation was beneficially overlapped with the gateway validation. Thus, both have been successfully validated and the gateway has shown impressive throughput results.

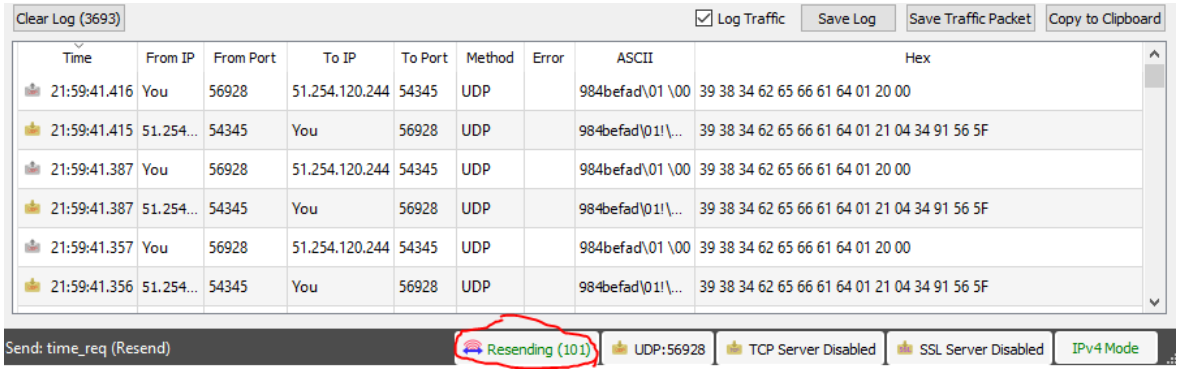


Figure 8.2: Packet Sender networking tool sending 100 packets per second

It must be noticed that the protocol and gateway functioning is not apparent. The gateway processes the communication protocol packets serving devices' requests and the users cannot see visually this work. They appreciate data that appears on the devices dashboards. Thus, for this part of the validation it is difficult to satisfy the scientific criteria for evidence recording every validation step at device and gateway sides. In the next section, it will be more clearly seen that the protocol as well as the gateway work properly together. Otherwise, it would be impossible for the THSO application to recollect the sensors data.

8.2 IoT Platform Validation

IoT Platform provides the running environment for the IoT applications. It implements necessary tools for creation of applications logical structure. That structure was designed to match the whole variety of possible physical arrangements of devices. In order to validate the IoT Platform a logical structure for the THSO project was created. An application with corresponding name was created and new device added with the binary device data model specified.

The application creation and device addition was traced with visual information presented on the platform and the database structure changes. Once the application was created (see Figure 8.3), corresponding row was inserted into the database `applications` table (Figure 8.5), and a new table for storing the list of devices belonging to the application was created (Figure 8.4).

The same operations were performed when a new device was added. Corresponding row was inserted into the `devices_49f4d289` table (see Figure 8.6). Note, that the table `devices_49f4d289` appeared in Figure 8.4 previously. The application `appkey` is `49f4d289`. The row contains device name (`ESP32`), id (`1`), description, and device data model stored in binary format. Since there are numerous variables associated with the device, the data model is so big. On the other hand, the table for storing its data, `dev_49f4d289_1` was also created (see Figure 8.4). The front-end evidence is shown in Figures 8.7 and 8.8.

It should be noticed that on the bottom part of Figure 8.8 are located tabs, where each tab corresponds to one device variable. The tabs are dynamically loaded on user demand. As

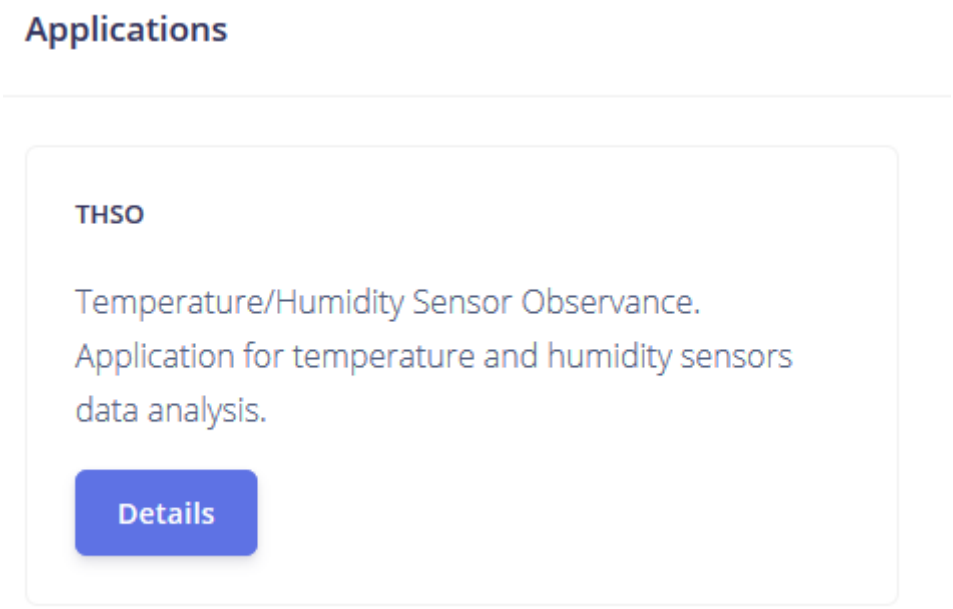


Figure 8.3: Front-end evidence for the THSO application creation

```
                List of relations
 Schema |        Name        | Type  | Owner
--------+--------------------+-------+-------
 public | applications       | table | vlad
 public | dev_49f4d289_1     | table | vlad
 public | devices_49f4d289   | table | vlad
 public | notifications      | table | vlad
 public | pend_msgs          | table | vlad
 public | users              | table | vlad
(6 rows)
```

Figure 8.4: Back-end database structure evidence for the THSO application creation

```
iotserver=> select * from applications ;
 name | app_key  | username |                                                    description
      |          secure_key        | secure
------+----------+----------+-----------------------------------------------------------------------------------------------------------
--------+--------------------------+--------
 THSO | 49f4d289 | admin    | Temperature/Humidity Sensor Observance. Application for temperature and humidity sensors data an
alysis. | c2DkXgmgXquxad8fjIBy1Q==+| f
      |          |          |      |
(1 row)
```

Figure 8.5: Back-end evidence for the THSO application creation (inserted row intro the `applications` table)

was mentioned `dev_49f4d289_1` table contains the data received from the device. Figure 8.9 shows the last 4 rows that correspond with data sent from ESP32. When a particular tab is selected a view with the data chart and table appears as shown in Figure 8.10. The log file also contains the information about applications creation and registers every added device. Figure

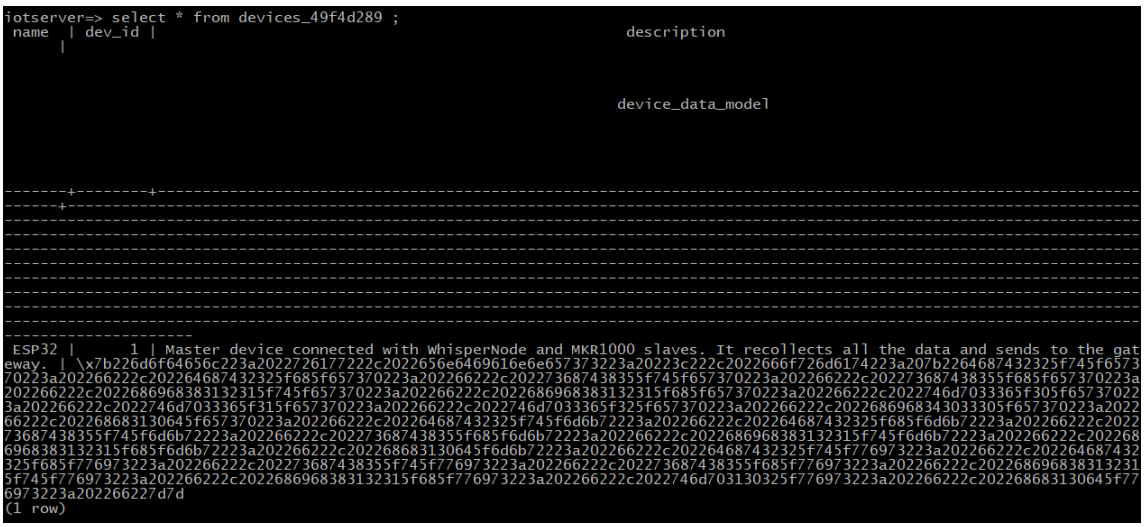


Figure 8.6: Back-end evidence for the ESP32 device addition (inserted row intro the `devices_49f4d289` table)

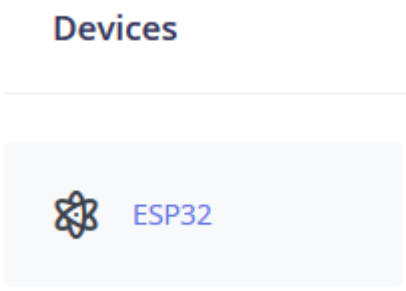


Figure 8.7: Front-end evidence for the ESP32 device addition (list of application devices view)

8.11 shows the log content when THSO was created.

The updates of the THSO application entry as well as the ESP32 device were also successfully tested. In order to test application and device deletion, a dummy application and several devices were added and subsequently deleted. Everything was traced through the IoT Platform interface user interface and database.

In order to test the alert and automation operations respective actions were carried out. First, an alert was created, as shown in Figure 8.12, to send an email if the temperature on sensor `tmp36` connected to the ESP32 is greater than 25°C. The alert successfully worked and the email was automatically sent to the indicated address.

Automation mechanism works identically as alerts. It was tested to increase sampling frequency in a hot day. When the temperature on the same sensor, `tmp36`, is greater than 25°C, then a configuration message is enqueued for the device that sets the frequency to 5 minutes instead of standard 10. Figure 8.13 shows created automation.

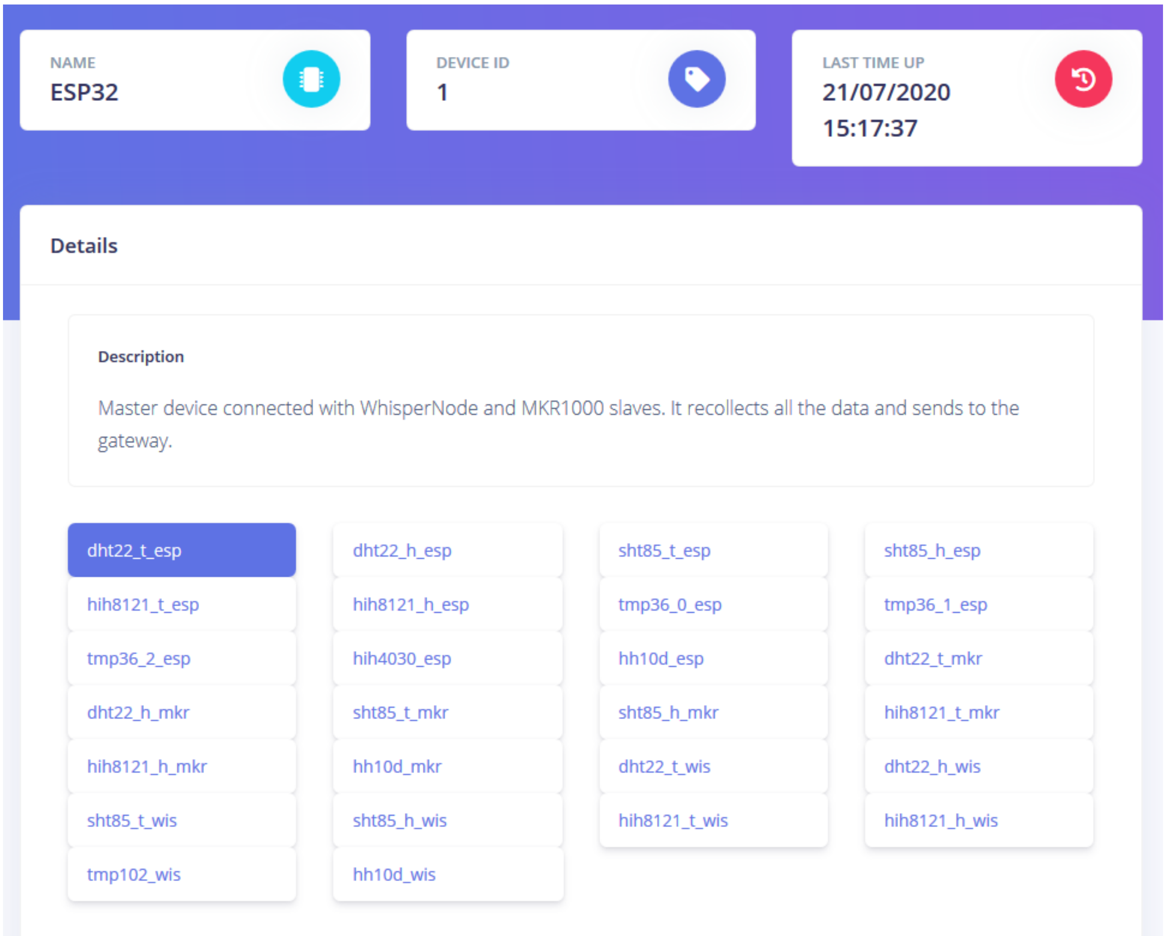


Figure 8.8: Front-end evidence for the ESP32 device addition (Device view)

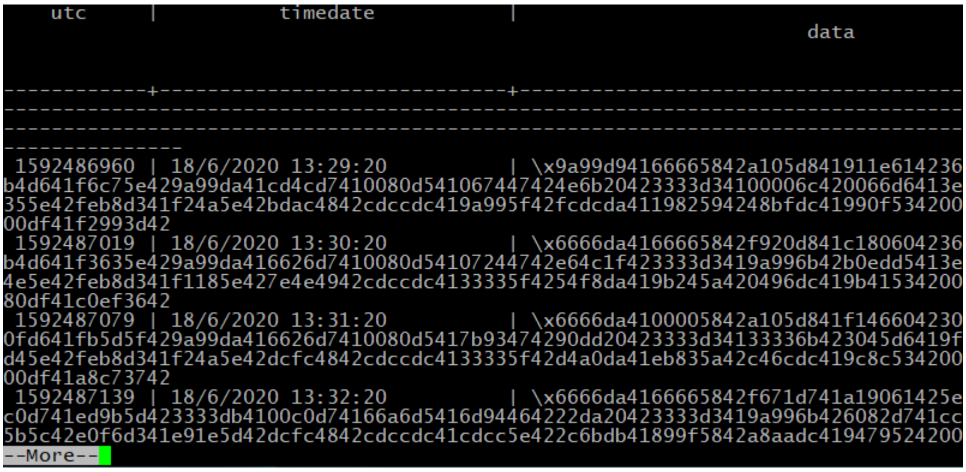


Figure 8.9: Last 4 rows in the `dev_49f4d289_1` database table

On the next incoming message from the device the automation was enabled, and the configuration message was enqueued as shown in Figure 8.14.

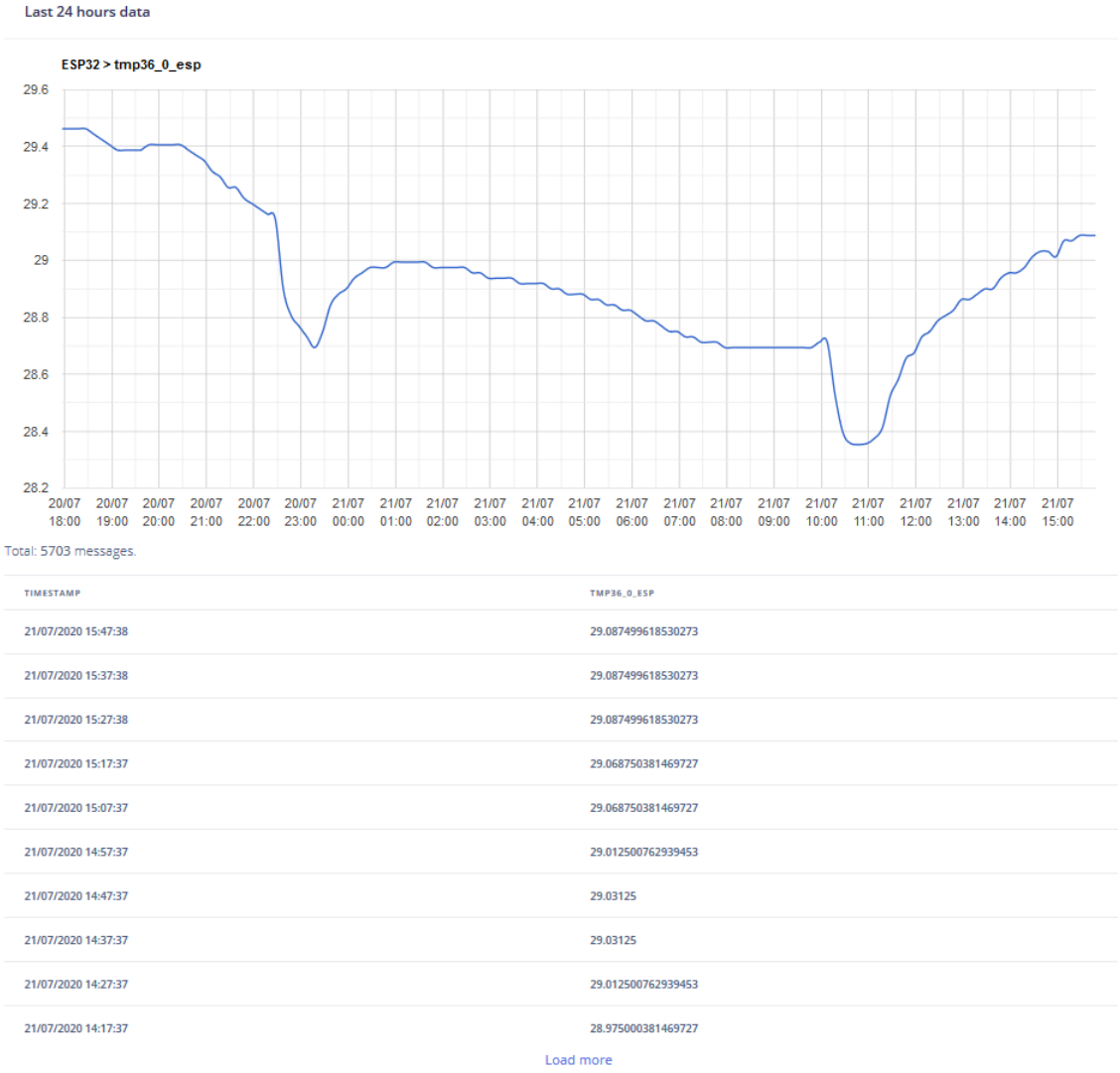


Figure 8.10: Front-end evidence for the ESP32 device addition (device variable tab view)

```
pi@raspberry:~/thso.server/app $ cat app.log
[2020-09-13 17:53:00,310] INFO in views: Logged in superuser admin
[2020-09-13 17:55:00,520] INFO in views: superuser admin created new application THSO 49f4d289
[2020-09-13 17:58:38,291] INFO in views: superuser admin added new device ESP32 for application 49f4d289
```

Figure 8.11: Log file content when THSO application was created and ESP32 devices was added

Alerts and automation removal were also tested and successfully validated. Finally, the data download feature was validated. The data were successfully downloaded in CSV format. Figure 8.15 shows a part of data downloading process.

The validation was demonstrated though the THSO logical structure creation based on the screenshots of the front-end developments and corresponding back-end automated changes. Every step of an application deployment was considered starting from an application creation and device addition, and finishing with the device data visualized and downloaded by the user.

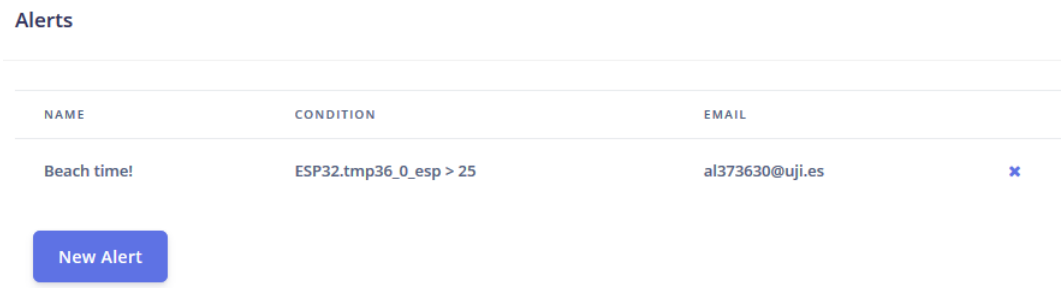


Figure 8.12: Alerts view showing the alert created for the validation test.

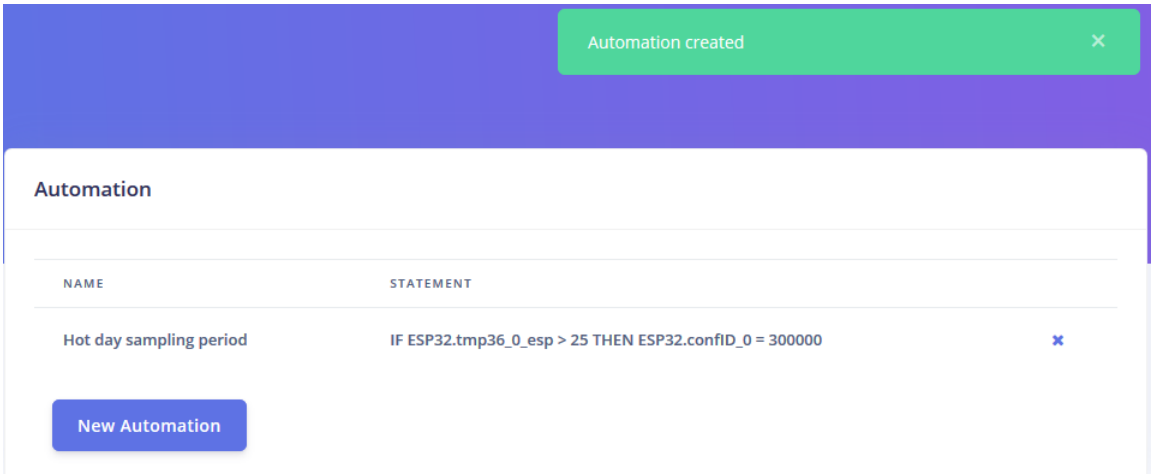


Figure 8.13: Created for validation tests automation.

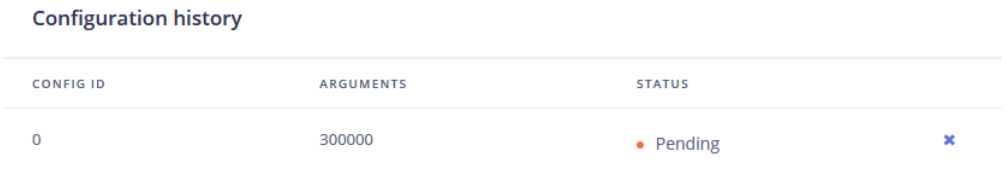


Figure 8.14: Enqueued automation message for the automation presented in Figure 8.13

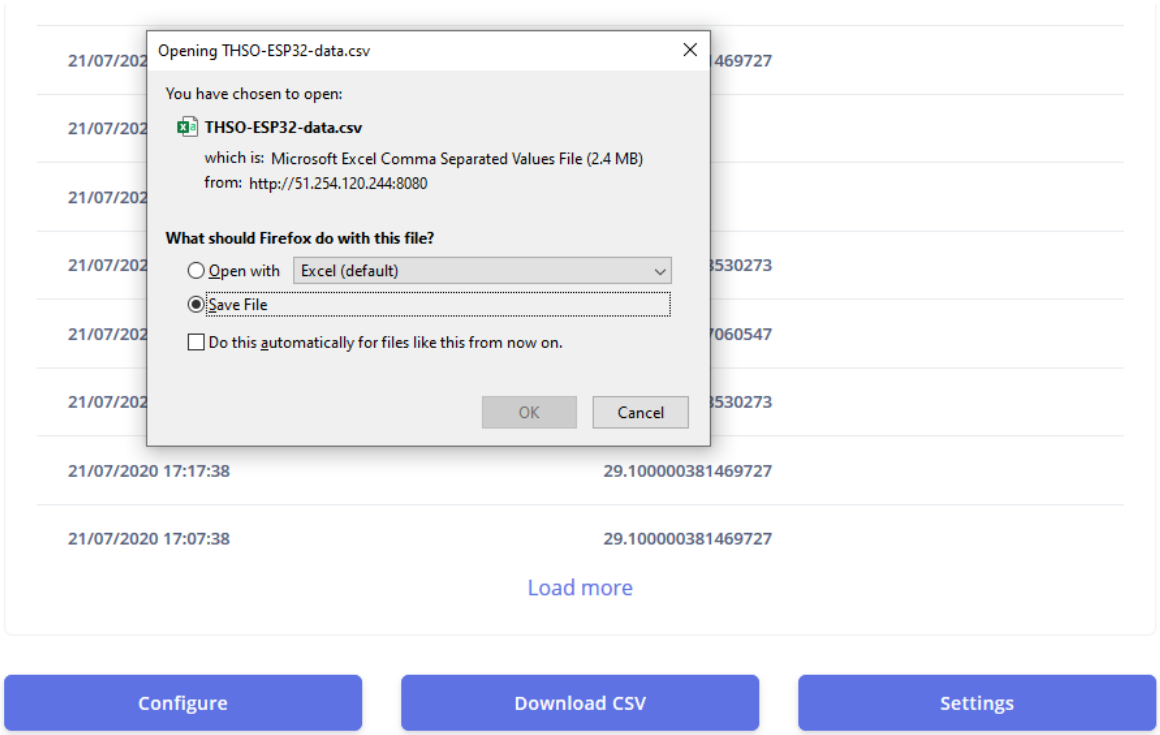


Figure 8.15: Data download validation process

8.3 Chapter Summary

This chapter was dedicated to the project validation. In the first place, the validation procedures for the communication protocol, in parallel with the gateway firmware took place. Finally, the IoT Platform complete functionality suit was successfully tested and validated. The IoT Platform validation, can also be considered as the validation of the system as a whole. It would not work without the correct Communication Protocol operation, neither without accurate running of the gateway firmware.

The overall validation process demonstrates that the proposed infrastructure is able to provide the sufficient running environment for IoT applications, accommodate and visualize devices data, generate alerts, and create automations. With that said, it can be concluded that the general and specific objectives posed in the beginning of the project are fully achieved.

Chapter 9

Conclusions and Future Improvements

Contents

9.1 Conclusions

Compete IoT infrastructure development cycle was presented in this project. From the first phases of general and specific objectives establishment, throughout the detailed planning and development stages of each architecture layer until complete final architecture verification, an enormous experience has been obtained. Soon an official open source project will be created for making public the infrastructure. It will be open for SME, students, and hobbyists.

Personally, I have seen the project as quite opportune and timely. Many times I was searching for an elegant lightweight infrastructure that can be employed for home automation and other personal projects. I think, that small business owners or technicians have done similar searches and came to only paid options that often are too complex or are not compatible with the application requirements. Once it will be made public, it will stimulate education and business development.

On the other hand, the obtained skills, working with different technologies and software are the priceless experience I received. Before starting the project I had never designed IoT architectures neither IoT communication protocols, nor developed IoT gateway firmware nor complete IoT platforms from scratch.

My current job typify a continuous research process. I come across and read numerous research papers that describe many projects. However, few of such papers demonstrate practical work and open it to the reader. I value when a research or work is practical and can be applied

to solve real-life problems, and I tried to make my best to develop this project as such.

9.2 Future Improvements

The project was finished, however, since it covers significantly wide work-scope, there are many aspects that can be improved at every level.

Communication protocol is very simple in its current version. Several aspects that can be improved are mentioned in the following listing.

- Error detection methods are not implemented. Data integrity is very important characteristic and it should be provided by a well-designed protocol. Thus, a CRC or checksum error-detection techniques should be implemented in future versions.
- Support for FOTA update. Though, there must be developed a smart scheme for firmware versioning on the IoT Platform, this feature should be implemented in future versions. Devices should have possibility to be updated remotely.
- Automation deployment avoiding polling. The real-time automation is supported though continuous polling. Mechanisms for polling-less automation will greatly alleviate network overhead.
- Security might be improved employing sessions for every device. The gateway and device, before the full-fledged communication starts, would need to exchange messages for their mutual recognition. Additional security layer can be added during this procedure.

The gateway firmware might be improved according to the Communication Protocol extension. New features will pose new challenges that must be approached in the most efficient way.

On the other hand, in order to improve compatibility with popular communication standards, several versions of new gateways should be developed. Gateway that offer REST HTTP interface should be implemented for device communication. It is very popular, standardized, and practical way for information exchange nowadays. Another gateway that provides MQTT interface needs to be implemented. Since it is used even in industrial environments, it will be a good extension for real-time automation empowerment.

Those standards have there own ways of resource identification, for instance, REST employs URLs and MQTT topics. The proposed platform resource identification, `appkey` and `devid`, will need to be adapted by those standards.

IoT Platform can also be improved in several aspects:

- User dashboard as well as device data view can be potentially improved with real-time charts. On the other hand, recent activity table from the user dashboard may include links to corresponding applications and devices.

- Geolocation services should be added in future versions. It is often useful not only to acquire device data, but also to know where it is located. Alternatively, it will introduce tracking applications to the platform.
- Alerts and automation creation views can be improved to be more elegant. In their current versions, simple HTML options and inputs are used. It can be embellished using advanced JavaScript and CSS.
- The assumption that the platform might serve its purpose being disconnected from the Internet imposed current user registration model. However, additional registration model should be introduced in the future, in order to complete registration using email activation messages. It would also bring convenient account recovery possibility.

Certainly, there are more aspects that can be improved and that will be discovered during the platform usage.

Appendices

Appendix A

Project Description

A.1 Detailed Project Planning

The general project planning was presented in Chapter 4. For the sake of clarity it was decided to move the detailed planning into the Appendices. It is presented as a series of Gantt charts. Every chart corresponds to a particular specific objective that was covered during the development.

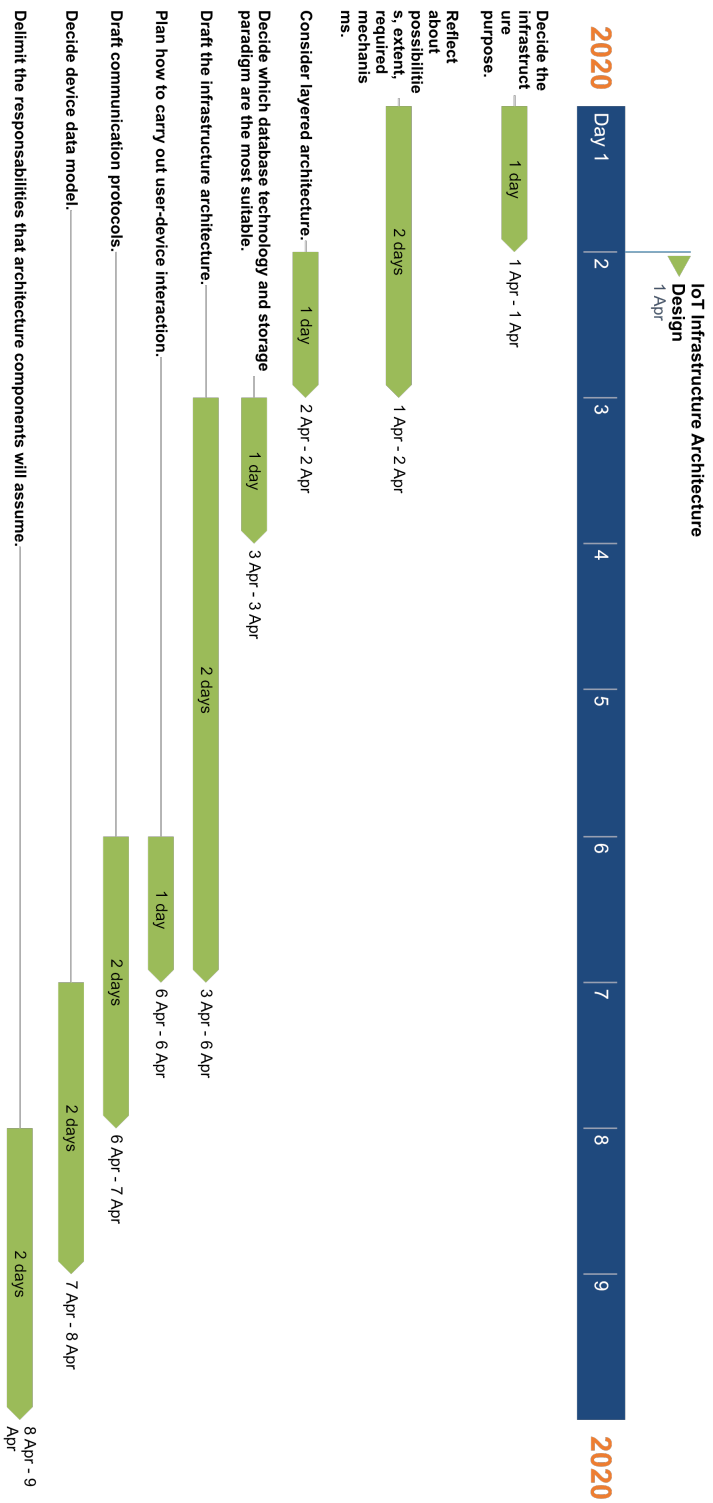


Figure A.1: Gantt chart for the IoT Infrastructure design phase

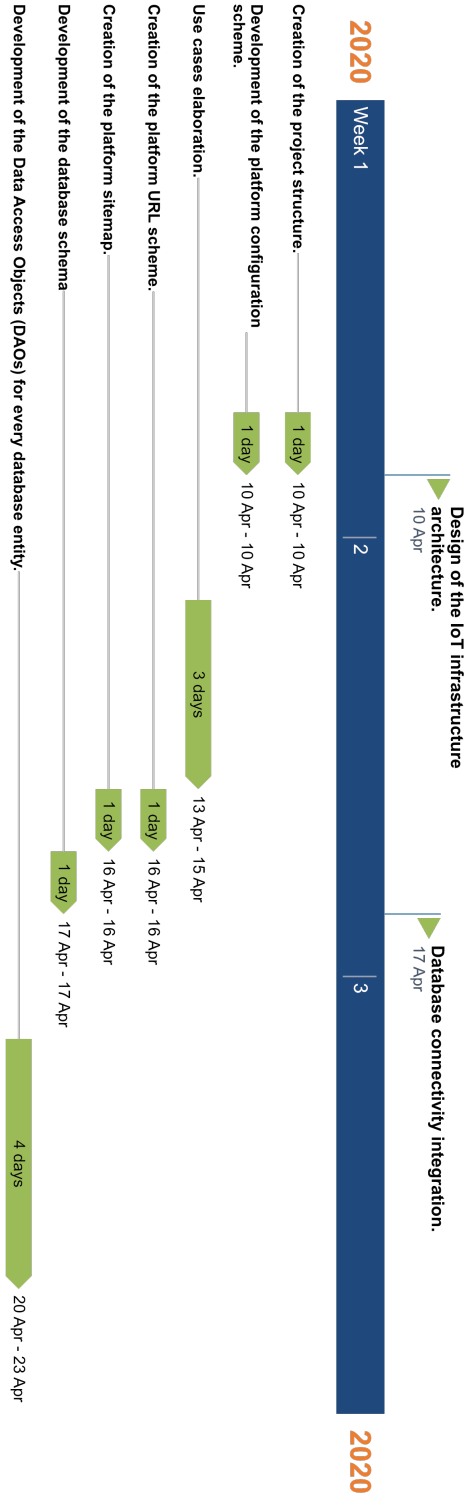


Figure A.2: Gantt chart for the IoT Platform Implementation. Part 1

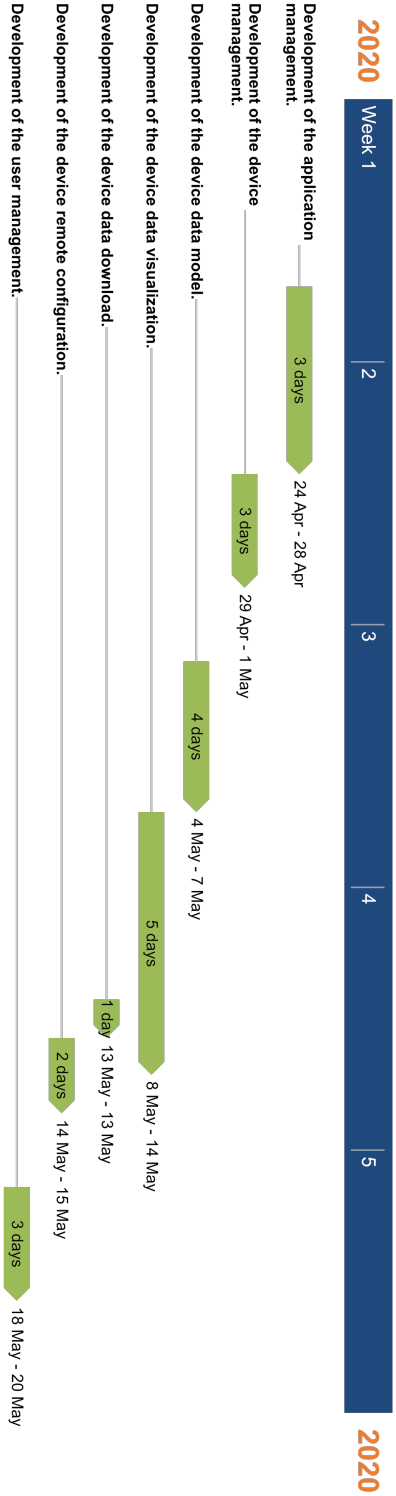


Figure A.3: Gantt chart for the IoT Platform Implementation. Part 2

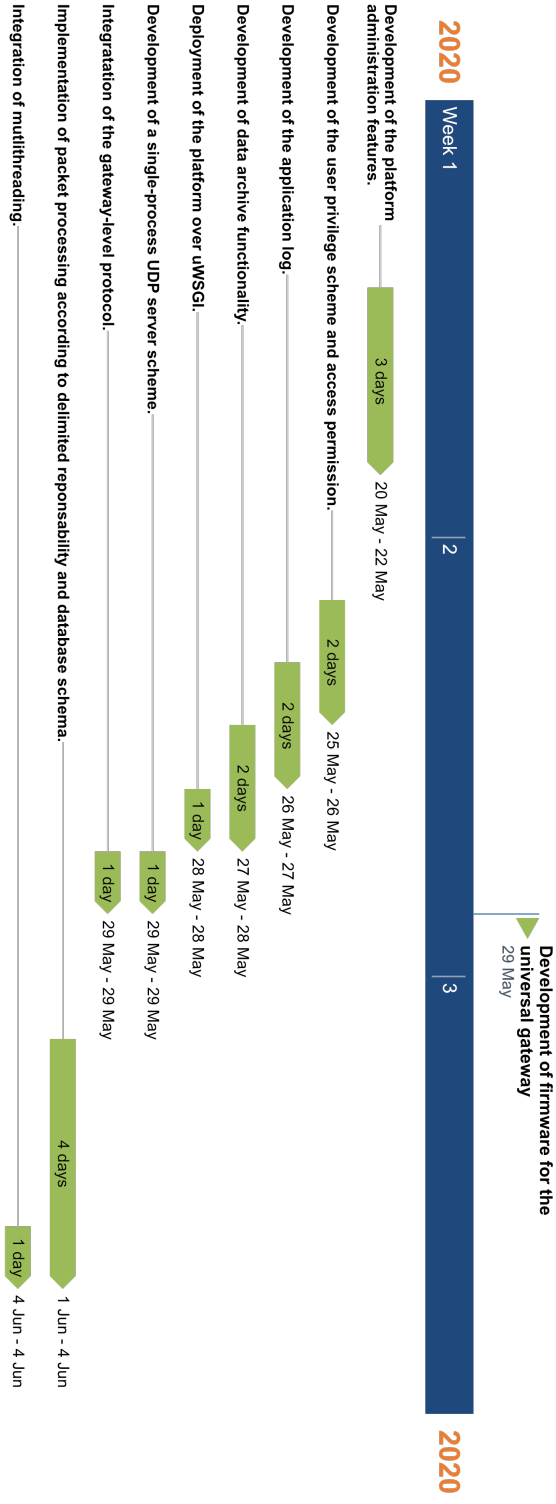


Figure A.4: Gantt chart for the IoT Platform Implementation. Part 3

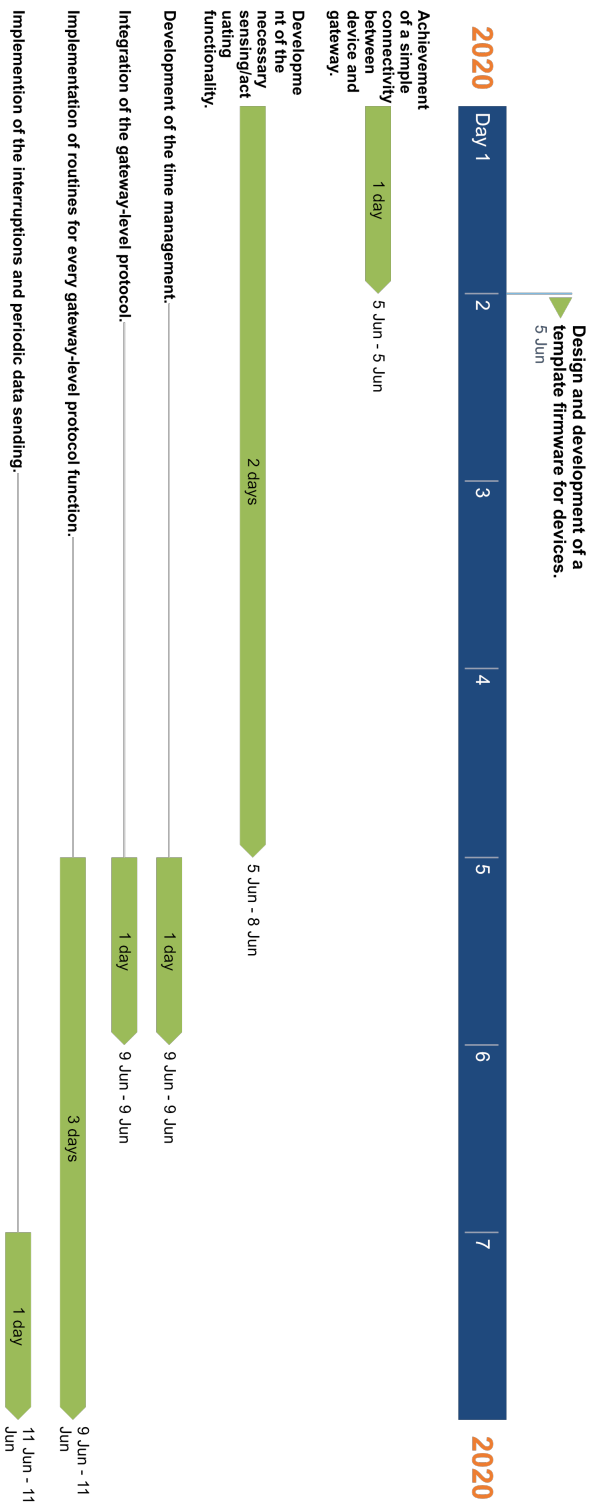


Figure A.5: Gantt chart for the Gateway development

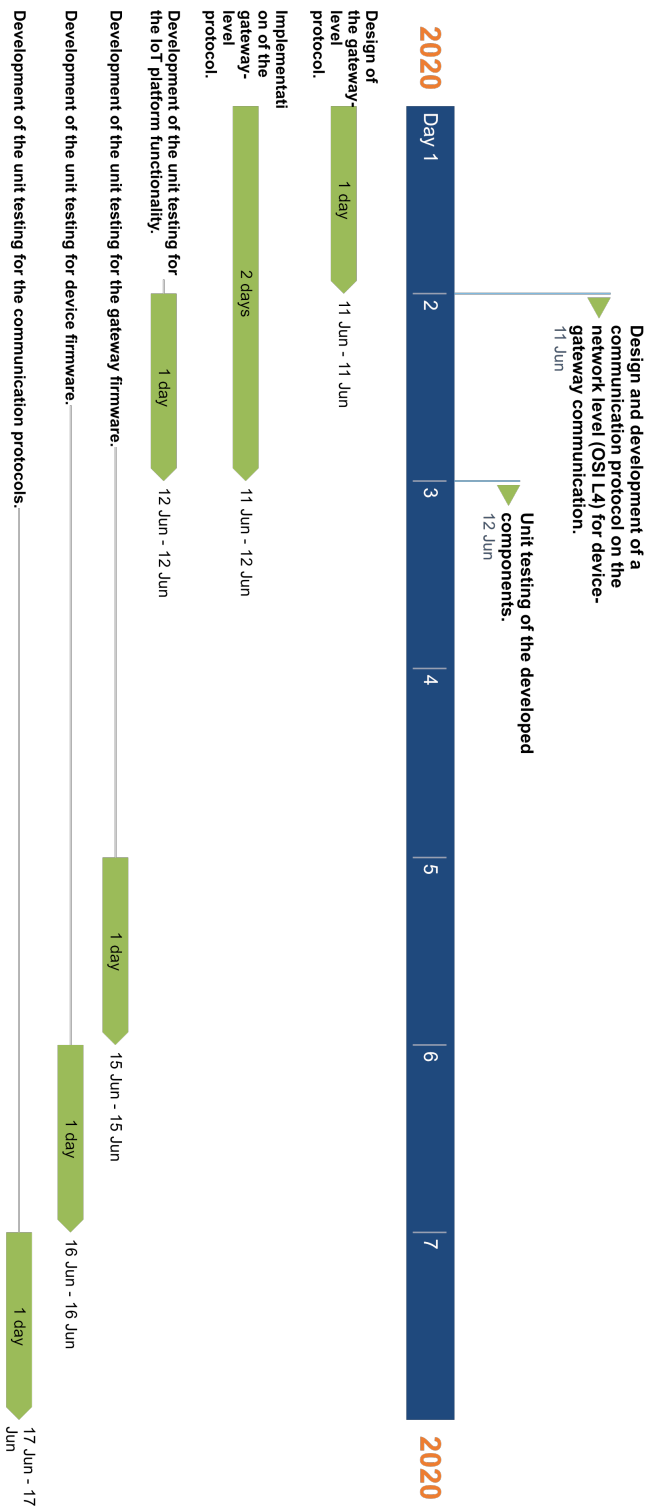


Figure A.6: Gantt chart for the Communication Protocol development and Unit Testing

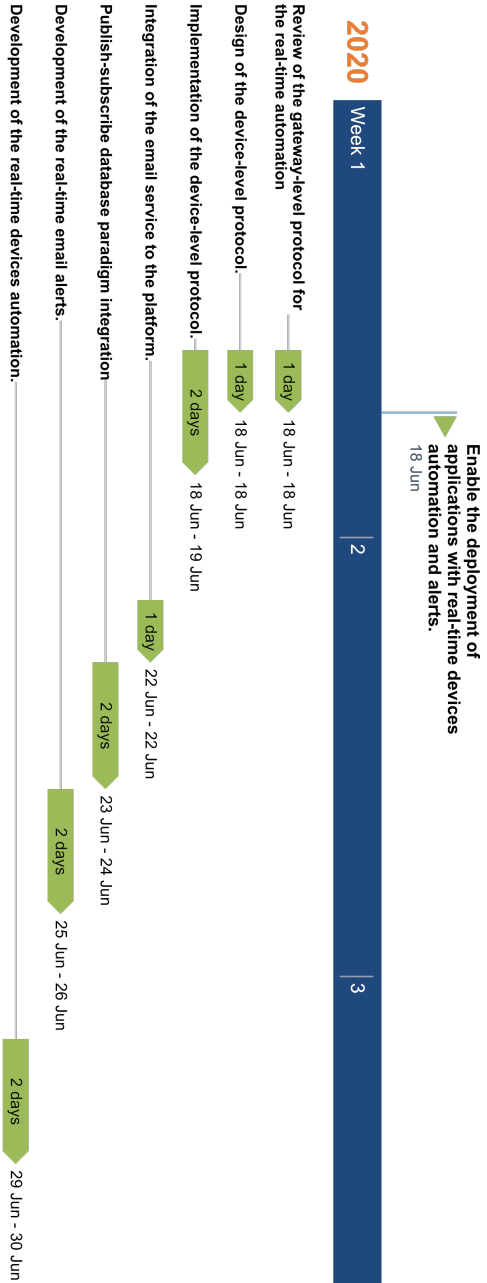


Figure A.7: Gantt chart for the IoT Platform real-time features development

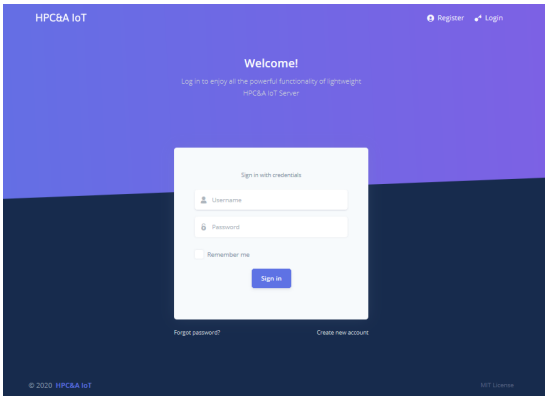
Appendix B

IoT Platform Development

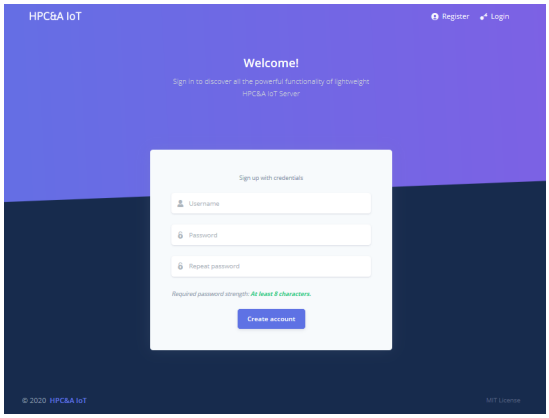
B.1 IoT Platform User Interface

Section 7.4 has discussed the importance of a good user interface (UI). It also has presented the general layout and the dashboard front-end view of the IoT Platform. This appendix will provide the rest of views.

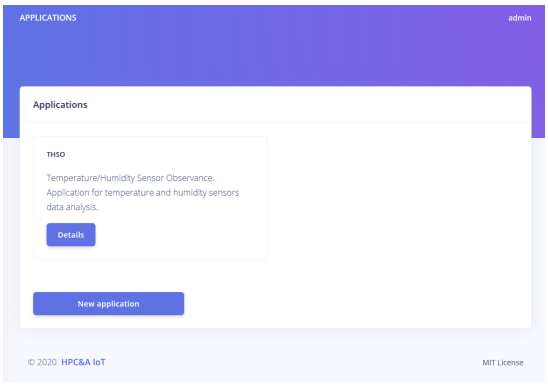
Due to numerous views of the platform the images will be packed in groups to reduce used space.



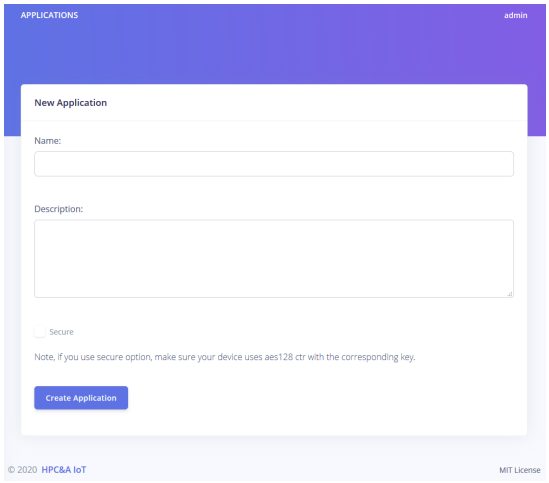
(a) Login



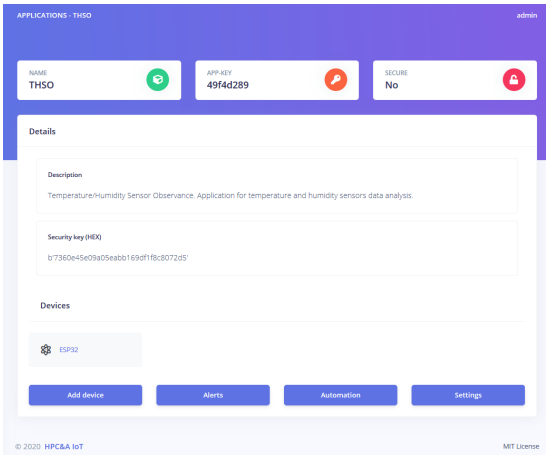
(b) New user registration



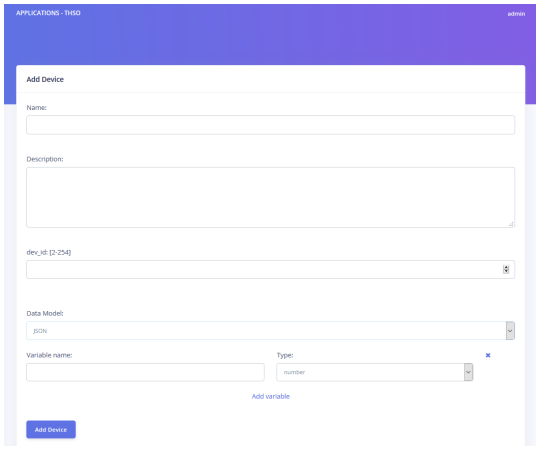
(c) Applications



(d) New application

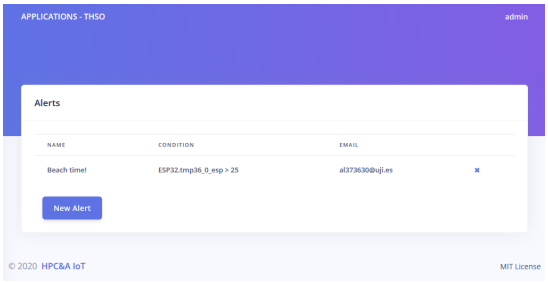


(e) Application

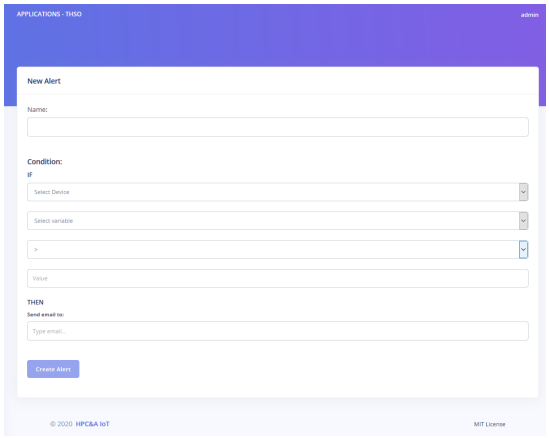


(f) Add device

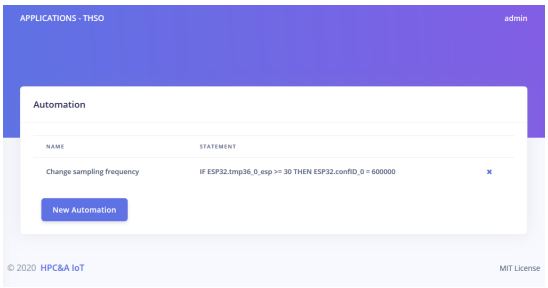
Figure B.1: IoT Platform user interface views. Part 1



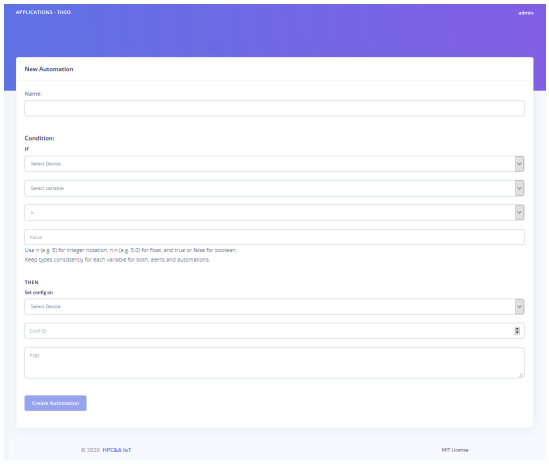
(a) Alerts



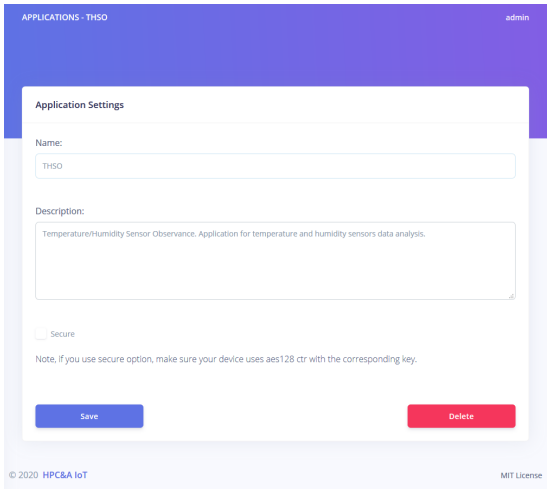
(b) New alert



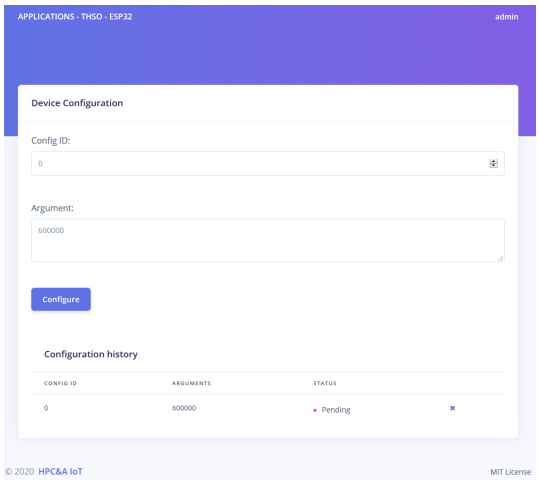
(c) Automation



(d) New automation

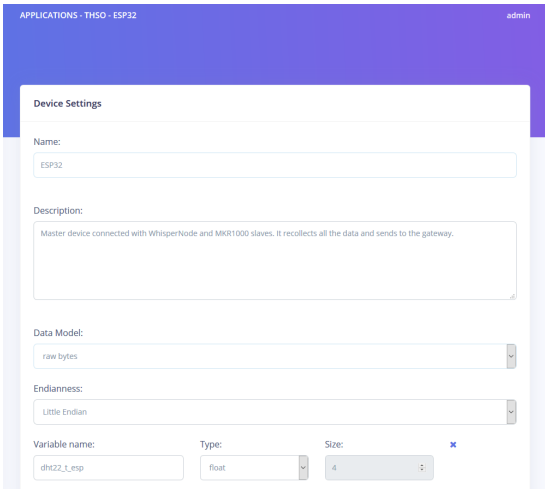


(e) Application settings

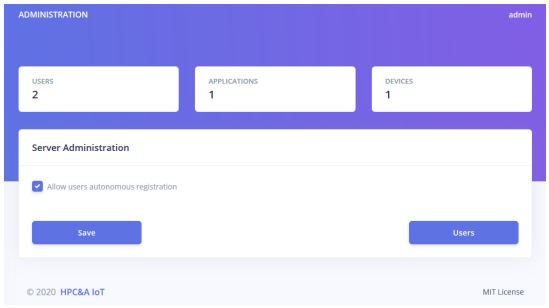


(f) Device remote configuration

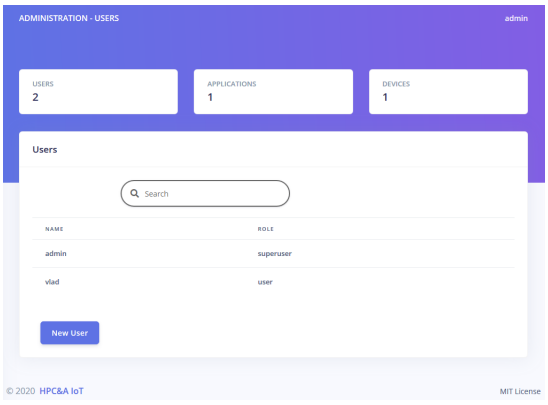
Figure B.2: IoT Platform user interface views. Part 2



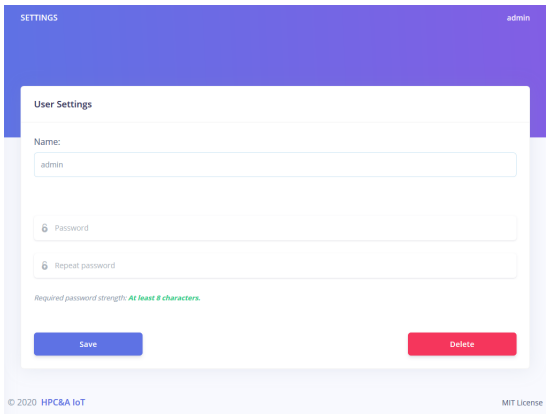
(a) Device settings



(b) Administration



(c) Users



(d) User settings

Figure B.3: IoT Platform user interface views. Part 3

Bibliography

[1] The LoRa Alliance. Lorawan specification. https://lora-alliance.org/sites/default/files/2019-05/lorawan_security_whitepaper.pdf (15.07.2020).

[2] Amazon. Aws overview. https://aws.amazon.com/ (08.09.2020).

[3] ST Application Note AN5156. Introduction to stm32 microcontrollers security. https://www.st.com/resource/en/application_note/dm00493651-introduction-to-stm32-microcontrollers-security-stmicroelectronics.pdf (15.07.2020).

[4] J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: the definitive guide: time to relax.* " O'Reilly Media, Inc.", 2010.

[5] Jauhari Arifin, Leni Natalia Zulita, et al. Perancangan murottal otomatis menggunakan mikrokontroller arduino mega 2560. *Jurnal Media Infotama*, 12(1), 2016.

[6] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[7] Aloÿs Augustin, Jiazi Yi, Thomas Clausen, and William Mark Townsley. A study of lora: Long range & low power networks for the internet of things. *Sensors*, 16(9):1466, 2016.

[8] Regis J Bates. *GPRS: general packet radio service.* McGraw-Hill Professional, 2001.

[9] Tim Berners-Lee. Universal resource identifiers. https://www.w3.org/DesignIssues/Axioms.html (14.07.2020).

[10] Yihenew Dagne Beyene, Riku Jantti, Olav Tirkkonen, Kalle Ruttik, Sassan Iraji, Anna Larmo, Tuomas Tirronen, and Johan Torsner. Nb-iot technology overview and experience from cloud-ran implementation. *IEEE wireless communications*, 24(3):26–32, 2017.

[11] Eric A Brewer. Kubernetes and the path to cloud native. In *Proceedings of the sixth ACM symposium on cloud computing*, pages 167–167, 2015.

[12] Anderson Cardozo, Adenauer Yamin, Lucas Xavier, Rodrigo Souza, João Lopes, and Cláudio Geyer. An architecture proposal to distributed sensing in internet of things. In *2016 1st International Symposium on Instrumentation Systems, Circuits and Transducers (INSCIT)*, pages 67–72. IEEE, 2016.

[13] Josiah L Carlson. *Redis in action.* Manning Publications Co., 2013.

[14] cherokee. cherokee. http://cherokee-project.com/ (20.07.2020).

[15] Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage.* " O'Reilly Media, Inc.", 2013.

[16] Wallace Clark. *The Gantt chart: A working tool of management.* Ronald Press Company, 1922.

[17] European Comission. The future internet platform fiware. `https://ec.europa.eu/digital-single-market/en/future-internet-public-private-partnership` (07.07.2020).

[18] FIWARE Community. What is fiware? `https://www.fiware.org/developers/` (07.07.2020).

[19] Vim community. About vim. `https://www.vim.org/about.php` (28.06.2020).

[20] Brian P Crow, Indra Widjaja, Jeong Geun Kim, and Prescott T Sakai. Ieee 802.11 wireless local area networks. *IEEE Communications magazine*, 35(9):116–126, 1997.

[21] Márcio José da Cunha, Marcelo Barros de Almeira, Renato Ferreira Fernandes, and Renato Santos Carrijo. Proposal for an iot architecture in industrial processes. In *2016 12th IEEE International Conference on Industry Applications (INDUSCON)*, pages 1–7. IEEE, 2016.

[22] Waltenegus Dargie and Christian Poellabauer. *Fundamentals of wireless sensor networks: theory and practice.* John Wiley & Sons, 2010.

[23] ESP8266 Datasheet. Esp8266ex datasheet. *Espr. Syst. Datasheet*, pages 1–31, 2015.

[24] Mysql vs nosql. https://www.educba.com/mysql-vs-nosql/ (28.06.2020), 2020.

[25] Federico Di Gregorio and Daniele Varrazzo. Psycopg: Postgresql database adapter for python, 2010.

[26] Massimo Di Pierro. *web2py.* Lulu. com, 2013.

[27] UBIDOTS API docs. Send data over tcp/udp. `https://www.ubidots.com/docs/hw/#send-data` (15.07.2020).

[28] ESP32 Documentation. Flash encryption. `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/flash-encryption.html` (15.07.2020).

[29] FIWARE Documentation. Ultralight 2.0 protocol. `https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual/index.html` (15.07.2020).

[30] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Lsb: A lightweight scalable blockchain for iot security and anonymity. *Journal of Parallel and Distributed Computing*, 134:180–197, 2019.

[31] Jordan Eller. Iot statistics and trends to know in 2020. https://dzone.com/articles/radio-solutions-of-iot (25.06.2020), 2018.

[32] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011.

[33] Shahin Farahani. *ZigBee wireless networks and transceivers.* Newnes, 2011.

[34] Roy Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Message syntax and routing. Technical report, RFC 7230, June 2014 (TXT), 2014.

[35] Sadayuki Furuhashi. Messagepack. *URL: https://msgpack. org*, 2013.

[36] Google. Google cloud platform overview. `https://cloud.google.com/` (08.09.2020).

[37] Google. Google line chart specification. `https://developers.google.com/chart/interactive/docs/gallery/linechart` (08.09.2020).

[38] Miguel Grinberg. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[39] Tobias Hammer. Hterm. *Abgerufen am*, 7:2015, 2008.

[40] Adrian Holovaty and Jacob Kaplan-Moss. *The definitive guide to Django: Web development done right*. Apress, 2009.

[41] Joachim Holtz. Pulsewidth modulation-a survey. *IEEE transactions on Industrial Electronics*, 39(5):410–420, 1992.

[42] Gerard J Holzmann and William Slattery Lieberman. *Design and validation of computer protocols*, volume 512. Prentice hall Englewood Cliffs, 1991.

[43] Steve Corrigan HPL. Introduction to the controller area network (can). *Application Report SLOA101*, pages 1–17, 2002.

[44] IBM. Ibm cloudant overview. `https://www.ibm.com/cloud/cloudant` (08.09.2020).

[45] IBM. Ibm watson iot service. `https://www.ibm.com/cloud/internet-of-things` (08.09.2020).

[46] Dogan Ibrahim. *ARM-Based Microcontroller Projects Using Mbed*. Newnes, 2019.

[47] Microsoft Inc. Visual studio code documentation. `https://code.visualstudio.com/docs` (28.06.2020).

[48] Motorolla Inc. Spi block guide. `https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf`. [Published on January 21, 2000].

[49] NXP Inc. I2c-bus specification and user manual. `https://www.nxp.com/docs/en/user-guide/UM10204.pdf`. [Inquired on September 2, 2018].

[50] Collective Innovation. Scrum methodology: Explained. `https://collectiveinnovation.com/scrum-methodology-explained/` (29.06.2020).

[51] William Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

[52] Eddie Kohler, Mark Handley, Sally Floyd, and Jitendra Padhye. Datagram congestion control protocol (dccp). 2006.

[53] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[54] Thomas Kothmayr, Corinna Schmitt, Wen Hu, Michael Brünig, and Georg Carle. Dtls based security and two-way authentication for the internet of things. *Ad Hoc Networks*, 11(8):2710–2723, 2013.

[55] Iot statistics and trends to know in 2020. https://leftronic.com/internet-of-things-statistics/ (21.06.2020), 2020.

[56] lighttpd. lighttpd. `https://www.lighttpd.net/` (20.07.2020).

[57] LinkedIn. Linkedin salary. `https://www.linkedin.com/salary/` (29.06.2020).

[58] Arduino LLC. Arduino/genuino mkr1000 product description.

[59] João Ladislau Lopes, Rodrigo Santos Souza, Cláudio Resin Geyer, Cristiano André Costa, Jorge Victoria Barbosa, Márcia Zechlinsk Gusmão, and Adenauer Correa Yamin. A model for context awareness in ubicomp. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web*, pages 161–168, 2012.

[60] Nuno Vasco Lopes, Filipe Pinto, Pedro Furtado, and Jorge Silva. Iot architecture proposal for disabled people. In *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 152–158. IEEE, 2014.

[61] Gordon Lyon. Nmap security scanner. *línea] URL: http://nmap. org/[Consulta: 8 de junio de 2012]*, 2014.

[62] Marcello A Gómez Maureira, Daan Oldenhof, and Livia Teernstra. Thingspeak–an api and web service for the internet of things. *World Wide Web*, 2011.

[63] Martin S Michael. Universal asynchronous receiver/transmitter, August 18 1992. US Patent 5,140,679.

[64] Brent A Miller, Chatschik Bisdikian, and Tom Foreword By-Siep. *Bluetooth revealed*. Prentice Hall PTR, 2001.

[65] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[66] AB MySQL. Mysql, 2001.

[67] Dan Nagle. Packetsender. `https://packetsender.com/` (20.07.2020).

[68] Arduino Nano. Arduino nano, 2018.

[69] The Things Network. The things network. `https://www.thethingsnetwork.org/` (08.09.2020).

[70] NGINX. Nginx. `https://nginx.org/` (20.07.2020).

[71] Jakob Nielsen. Usability 101: Introduction to usability. `https://www.nngroup.com/articles/usability-101-introduction-to-usability/` (20.07.2020).

[72] Jacob Beningo on ARM community. 10 steps to selecting a microcontroller. `https://community.arm.com/iot/embedded/b/embedded-blog/posts/10-steps-to-selecting-a-microcontroller`, January 2014. [Published on January 12, 2014].

[73] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[74] Geoffrey Ottoy, Tom Hamelinckx, Bart Preneel, Lieven De Strycker, and Jean-Pierre Goemaere. Aes data encryption in a zigbee network: Software or hardware? In *International Conference on Security and Privacy in Mobile Information and Communication Systems*, pages 163–173. Springer, 2010.

[75] Mike Owens. *The definitive guide to SQLite*. Apress, 2006.

[76] Rafael Pastor-Vargas, Llanos Tobarra, Antonio Robles-Gómez, Sergio Martin, Roberto Hernández, and Jesús Cano. A wot platform for supporting full-cycle iot solutions from edge to cloud infrastructures: A practical case. *Sensors*, 20(13):3770, 2020.

[77] Sagarkumar Patel, Vatsal Shah, and Maharshi Kansara. Comparative study of 2g, 3g and 4g. *International Journal of Scientific Research in Computer Science. Eng. Inf. Technol*, 3:55–63, 2018.

[78] pep333. pep333. `https://www.python.org/dev/peps/pep-0333/` (20.07.2020).

[79] Serguei Popov. Iota: Feeless and free. *IEEE Blockchain Technical Briefs*, 2019.

[80] Manisha Priyadarshini. 10 most popular programming languages in july 2020: Learn to code. `https://fossbytes.com/most-popular-programming-languages/` (13.07.2020).

[81] Mark Ramm, Kevin Dangoor, and Gigi Sayfan. *Rapid Web applications with TurboGears: using Python to create Ajax-powered sites*. Pearson Education, 2006.

[82] Javier Rodríguez-Robles, Álvaro Martin, Sergio Martin, José A Ruipérez-Valiente, and Manuel Castro. Autonomous sensor network for rural agriculture environments, low cost, and energy self-charge. *Sustainability*, 12(15):5913, 2020.

[83] Peter Saint-Andre et al. Extensible messaging and presence protocol (xmpp): Core. 2004.

[84] RA Scantlebury and KA Bartlett. A protocol for use in the npl data communications network. *Technical Memorandum*, 1967.

[85] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011.

[86] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). 2014.

[87] Nicolas Sornin, Miguel Luis, Thomas Eirich, Thorsten Kramp, and Olivier Hersent. Lorawan specification. *LoRa alliance*, 2015.

[88] Andy Stanford-Clark and Hong Linh Truong. Mqtt for sensor networks (mqtt-sn) protocol specification. *International business machines (IBM) Corporation version*, 1(2), 2013.

[89] Martin Stusek, Krystof Zeman, Pavel Masek, Jindriska Sedova, and Jiri Hosek. Iot protocols for low-power massive iot: A communication perspective. In *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 1–7. IEEE, 2019.

[90] Meilinda Eka Suryani. Platformio ide. *https://platformio.org/*.

[91] ASCII Table. Ascii table. 2007.

[92] Simon Tatham, Owen Dunn, Ben Harris, and Jacob Nevins. Putty: A free telnet/ssh client. *Available on line at: http://www. chiark. greenend. org. uk/˜ sgtatham/putty*, 2006.

[93] Technopedia. Platform. `https://www.techopedia.com/definition/3411/platform-computing` (14.07.2020).

[94] J-P Thomesse. Fieldbus technology in industrial automation. *Proceedings of the IEEE*, 93(6):1073–1101, 2005.

[95] Kun-Lin Tsai, Yi-Li Huang, Fang-Yie Leu, Ilsun You, Yu-Ling Huang, and Cheng-Han Tsai. Aes-128 based secure low power communication for lorawan iot environments. *IEEE Access*, 6:45325–45334, 2018.

[96] Eben Upton and Gareth Halfacree. *Raspberry Pi user guide*. John Wiley & Sons, 2014.

[97] uWSGI Documentation. Configuring uwsgi. `https://uwsgi-docs.readthedocs.io/en/latest/Configuration.html` (13.07.2020).

[98] Valgrind. Valgrind documentation. `https://valgrind.org/docs/manual/quick-start.html` (12.07.2020).

[99] Abhishek Viswanathan. *Analysis of power consumption of the MQTT protocol*. PhD thesis, University of Pittsburgh, 2017.

[100] Virualbox. https://www.virtualbox.org/manual/ch01.html (28.06.2020), 2020.

[101] Stephan Weyer, Mathias Schmitt, Moritz Ohmer, and Dominic Gorecky. Towards industry 4.0-standardization as the crucial challenge for highly modular, multi-vendor production systems. *Ifac-Papersonline*, 48(3):579–584, 2015.

[102] Wikipedia. Epoch (computing). `https://en.wikipedia.org/wiki/Epoch_(computing)` (10.07.2020).

[103] Wikipedia. Software design pattern. `https://en.wikipedia.org/wiki/Software_design_pattern` (15.07.2020).

[104] Wisen. Wisen whisper node documentation. `https://wisen.com.au/store/products/whisper-node-avr/` (08.09.2020).

[105] Tatu Ylonen, Chris Lonvick, et al. The secure shell (ssh) protocol architecture, 2006.

[106] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[107] Juan Carlos Zuniga and Benoit Ponsard. Sigfox system description. *LPWAN@ IETF97, Nov. 14th*, 25, 2016.