



# DESPLIEGUE DE RED LPWAN EN ENTORNO INDUSTRIAL CON MOVILIDAD

**Ricardo Hernández Álvarez**

**Tutor: Sempere Paya, Víctor Miguel**

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2021-22

Valencia, 13 de marzo de 2022

# Abstract

The technology that started massively connecting people decades ago has been developed to begin connecting devices as well. The resulting global connectivity network is called the Internet of Things. It has useful applications in every sector and is set to lead the fourth industrial revolution. Efficiency through data gathering is the goal of an ever-increasing number of devices. Energy efficiency is key to make this network scalable without skyrocketing electrical consumption. Covering big spaces with as few hardware resources as possible also helps at reducing costs. This is exactly where Low-Power Wide-Area Networks come into play.

The aim of this project is to create a tool that allows the fast and easy deployment of a LPWAN network in an industrial environment in a mobility context. The author has selected the LPWAN technology that best fits the project (LoRaWAN) and a solution based on it, ChirpStack. A functional web application has been developed as an ideal candidate to be the tool that allows LPWAN mobility deployments.

Further cost efficiency is unlocked by the developed web application, which saves the user multiple tedious configuration steps before activating a new end-device. This tool also achieves further abstraction from the technology that is being implementing, making it accessible to an even greater market.

An analysis of the results obtained highlights the success in achieving both secondary goals, a reduction in end-device activation time and an abstraction of the telecommunications technology, apart from being a mobility tool for industrial deployment of LPWAN networks.

**Keywords:** LoRa, LoRaWAN, ChirpStack, end-device, API.

# Resumen

La tecnología que comenzó a conectar masivamente a las personas hace décadas se ha desarrollado para conectar dispositivos también. La red de conectividad global resultante se denomina el internet de las cosas. Tiene aplicaciones útiles en todos los sectores de la economía y está preparado para liderar la cuarta revolución industrial, que busca la eficiencia a través de la recopilación de datos. Para lograrlo se necesita un número cada vez mayor de dispositivos, que deben ser eficientes energéticamente para permitir que estas redes sean viables tanto económica como ambientalmente. Cubrir grandes espacios con la menor cantidad posible de recursos de hardware también ayuda a reducir los costes de despliegue, y aquí es exactamente donde entran en juego las redes LPWAN (Low-Power Wide-Area Network).

El objetivo de este proyecto es crear una herramienta que permita el despliegue rápido y sencillo de una red LPWAN en un entorno industrial en un contexto de movilidad. El autor ha seleccionado la tecnología LPWAN que mejor se adapta al proyecto (LoRaWAN) y una solución basada en ella, ChirpStack. Se ha desarrollado una aplicación web funcional como candidata ideal para ser la herramienta que permita despliegues de movilidad LPWAN.

El uso de la aplicación web desarrollada conlleva además una mayor eficiencia de costes, ya que ahorra al usuario múltiples pasos de configuración tediosos antes de activar un nuevo nodo. Esta herramienta también logra una mayor abstracción de la tecnología de comunicaciones que se está implementando, haciéndola accesible a un mercado aún mayor.

Un análisis de los resultados obtenidos destaca el éxito en la consecución de dos objetivos secundarios, la reducción del tiempo de activación del dispositivo final y la abstracción de la tecnología adyacente, además de ser una herramienta de movilidad válida para el despliegue industrial de redes LPWAN.

**Palabras clave:** LoRa, LoRaWAN, ChirpStack, nodo, API.

# Resum

La tecnologia que va començar a connectar persones massivament fa dècades s'ha desenvolupat per començar a connectar dispositius també. La xarxa de connectivitat global resultant s'anomena Internet de les coses. Té aplicacions útils en tots els sectors i està preparat per liderar la quarta revolució industrial. L'eficiència mitjançant la recollida de dades és l'objectiu d'un nombre cada cop més gran de dispositius. L'eficiència energètica és clau per fer escalable aquesta xarxa sense augmentar el consum elèctric. Cobrir grans espais amb el mínim de recursos de maquinari possible també ajuda a reduir costos. Aquí és exactament on entren en joc les xarxes d'àrea àmplia de baixa potència.

L'objectiu d'aquest projecte és crear una eina que permeti el desplegament ràpid i senzill d'una xarxa LPWAN en un entorn industrial en un context de mobilitat. L'autor ha seleccionat la tecnologia LPWAN que millor s'adapta al projecte (LoRa) i una solució basada en ella, ChirpStack. S'ha desenvolupat una aplicació web funcional com a candidat ideal per ser l'eina que permeti els desplegaments de mobilitat LPWAN.

L'aplicació web desenvolupada permet una major rendibilitat econòmica, que estalvia a l'usuari diversos passos de configuració tediosos abans d'activar un nou dispositiu final. Aquesta eina també aconsegueix una major abstracció de la tecnologia que s'està implementant, fent-la accessible a un mercat encara més gran.

L'anàlisi dels resultats obtinguts destaca l'èxit en la consecució dels dos objectius secundaris, una reducció del temps d'activació del dispositiu final i una abstracció de la tecnologia de telecomunicacions, a més de ser una eina de mobilitat per al desplegament industrial de xarxes LPWAN.

**Paraules clau:** LoRa, LoRaWAN, ChirpStack, nodo, API.



# Contents

<b>INTRODUCTION</b>	<b>1</b>
1.1 MOTIVATION	1
1.2 SUMMARY OF THE METHODOLOGY FOLLOWED	2
1.3 STRUCTURE OF THE THESIS	3
1.4 OBJECTIVES	4
1.5 STATE-OF-THE-ART REVIEW	5
<b>TECHNOLOGICAL FRAMEWORK</b>	<b>6</b>
2.1 LPWAN TECHNOLOGY SELECTION	6
2.1.1 SigFox	6
2.1.2 RPMA (Ingenu)	7
2.1.3 LoRaWAN	8
2.1.4 DASH7	9
2.1.5 Conclusion	10
2.2 LoRaWAN	10
2.2.1 LoRaWAN network topology	11
2.2.2 Activation of an end device in LoRaWAN	13
2.3 LoRaWAN TECHNOLOGY SELECTION	17
2.4 WEB DEVELOPMENT CONCEPTS	18
<b>METHODOLOGY</b>	<b>22</b>
3.1 UNDERSTANDING THE NETWORK	23
3.1.1 Test suit and development platform	24
3.2 UNDERSTANDING THE TASK - REVERSE ENGINEERING	27
3.3 WEB APPLICATION DESIGN	35
3.3.1 The home view	42
3.3.2 The scan view	44



3.3.3 <i>The connect view</i>	45
<b>RESULTS</b>	<b>53</b>
4.1 USER INTERFACE	53
4.2 TIME SAVING WITH NODESAPP	57
4.3 TESTING NODESAPP	59
<b>CONCLUSIONS AND FUTURE WORK</b>	<b>61</b>
<b>BIBLIOGRAPHY</b>	<b>63</b>
<b>ANNEXES</b>	<b>64</b>

# List of figures

FIGURE 1 - REPRESENTATION OF THE LORAWAN ARCHITECTURE .....	9
FIGURE 2 - LORAWAN NETWORK TOPOLOGY .....	11
FIGURE 3 - PRE-PROVISIONED KEYS FOR LORAWAN 1.0.X ACTIVATION BY PERSONALIZATION .....	14
FIGURE 4 - OVER-THE-AIR ACTIVATION OF LORAWAN NODES, MESSAGE FLOW BETWEEN END-DEVICE, NETWORK SERVER AND APPLICATION SERVER.....	15
FIGURE 5 – LORAWAN NETWORK ARCHITECTURE USED IN THIS THESIS .....	23
FIGURE 6 - ORGANIZATION OF CODE IN NODESAPP .....	37
FIGURE 7 - SCHEMATIC OF THE NODESAPP FEATURES.....	41
FIGURE 8 - FLOWCHART NODESAPP .....	41
FIGURE 9 - FLOW CHART OF THE CONNECT VIEW .....	45
FIGURE 10 - SCREENSHOT OF THE PC VERSION OF THE HOME VIEW USER INTERFACE .....	54
FIGURE 11 - SCREENSHOT OF THE MOBILE VERSION. SCAN VIEW HTML TEMPLATE WITHOUT ERROR MESSAGES.....	55
FIGURE 12 - SCREENSHOTS OF THE MOBILE VERSION. CONNECT VIEW HTML TEMPLATE SUCCESSFUL ACTIVATION...56	

# List of tables

TABLE 1 - MEANING OF THE BITS IN THE LORAWAN DEVADDR .....	13
TABLE 2 - BYTES AND COMPONENTS OF THE JOIN-REQUEST MESSAGE .....	16
TABLE 3 - BYTES AND COMPONENTS OF THE JOIN-ACCEPT MESSAGE .....	16



# Chapter 1

# Introduction

Advanced technologies and a proliferation of devices have helped fuel the growth of Internet of Things technologies. After years of steady uptake, the IoT seems poised to cross over into mainstream business use. Sensor technology will continue to become cheaper, more advanced, and more widely available, which will in turn increase the cost-effectiveness of this technology and make new sensor applications possible. As computing power keeps increasing and mobile connectivity improves, the worldwide market for IoT solutions across multiple sectors has observed an annual growth rate of 20% CAGR (Compound Annual Growth Rate), according to the International Data Corporation (IDC). Furthermore, an even more impressive growth rate is expected during the 2020 to 2030 decade [1]. Thus, large companies are already investing in IoT in various sectors, which is shaping a varied, enormous, and steadily growing IoT market.

However, can small businesses benefit from the IoT? [2] Studies show that small and medium-sized businesses are not making the most out of the possibilities that IoT gives them. The required technical knowledge and consume of resources prevents the expansion of the Internet of Things into these segments. These resources can be time and energy related. That is why the necessity of fast, efficient, and easy-to-deploy telecommunication networks is at its all-time high. Low-Power Wide Area Network (LPWAN) technologies are aimed exactly at making networks as energy efficient as possible while covering large distance and achieving a resilient and reliable communication.

## 1.1 Motivation

This thesis was presented by Ricardo Hernández Álvarez as the final project of his degree in Telecommunication Technologies and Services Engineering in September 2021.

This document contains a study of the technology that is currently being used to deploy private LoRaWAN networks and develops the process of creating an application to automate end device connection and activation for ChirpStack [3] networks. ChirpStack provides open-source components for LoRaWAN networks under the MIT license and can be used for commercial purposes. Together, these components form a ready-to-use solution called the ChirpStack LoRaWAN Network Server stack.

In this context, this thesis finds its motivation in leveraging the potential of the open-source technology and its modular architecture to create an extra layer of abstraction in the creation and activation of an end device in a ChirpStack network. This is currently the most tedious task for a network owner, but the most common one. Thus, its simplification will significantly enhance user experience.

The scope of this work is to present an advanced prototype of a tool that accomplishes the task. The finalized application should then undergo further testing and refining in a real production environment.

## 1.2 Summary of the methodology followed

For the execution of this work, a methodology was followed that lasted from May 2021 until the end of August of that same year. During the whole process conversations and reviews were held with the thesis' supervisors to ensure that the project was headed in the correct direction. In short, the methodology followed during that time can be divided in the following parts:

**Preliminary work:** The first task was to get familiar with the main topics handled in this thesis in order to deeply understand and be able to write about them. Furthermore, an introduction to the tools used to develop this work was also necessary. This part of the thesis preparation can be summarized in the following tasks:

Extensive review of specialized bibliography on LPWAN networks [4] and MQTT [5]. The latter was already introduced to the author in a course completed in the last year of his degree.

In depth study of the available LPWAN technologies to carry out the objectives and selection of the most appropriate one, LoRaWAN [6].

**Proposal and practical approach to the technology:** Once the problem and its possible approaches were known, their feasibility and possibilities were studied. Based upon that, a solution was proposed. At this point the author had to get hands-on experience with the hardware and software selected to realize the project's goals. This part of the thesis preparation can be summarized in the following tasks:

1. Selection and documentation of the most feasible and cost-effective LoRaWAN project to base this thesis on, ChirpStack[3]. The project's architecture and characteristics have been documented.

2. Subsequently, the ChirpStack network used to test and develop the thesis had to be built. The author achieved expertise in the network's architecture to set it up and access its configuration from an external computer. The required hardware to build the LoRaWAN network was provided by ITI and the entire process was carried out in the author's home.
3. Testing end device registration and activation in ChirpStack networks. The process followed by the Application Server to complete the actions was reverse engineered through repetition using several tools such as Google Chrome Developer Tools and Postman. A MQTT client was also scripted using python to detect any message in the network.
4. Lastly, the planning of the steps required to build the application was carried out. This will consist of two main phases. First, the core functionality of the application, which is the API that automatically connects and activates an end device. After that part is finalized and tested, the development of the web application which will make use of it will be carried out.

**Development of the project:** This part of the thesis is the one that required more dedication and time. Following the detailed planification of the phases required to complete this project, the development of the application was carried out. The process can be divided in two big tasks:

1. The first step was to build an API to perform the registration and activation of an end device automatically given its joinEUI and devEUI. It consists of a python script and was tested using Postman and creating virtual and real nodes in the available ChirpStack Network.
2. Coding the web application to host the finalized API and surround it with a user interface to open its usage to as many people as possible. Therefore, minimalism and ease of use have been of particular importance to provide a practical and simple tool.

**Writing and revision of the thesis.**

## 1.3 Structure of the thesis

This section summarizes the structure of the thesis. This is also reflected in the content index.

Chapter 1 is the introduction to the work. Sections 1.1 and 1.2 serve as a preface, defining some of the concepts that will appear recurrently in the document, as well as a declaration of intent, exposing the motivation of the work and some considerations, such as the methodology followed. Next, Section 1.4 summarizes the objectives of the present work, separating the general ones from those related to sustainable development. Finally, Section 1.5 is a review of the state-of-the-art in the specific task of automatically connecting and activating end nodes to a LoRaWAN network.

Chapter 2 develops the theoretical framework of the thesis. An introduction to LPWAN and LoRaWAN is provided. In this context, after a succinct overview of LPWAN, the author focuses on its use cases and the technological alternatives that currently exist to implement such networks. One of them is LoRaWAN, which is subsequently studied in the depth that is required to comprehend this thesis.

Chapter 3 defines the methods used to accomplish the project's objectives. Firstly, the general project design is put into perspective. That is, which LoRaWAN technology is our work applied to and which part of the network is affected by it. Next, software and hardware used during the preparation of the thesis are presented. Finally, the selection of the Django framework for python backend web development is explained. Furthermore, key concepts of application development that are important for the project are introduced.

## 1.4 Objectives

This section presents the general objectives of the work. These serve as a guide and are directly related to the conclusions stated in Section Conclusions and future work. The main objectives of this thesis are as follows:

- To study the importance of IoT technology in today's economy and demonstrate how its implementation could help the small and medium size businesses. Deepen in the analysis of LPWAN networks and how they are used nowadays.
- To define the advantages and disadvantages of LoRaWAN and how it compares to other popular LPWAN technologies. Provide information about open-source projects based on this technology.
- To understand how ChirpStack networks work. Emphasize in the mechanism used by the application server to allow the connection of an end device.
- To develop an application to automatically connect a new end device to a working ChirpStack network.
- To provide a tool that facilitates the deployment of LoRaWAN technology in a mobility environment by technic and non-technic users.
- To document the application appropriately to allow this thesis to serve as the groundwork for the development of similar technologies in the future.

Furthermore, it is relevant to highlight objectives directly related to sustainable development. Since the present project aspires to have a real application, it is necessary to establish how it is going to favor the economy and society:

- Favor the automatization and subsequent optimization of large areas through lowering the technical barriers present today.
- Democratize the internet of things technology, especially LPWAN, among businesses with less technical resources.

## 1.5 Sate-of-the-art review

In the field of LPWAN, the LoRa Alliance has published recommendations as a baseline for the development of automated deployment of LoRaWAN end-devices through QR code detection [7]. Mandatory values, optional extensions and data organization are recommended. Following this, The Things Industries, a reference in the open-source LoRaWAN community with The Things Stack (TTS) LoRaWAN network server, has developed a QR-centered system that automates the claiming of devices.

If a device has been set up to be claimable in a TTS application, someone else can move the device out of it in a secure and legal manner. To do this, the device needs to be added first in TTS and then registered on The Things Join Server, without linking it to a Network Server or Application Server. Once this is settled, the claiming settings can be set up using TTS console or CLI. The environment also allows the automatic creation of a device-personalized QR-code to industrialize this procedure.

At the same time, TTS has added API endpoints to perform actions on the network directly via the API. For that, the user needs to make calls to the Identity Server, the Join Server, the Network Server, and the Application Server to create a device using the Over-The-Air-Activation procedure. A similar procedure, but simpler, is also possible to register a gateway.

This thesis aims to achieve a similar level of automation as the claiming mechanism but focused on the registration and activation of an end-device via the API, as the creation mechanism in TTS allows.

# Chapter 2

## Technological framework

### 2.1 LPWAN Technology Selection

The first and more important task to begin this project is the selection of the low-power long-range technology most suited to the thesis' goal. Multiple competing LPWAN technologies are being developed. One of them must be chosen to carry out the proposed study of a LPWAN deployment in an industrial mobility scenario.

An overview of existing LPWANs can be started dividing them into two groups, Infrastructure Based LPWAN Technologies and Infrastructure-less LPWAN Technologies. The first group comprises LTE Cat M1 and 5G, among others. They do not serve our purpose, since we aim to provide a tool which can be used in as many scenarios as possible, while these make use of infrastructure which many places lack. Therefore, infrastructure-less LPWAN Technologies are the way to go.

Regarding their popularity and world-wide usage, the most important Infrastructure-less LPWAN Technologies are LoRa, Sigfox and RPMA (Ingenu). To select the communications protocol to base this project on, a brief analysis of their main characteristics is presented.

#### 2.1.1 SigFox

The SigFox network provides IoT devices with a low-power low-speed data connection in selected countries around the world. It is a private-initiative paid public-access network. This means that the infrastructure (for example gateways) is not dedicated to only one customer, although the network protocol and architecture can be used for private LAN networks. Apart from its own deployments, SigFox licenses other companies (operators) to provide coverage in countries that they do not reach with their own.

The SigFox network uses ISM bands to communicate, employing the 868 MHz band in Europe and the 902 MHz band in the US.

### Technical overview

Sigfox uses D-BPSK (Differential Binary Phase-Shift Keying) UNB (Ultra Narrow Band) modulation in both uplink and downlink transmissions. The bitrate of its transmission can be either of 100 bps or 600 bps, depending on the region of operation. Uplink (Mobile Station to Base Station) messages are of a maximum length of 26 bytes, conformed by a 14 bytes protocol header and a 0-, 4-, 8- or 12-bytes user data payload. A payload of 0 bytes is used as a heartbeat. The current Sigfox regulation limits the amount of uplink packets to a maximum of 140 messages per day. These messages are usually sensor information. Downlink (BS to MS) messages can handle up to 8 bytes of user data. The current Sigfox regulation limits the amount of downlink packets to a maximum of 4 messages per day. These are usually control messages issued by the owner or operator of the devices.

The architecture is composed by 4 types of devices:

- The Objects: sensors or devices that capture information and communicate it to the Network.
- The Sigfox stations: gateways that connect Objects to the Sigfox cloud through an IP network (the Internet).
- The Sigfox cloud servers: The routers of the network. They direct uplink and downlink traffic between the Objects and the Customer's IT servers and are responsible for the coordination of the network.
- The Customer IT servers: Application servers used by the customer to store and process the data collected by the Objects associated with them.

### Usage, Availability and Cost

Prices for communication hubs (end nodes/objects) compatible with Sigfox are as low as 20\$ and provide full access to the communication capacities of the network. Multiple real-life Sigfox projects and implementations can be found on its official web page.

### 2.1.2 RPMA (Ingenu)

Random Phase Multiple Access (RPMA) technology is a communications stack solution (from the physical layer to the MAC and network) provided by Ingenu, an American company.

RPMA is a technology designed specifically for the IoT wireless machine-to-machine (M2M) ecosystem which and, unlike other low-power networks, uses the popular 2.4 GHz band available around the world. It's able to offer low-power, wide-area coverage using the band's minimum duty-cycle.

The spreading technique used by RPMA is the Direct-Sequence Spread Spectrum (DSSS) modulation scheme for uplink communication, while downlink packages are transmitted using CDMA.

Ingenu's RPMA networks present a star topology [Ingenu, Inc., "The Making of RPMA," Ingenu, 2016]. End nodes communicate with the base station, which in turn communicates with the backend of the network via gateways to orchestrate the network's behavior. Packet sizes range from 6 bits to 10 kbits, thanks to Ingenu's RPMA flexible packet size.

#### Usage, Availability and Cost

RPMA is focused on scalability and coverage, being able to support an ever-increasing number of connected devices with enough capacity in the network while optimizing the infrastructure usage with extra broad coverage.

Some RPMA networks have been deployed (mostly in America), but its high scale world-wide deployment is still pending, and it is not sure that there will be any, because RPMA has not taken over its competitors in its 10 years in the market (by 2018). Development kits are available from Ingenu, Inc. The cost of the development kits is not specified in their webpage.

### 2.1.3 LoRaWAN

LoRaWAN is a wide-area networking protocol and architecture for long range IoT communications which makes use of the LoRa and FSK radio modulations. The star topology of a LoRaWAN network is based on gateway to end-node communication, like the mobile-station to base-station GSM scheme. As this topology suggests, gateways and end-nodes in a LoRaWAN network are not capable of peer-to-peer communication.

LoRaWAN defines different data rates depending on the band of operation and region. Different spreading factor-bandwidth combinations are implemented to achieve different bit rates. The use of FSK modulation instead of LoRa is also supported to allow a data rate of up to 50 kbps.

The application payload for the lowest data rates is limited in Europe by the European Telecommunications Standards Institute (ETSI) to a maximum of 51 bytes per packet. For the highest transmission rates, the application payload can reach up to 222 bytes.



A LoRaWAN network has the physical layer, which includes telematic transmission in the 863 to 870 MHz band (in Europe), MAC layer and application layer. The tasks to be carried out in each of these layers are distributed among different hardware and software components. The scheme is presented in Figure 1.

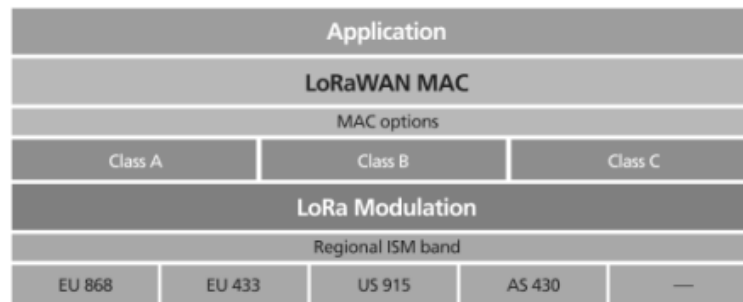


Figure 1 - Representation of the LoRaWAN architecture

### Usage, availability, and cost

LoRaWAN networks were already deployed and are working in multiple regions. LoRaWAN networks, due to LoRaWAN openness to custom implementations, can be built using any device capable of hosting the Network Server and communicate via radio using LoRa and FSK modulations.

#### 2.1.4 DASH7

The DASH7 Alliance Protocol is the name of the technology promoted by the non-profit consortium called DASH7 Alliance. It is an open-source network protocol with range of up to two kilometers, which is characterized by its low latency, the multiyear battery life of its wireless sensors and a very small open-source protocol stack [8]. Unlike most LPWAN technologies, DASH7 supports tag-to-tag communications. It operates in the 433 MHz, 868 MHz, and 915 MHz unlicensed ISM band. DASH7 networks are ideal for low power usage and sporadic data transmission. With this niche in mind, the concept of B.L.A.S.T. was designed.

- **Bursty:** Data transfer is small and abrupt, perfect for sensor applications.
- **Light:** For most applications, packet sizes are limited to 256 bytes.
- **Asynchronous:** DASH7's main method of communication is command-response, which by design requires no periodic synchronization with the network.
- **Stealth:** DASH7 devices do not need periodic beaconing to be able to respond in communication.

- Transitive: DASH7 is upload centric. Thus, devices do not need to be managed extensively by fixed infrastructure.

### Usage, availability, and cost

DASH7 implementations can be found in applications that require modest bandwidth like text messages, sensor readings, or location-based advertising coordinates. DASH7 components are inexpensive, with single chip silicon purchased in volume for 5€ each and remaining components available for even less. However, the networking stack solution OpenTag is not as read-to-use as some other LPWAN alternatives.

## 2.1.5 Conclusion

Ideally, this project could be based on any of the three technologies presented in this section. However, the real difference between them in the scope of this thesis is their availability and easiness of implementation. LoRaWAN and Sigfox networks can be deployed with few resources. LoRaWAN has a very rich open-source community. Some of them offer a ready-to-use all-in-one package to deploy a LoRaWAN network on a Raspberry Pi. Therefore, LoRaWAN is the chosen LPWAN communication protocol.

## 2.2 LoRaWAN

After the selection of LoRaWAN as the LPWAN protocol of choice for this work, this section presents the LoRaWAN characteristics that affect this thesis. LoRaWAN 1.0.2 is has been used to develop the project and the following explanations always refer to that version.

The key aspects of LoRaWAN that are of interest to this thesis are the network's topology and the procedure of activating end-devices.

## 2.2.1 LoRaWAN network topology

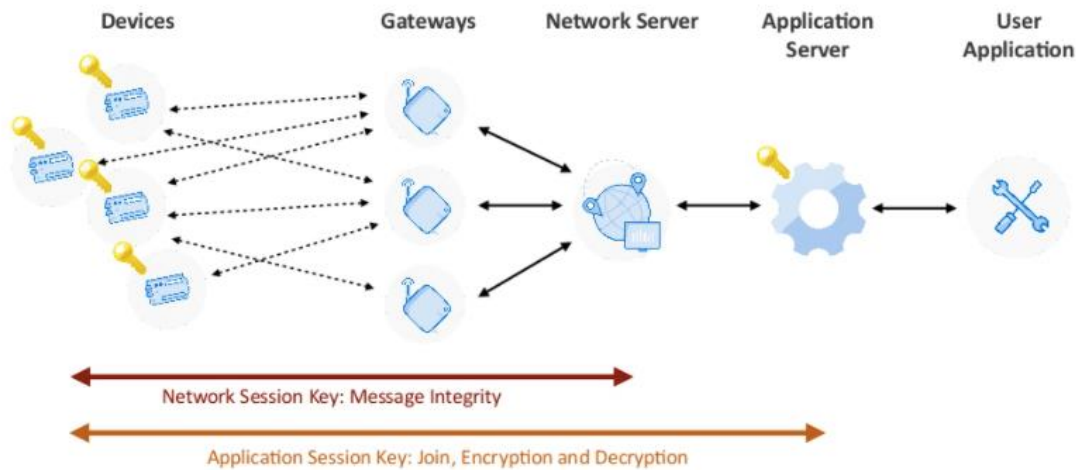


Figure 2 - LoRaWAN network topology

It is necessary to detail the functions of each component to understand how these types of networks work.

End devices are of particular importance in this work. **LoRa End Nodes (devices)** are usually sensors. They can be of many types, depending on the use given to the network. Their task is to send the collected information to the LoRaWAN Network Server through one or more LoRa Gateways. The nodes are connected to the gateway that provides the best communication quality among the detected ones. In order to join the network and take advantage of the benefits of the protocol, the node must send a series of identification and security keys, which is explained in detail in the next section.

Depending on their capacity to receive and send information, which greatly affects their power conservation, LoRa End Nodes are split in three categories. They range from the extreme power-saving Class A nodes to the continuously alert Class C devices, which are more power hungry. A summary of the characteristics of each class of end devices:

- **Class A, device-initiated communication:** Devices are typically in deep sleep and send messages on intervals and/or events. Only after uplink transmission, there is a receive window for downlink messages. This is best for most sensor applications because of the increased battery conservation achieved throughout the deep sleep state.
- **Class B [9], time synchronized communication:** The network broadcasts beacons for devices to sync time. In so-called ping slots, devices wake up and the network may send downlink messages. This class is the preferred selection for most downlink intensive applications.

- **Class C, network-initiated communication:** The devices are continuously listening, often temporarily or on power supply. The network can send downlink messages at any given time. This approach is best for downlink intensive applications that require low latencies.

**LoRa Gateways** operate in the physical layer. In uplink communication, a LoRa gateway receives LoRa modulated packages from end nodes in its range, demodulates them and sends them to the Network Server through an internet protocol. This process is inverted to handle downlink communication. Currently, LoRa gateways are able to exchange information with up to 8 end nodes simultaneously.

The software responsible for receiving and sending the information is called Packet Forwarder. The most known implementations are the Semtech UDP Packet Forwarded and Semtech Basic Station Packet Forwarder.

A **LoRa Network Server** monitors the network's state. Therefore, it is considered one of the most critical components in a LoRaWAN network. Among others, a LoRa Network Server performs the following actions:

- Keep track of the gateways and end nodes connected to the network, their state (active, on hold, etc.).
- Activation and authentication of new end nodes that are to be connected to the network. This process is handled by a dedicated service, the Join Server.
- Uplink communication: Deduplication of identical information packages received from different LoRa gateways, which have probably communicated with the same end device because it is in both of their range areas. Transmission of the deduplicated data to the Application Server.
- Downlink communication: Keep the information packages that have been received from the Application Server in cue until they can be forwarded to the appropriate end nodes. This is extremely important in LoRa networks based on Class A or Class B end nodes.
- Authentication of the forwarded packages.

The **LoRa Application Server** is, as its name may suggest, the one responsible for the application layer in a LoRaWAN network. This makes it another critical component of a LoRaWAN network along with the Network Server. The Application Server receives and decodes information packages coming from the network server. Analogically, it encrypts the data it sends to the network server, which is therefore unable to understand the content of the packages it handles.

**User applications or end applications** connect with the Application Server to send information to the nodes. An end application usually serves data analysis and visualization, security or monitorization purposes, among many other.

## 2.2.2 Activation of an end device in LoRaWAN

To participate in a LoRaWAN network while maintaining security, an end-device must be personalized and activated. Activation of an end-device can be done in two ways. Over-The-Air Activation (OTAA) and Activation by Personalization (ABP), in which the end-device personalization and activation are done simultaneously.

After an end-device has been successfully activated, it stores the following information:

### End-device address (DevAddr)

The DevAddr is a 32-bit logical address that identifies the device within the network. It is used for all subsequent communication with the network.

Bit number	[31:25]	[24:0]
Components	NwkID	NdkAddr

*Table 1 - Meaning of the bits in the LoRaWAN DevAddr*

The most significant 7 bits form the Network identifier (NwkID) to separate device addresses of territorially overlapping networks of different network operators and to prevent roaming issues. The remaining 25 bits form the network address (NwkAddr) of the end-device, which can be arbitrarily assigned by the network manager.

### Application identifier (AppEUI)

The AppEUI is a global application identifier in IEEE EUI64 address space. It uniquely identifies the entity able to process the join request frame. The AppEUI is stored in the end-device before the activation procedure is executed.

### Network session key (NwkSKey)

The NwkSKey is a network session key specific for an end-device. It is an encryption key used by the end device and Network Server to calculate and verify the Message Integrity Code (MIC). It is also used to encrypt and decrypt payloads with MAC commands.

### Application Session Key (AppSKey)

The AppSKey is an application session key specific for an end-device. It is an encryption key used by the Application Server and the end-device to encrypt and decrypt application payloads in data messages to ensure message confidentiality.

### Activation by personalization (ABP)

Activation by Personalization directly ties an end-device to a pre-selected network. It requires hardcoding the DevAddr as well as the two session keys (NwkSKey and AppSKey) in the device. Therefore, an end device activated using the ABP method can only work with a single network, since it is only equipped with the required information for participating in that specific LoRa network. It also keeps the same security session for its entire lifetime.

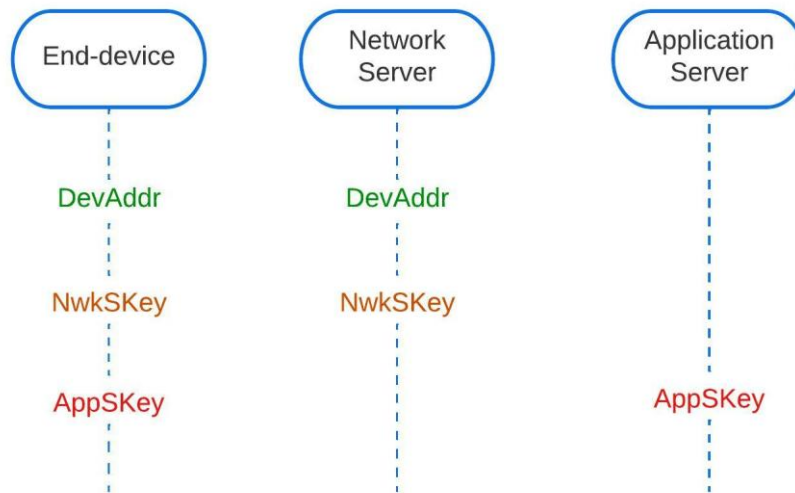


Figure 3 - pre-provisioned keys for LoRaWAN 1.0.x Activation by Personalization

The main advantage of activation by personalization is that the deployment of an end-device doesn't require any action after the preparation of the node. However, security concerns render this connection solution as the least favorable choice when adding an end-device to a network. The lack of flexibility of the end-device after being hardcoded with the device address and session keys is also a big downside of this method.

### Over-the-air activation (OTAA)

This is the most secure and recommended activation method for end devices. It consists of a join procedure between the end-device and the network, during which a dynamic device address is assigned to the end-device and the security keys are negotiated between the actors. The main advantage of the OTAA is security, as an end-device must go through the join procedure every time it loses the connection and therefore the session context information.

This join procedure requires two MAC messages to be exchanged between the end-device and the network server. The end-device initiates the communication with a join-request sent to the network server, which, after being processed by its recipient, should result in the respective join-accept message being sent back to the end-device. Before activation, apart from the AppEUI detailed before, the end-device also needs to be personalized with the following information:

- End-device identifier (DevEUI): Global end-device ID in IEEE EUI64 address space that uniquely identifies the end-device.
- Application Key (AppKey): Specific to an end-device AES-128-bit secret key known as root key. The AppKey is used to derive the session keys NwkSKey and AppSKey specific for that end-device. This allows the roaming of end-devices across networks of different providers.

The AppKey is never sent over to the network and, thus, remains secret. The AppEUI and DevEUI are not secret and are visible to everyone.

The following flowchart describes the message exchange between an end-device and the network and application servers of a LoRaWAN deployment.

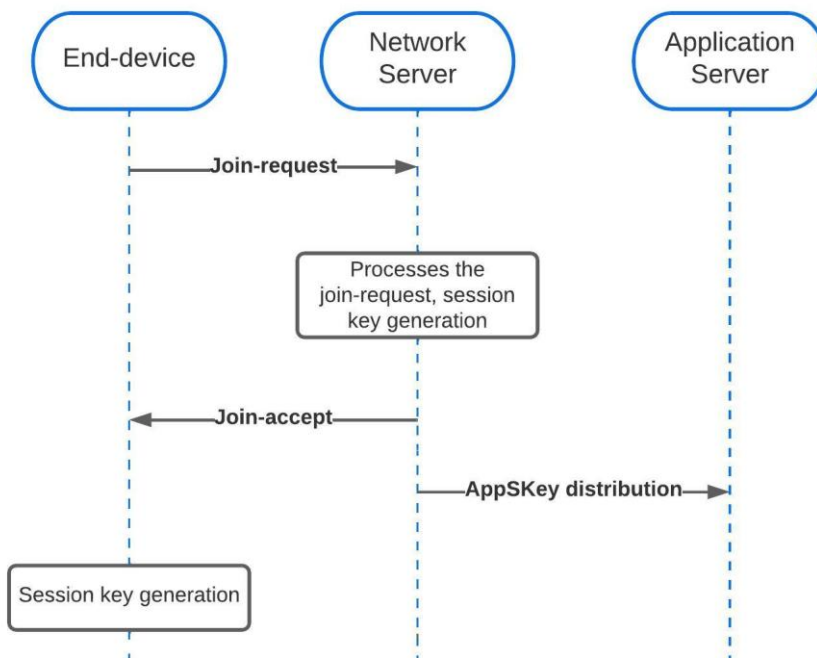


Figure 4 - Over-The-Air activation of LoRaWAN nodes, message flow between end-device, network server and application server.

As marked in Figure 4, the OTAA procedure comprehends mainly 5 steps:

Step 1: The node requests a join (or login) to the network with the configuration data and opens the reception window. The join-request message includes the DevNonce, a 2-byte value generated by the end device. The network server uses the DevNonce of each end-device to keep track of their join requests, rejecting one if a previous join request with the same DevNonce is already registered. As previously stated, the AppKey is not sent with the join-request message, as it is not encrypted.

The join-request message can be transmitted using any data rate and using one of the region-specific join channels. In Europe an end-device can transmit the join-request message by randomly choosing among the 868.10 MHz, 868.30 MHz, or 868.50 MHz frequencies.

<b>Size (bytes)</b>	8	8	2
<b>Components</b>	AppEUI	DevEUI	DevNonce

*Table 2 - Bytes and components of the join-request message*

Step 2: The network server processes the incoming join-request message. It generates two session keys (NwkSKey and AppSKey) and the join-accept message if the end-device is permitted to join the network. This message has the following fields:

- The AppNonce: A random 3-bytes value. It is used by the end-device to derive the two session keys.
- The NetID: The most significant 7 bits of this field are occupied by the network identifier (NwkID)
- The DevAddr, as explained above.
- The DLSettings, 1 byte containing the downlink settings which the end-device should adapt to.
- RxDelay, which contains the delay between TX and RX.
- CFList, a list that contains region-specific channel frequencies for the network the end-device is joining.

<b>Size (bytes)</b>	3	3	4	1	1	(16)
<b>Components</b>	AppNonce	NetID	DevAddr	DLSettings	RxDelay	CFList (optional)

*Table 3 - Bytes and components of the join-accept message*

Step 3: The network server calculates a Message Integrity Code (MIC), which is added to the join-accept message. This message is then encrypted with the AppKey and sent back to the end-device as a normal downlink message. If the join-request message was not accepted, no response is given by the network server.



Step 4: The network server keeps the NwkSKey and shares the AppSKey with the application server.

Step 5: Upon receiving the join-accept message, the end-device uses the AppKey and AppNonce to derive the network session key and the application session key. It is now activated on the network. After activation, the required information (DevAddr, NwkSKey and AppSKey) is stored in the end device.

## 2.3 LoRaWAN technology selection

There are a lot of LoRaWAN technologies, which offer network architecture, protocol, or partial implementations. However, two open-source technologies stand out from the rest, ChirpStack and The Things Stack. These two grow in popularity every passing year and offer a full stack LoRaWAN implementation, which means that a network deployment can be achieved without heavy development. Their popularity also gives developers the chance to exchange ideas and knowledge with thousands of developers, casuals, and professionals.

On the one hand, ChirpStack and The Things Stack share a lot of similarities. Both are complete and open-source LoRaWAN solutions designed for various deployment scenarios, supporting all existing LoRaWAN versions, operation modes A, B and C, and all regional parameters as released by the LoRa Alliance. Furthermore, these technologies can be implemented in an industrial environment and be used for business.

On the other hand, ChirpStack and The Things Stack differ in one very important aspect. While every component of a ChirpStack network can be installed separately and privately, The Things Stack offers an enterprise grade LoRaWAN Network Server based on an open-source core. The Things Network is a free LoRa Server environment in the cloud, building a community in which members agree that their infrastructure (gateways) can be used by anyone. This option is not recommended for commercial deployments since capacity for end-devices is not guaranteed. The Things Industries is the commercial offering of The Things Network. Here, the network elements are not shared, and members can get expert consulting to achieve their goals with The Things Stack. ChirpStack is a more economical solution, but one that requires more expertise because every element of the network must be set up and configured before deploying.

As the technical part of the network is not a problem in this thesis, ChirpStack is the chosen LoRaWAN technology. It offers a full stack solution but gives all the freedom you need to set up a custom network to develop and test a project of this scale. The configuration possibilities and the ever-growing integration

opportunities make ChirpStack a great alternative to the more popular The Things Stack, in particular for this work.

Lastly, ChirpStack is more appropriate for the scope of this project since it can be directly installed in the gateway itself. This way, the project can be done with only one Raspberry Pi 3 hosting every LoRaWAN network component needed to build a private testing network.

## 2.4 Web development concepts

This section serves as an introduction to core concepts of web application development used for the completion of this thesis. Web development has been on the rise since the world-wide web's potential was recognized. From video streaming services on the web to simple informative sites without an active task, internet hosts more information every passing day.

Countless technologies and application architectures have been used to develop said web services. An Application Programming Interface, usually known by its initials as API, allows different software components to be connected and share information and functionalities between them through a secure, flexible, and efficient integration.

This thesis aims to streamline the task of managing LoRaWAN Network's components in a mobility environment. An application to automate the process needs to be created. Its design needs to be lightweight and in accordance with its functionality.

This section defends the selection of a specific architecture for the API created in this work and the file format used to allow its communication with the LPWAN service of choice, the LoRa Network Server of ChirpStack.

### Representational State Transfer API

A Representational State Transfer (REST) API is a very popular API category because it does not establish any specific protocol as, for example, the Simple Object Access Protocol API does. REST-style APIs are data-driven, not functionality-driven, which is perfect for replicating the MQTT message fields that are performed for node registration.

REST establishes a series of principles that the API architecture must follow and allows developers freedom to choose the type of file to work with for the exchange of information and the corresponding transmission protocol. APIs designed following the REST principles, stated below, are called RESTful APIs.

1. **Client-Server architecture:** The REST principle works on the concept that client and server should be isolated from one another and permitted to develop independently. This way, you can improve manageability across numerous platforms and increase scalability y streamlining server components as user interface concerns are separate from the data storage concerns
2. **Stateless:** RESTful APIs, are stateless, which means that API calls are independent of each other. They contain all the information necessary for the server to carry out the work. Without server storage, the API is more scalable, following the trend presented in point 1 of this list.
3. **Cacheable:** Data within a response should be indirectly or clearly categorized as cacheable or non-cacheable. If a response is cacheable, the client cache is provided the right to recycle the response data for similar requests in the future. As a stateless APIs usually handle a great amount of inbound and outbound calls, a REST API should be able to store cacheable data.
4. **Layered system:** A REST API's architecture includes several layers that operate together to construct a hierarchy that helps generate a more scalable and flexible application. Each layer has its own tasks and functionality and can only interact with the subsequent one. This characteristic has the following advantages:
  - a) Individual components of the application can be specially protected based on their frequency of use, criticality, or exposure to other services.
  - b) One part of the application can be modified without affecting the overall functionality of the API, which adds longevity and flexibility to the application
  - c) Security is not based on an individual solution, but on multiple security components on several layers which work together and cover each other in case of failure. This helps to prevent attacks aimed at the server architecture.
5. **Uniform interface:** This characteristic allows client and server to communicate and understand each other, regardless of the back-end language used to create them. This is usually achieved through the usage of URI, CRUD (Create, Read, Update, Delete) and JSON resources.
6. **Code on demand:** This REST principle allows for coding scripts or applets to be communicated through the API used within the application. Usually, a server returns static resource representation in XML or JSON format, but when required, servers can also deliver executable code to the client. This extends client functionality by downloading and implementing coding which decreases the number of features to be pre-implemented.

## Parameters in an API

Parameters are options you can pass with the endpoint, such as specifying the response format, or the amount returned, to influence the response. There are several types of parameters in a HTTP request: header parameters, path parameters, and query string parameters. Request bodies are closely like parameters but are not technically a parameter.

HTTP requests have different “methods”. The most popular ones are GET, POST, PUT and DELETE.

- The POST method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and receives back a response.
- The GET method bundles the submitted data into a string and uses this to compose a URL. The URL contains the address to which the data must be sent, as well as the data keys and values.

Any request that could be used to change the state of the system -for example, a request that makes changes in the database- should use POST. GET should be used only for requests that do not affect the state of the system.

GET would therefore be unsuitable for a password form because the password would appear in the URL, and thus, also in browser history and server logs, all in plain text. Neither would it be suitable for large quantities of data, or for binary data, such as an image. A Web application that uses GET requests for admin forms is a security risk: it can be easy for an attacker to mimic a form’s request to gain access to sensitive parts of the system. The POST method offers more control over access.

On the other hand, GET is suitable for things like a web search form, because the URLs that represent a GET request can easily be bookmarked, shared, or resubmitted.

### A) Parameters as custom headers

The HTTP specification states that headers are like function parameters [10].

On the one hand, parameters that stay the same on all endpoints are better suited for headers. For example, authentication tokens get sent on every request.

On the other hand, parameters that are highly dynamic, especially when they're only valid for a few endpoints, should go in the query string. For example, filter parameters are different for every endpoint. Furthermore, adding a query string to an URL is quickly done and more obvious than creating a custom header.

### B) Parameters in the query string

If we know that the parameters which we want to add don't belong to a default header field and aren't sensitive data, the query string may be a good place for them.

Therefore, the main use-case of the query string is filtering, mostly two special cases of filtering: searching and pagination. But as repurposing for web-forms shows, it can also be used for different types of parameters. A RESTful API could use a POST or PUT request with a body to send data to the API.

However, the query string is part of the URL, which can be read by everyone between the client and the API, so sensitive data like passwords shouldn't be put into the query string.

## JSON

The open JavaScript Object Notation file format, known by its acronym as JSON, is the most popular and suitable option for this project. The syntax of a JSON file is like dictionaries in programming languages, as it is made up of variable's name-value pairs. Thus, it is ideal for conveying the name of a variable or field and its content. The following code snippet shows an example used in our API.

```
body_app = {"application":{
    "serviceProfileID":details['serviceProfileID'],
    "name":details['name'],
    "organizationID":"1"
}}
id_app = post_request(url=url, body=body_app, headers=headers).json()['id']
```

*Code 1 - JSON format*

The combination of a REST API and JSON file format makes it possible to reduce the bandwidth required in transmission as much as possible, in line with the low consumption of the LoRaWAN network. JSON files are extremely lightweight, in exchange for losing some of the security offered by other formats, such as XML.

Finally, it should also be noted that it is very comfortable for humans to read and interpret JSON files, which facilitates the field and value identification tasks carried out during the preparation of this thesis.

## Chapter 3

# Methodology

For the development of this thesis, a LoRaWAN network will be used based on the open-source components offered by the ChirpStack project. To develop the project, a LoRaWAN network will be used based on the open-source components offered by the ChirpStack project.

To achieve the goal of automatically connecting a new end-device to an existing LoRaWAN ChirpStack network, the following steps must be followed:

1. Understanding the network: At which point of the LoRaWAN network does the application need to act? A deep understanding of the ChirpStack architecture is needed.
2. Understanding the task: How can the application perform the task assigned to it? A process of reverse engineering the steps followed by the ChirpStack network in the OTAA process can be used to understand how everything works.
3. API scripting: Once the process is clear, the API that facilitates it needs to be scripted. The starting point for this API is knowing the individual data about the end-device and the network server, being the goal to connect them via HTTP requests.
4. Application design: The API can't be used in its raw form. An application must be designed to host the API as a process that can be activated by a user. The network server to be accessed and the end-device to be connected to it will be variables that the user will enter in the application.

In sections 3.1 to 3.4 the methodology followed to complete each step of the design process is explained.

### 3.1 Understanding the network

The network architecture used to develop the application follows ChirpStack’s network architecture. It includes the ChirpStack Application Server, the ChirpStack Network Server, the gateways, and the end-devices in a typical star-shaped architecture.

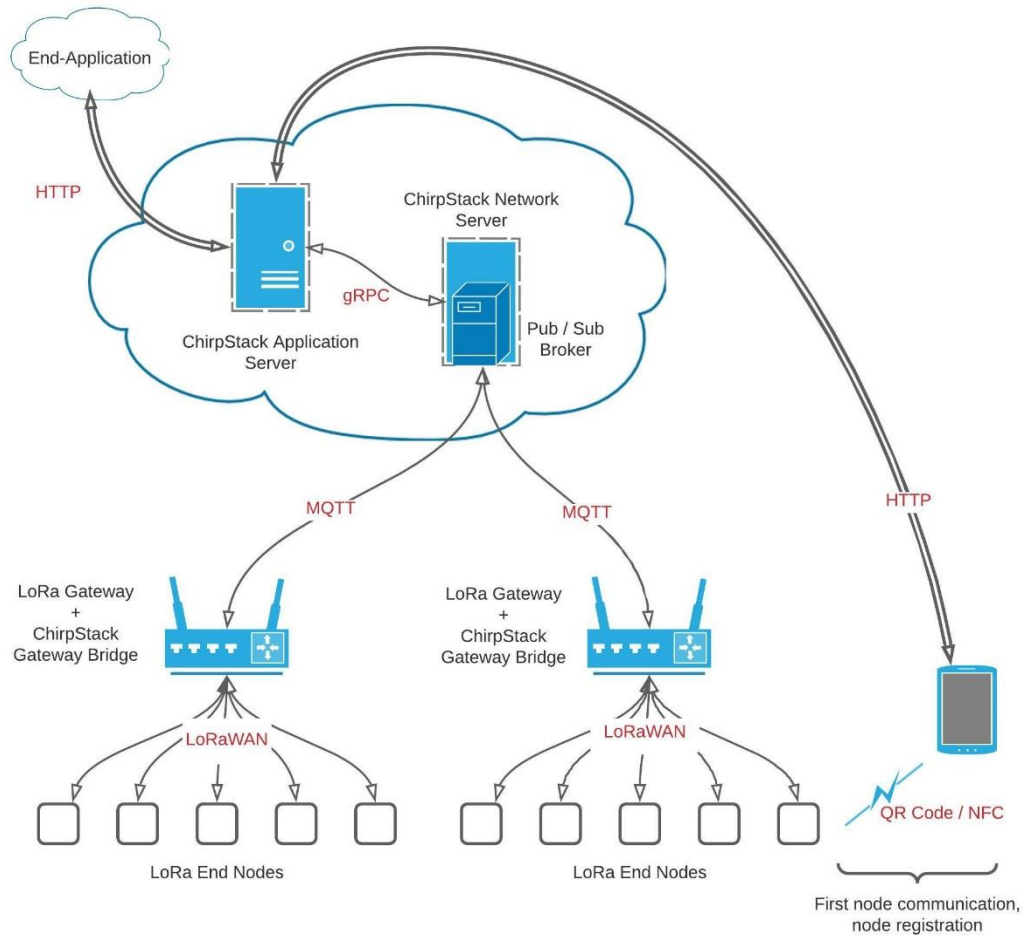


Figure 5 – LoRaWAN Network architecture used in this thesis

Figure 5 represents the network architecture that results from this discussion. It includes the ChirpStack network and the application’s role in it represented by a mobile phone.

The application, as the schematic represents, will interact with the end-device and with the Network Server to register and activate the first in the second's network. Of course, the activation procedure of the end-device must be Over-The-Air Activation, as stated in section 2.2 LoRaWAN.

This means that the application must know details about the end-device to start the OTAA process in its name, which it needs to get from an external source. Since the goal of this project is to reduce manual work in the activation of a new end-device, the introduction of the DevEUI and AppKey must occur automatically in a “reading” process performed by the mobile phone. This reading can be done in one of two ways. The end-device must be equipped either with a QR code or with an NFC card. For this project the QR code mechanism is ideal since it requires very little investment, in any case lower than a NFC system. QR codes can be added to existing devices, making any IoT end-device compatible with the application. Furthermore, a QR code is the superior choice in a testing environment like this because of its flexibility and the ability to redo it a virtually infinite number of times. However, due to the small size of some IoT end-devices and their possible exposure to climate and the passing of time, the transmission of the end-device's characteristic data via NFC could be advantageous in some environments.

Once established how end-device and application will interact, the connection to the network server remains. This will be done by performing HTTP requests to the Application Server. Thus, knowing the IP address of the network server is a must. Furthermore, since ChirpStack has a security system, authentication data is needed. This is represented by a username and its corresponding password.

Figure 5 also reveals a component of the ChirpStack network not described in the LoRaWAN introduction, the ChirpStack Gateway Bridge. It sits between the Packet Forwarder (software running on the LoRa Gateway responsible for receiving and sending data) and the MQTT broker. It transforms the Packet Forwarder format (like the Semtech UDP Packet Forwarder Protocol) into a data-format used by the Chirp-Stack components, like JSON.

### 3.1.1 Test suit and development platform

The LoRaWAN network used as test and development suit is centered around a Raspberry Pi 3 flashed with ChirpStack OS, a Linux-based OS to run the ChirpStack stack on a Raspberry Pi based LoRa gateway. Furthermore, the network contains a LoRaWAN class B node built by PyCom and formed by Pysense [ (PyCom, 2017)] hardware and a FiPy [ (Pycom, 2017)] board.

Furthermore, Bootstrap 5.0 has been used to help the author to efficiently design the web application's frontend and allow him to focus on backend development.



### A) Installing ChirpStack OS and accessing the network

The full version of ChirpStack OS (there is also a lighter one) provides a full ChirpStack Network Server and ChirpStack Application Server environment running on the gateway, on top of the Concentrator and Gateway Bridge usually installed in gateways. It also includes a CLI utility for gateway configuration, which is used for this work.

ChirpStack OS was flashed into the 16 GB microSD card that Raspberry Pi equips using balenaEtcher [11]. Next, the Raspberry Pi was booted up and connected to the internet via an ethernet cable. This option is better than Wi-Fi for dynamic testing because the IP address will not change if the device is reconnected to the same ethernet port again. The IP address assigned to the Raspberry Pi can be looked up accessing the network's router configuration through an internet browser such as Google Chrome.

Secondly, the user interface to change ChirpStack OS settings can be accessed from a computer in the same network through a SSH connection. The process to successfully set up the network is the following:

1. Setup LoRa concentrator shield: Select for the available hardware, which is “Sandbox: LoRaGo PORT”, the frequency band to be used in LoRa transmission.
2. Edit ChirpStack Concentrator configuration file: Put the *reset\_pin* argument equal to 25 to enable correct motherboard reboot in the Raspberry Pi 3 chosen for this project.
3. Edit ChirpStack Gateway Bridge configuration file: set the marshaler to *json*, the file format that has been selected to prepare this thesis. The other option is protobuf encoding.
4. Restart ChirpStack Concentrator.
5. Restart ChirpStack Gateway Bridge. If a user needs to enable Wi-Fi connection, it must be done at this point.
6. Set admin username and password.
7. Flash concentrator MCU.
8. Reload Gateway ID.

At this point, the ChirpStack Application Server can already be accessed from a computer in the same network, using a web browser such as Google Chrome and typing in the search bar the Raspberry's IP followed by the port that wants to be accessed, in this case the 8080 (can be configured, this is default).

### B) Configuration of the network

However, the network is not ready yet. If we access the application server, we don't see any active network server. We need to connect it. We need to enter its IP address accessing the configuration as

before through a SSH connection. As it is in the same address but a different port, a valid configuration would be the following.

```
[network_server.api]
# ip:port
bind="0.0.0.0:8000"
```

*Code 2 - API configuration, shell*

When entering the address in the Application Server to bind them, “0.0.0.0” needs to be replaced by “localhost” for them to correctly bind.

The next step is to add the Gateway via a Gateway-profile configuration, using the recently bind Network Server. Once this is done, a service-profile must be created, entering the recently bind Network Server and two fields describing the maximum and minimum allowed data rate. As per ChirpStack documentation, a zero must be entered in the “Minimum allowed data rate” and a four in the “Maximum allowed data rate”.

After that, the network is ready to receive its first end-device, which will need to be added to a device profile and to an application previously created.

A device profile establishes a common configuration shared by every device that is going to be added to the network using this device profile. The most important values to add to the device profile are:

- The LoRaWAN MAC version and the Regional Parameters that are supported by the end-devices. In this thesis, LoRaWAN 1.0.2 B is used.
- The uplink interval. This establishes the frequency of the status report messages sent by the end-devices to the gateway, which uses them to know if a device is active or not. Setting this value as low as 30 seconds allows for better testing speed.
- Join procedure, OTAA or ABP. We are going to use OTAA exclusively.
- Class B and Class C supporting end-devices need to be notified at this point. As the end-device used in this thesis is Class A, the device profile is configured appropriately.

An application in ChirpStack’s Application Server is a purpose given to a group of end-devices. It doesn’t need any special configuration and doesn’t server any purpose apart from an organizational one.

### C) Adding and activating a device

A device must be added inside an application to ensure its an organizational position in the network. To create it, the user must manually enter the Device Identifier and the Application Key, which for our end-device are:

```
app_key = '11B0282A189B75B0B4D2D8C7FA38548B'  
Device EUI = '70b3d549912693f3'
```

*Code 3 - keys of the end-device used in this thesis*

Furthermore, a device profile to assign the end-device to is also mandatory.

Once activated, the Network Session Key and the Application Session Key are shown with the Device Address.



*Figure 6 - Screenshot of the keys of an activated end-device in the Application Server.*

A device address is created, formed by a 8-digit hexadecimal identifier. An uplink and downlink frame counter monitors the number of LoRaWAN frames exchanged in each direction.

## 3.2 Understanding the task - Reverse engineering

Reverse engineering is a process in which a product is deconstructed to extract design information from it. Often, reverse engineering involves deconstructing individual components of larger products. The reverse engineering process enables you to determine how a part was designed so that you can recreate it. This is exactly what needs to be done. The examination of the process followed by the Application Server to register and activate the end node will allow its recreation and automation using an API.

The testing and analysis presented in this chapter have been obtained using virtual and physical nodes.

The first hypothesis is that the Application Server sends an HTTP request to the Network Server when an action is performed on its user interface. Google Chrome's developer tools can be used to monitor every HTTP request sent and received by its user. Using this method, it has been proven that the registration and activation of an end device is done through HTTP requests. In this case, the requests are sent to an URL inside the same local network, which has been used during the entire project as the testing and development site and hosts the Raspberry Pi.

The analysis of the request to register a new end-device throughs the results shown in Table 4. As an example of local IP address that hosts the Raspberry Pi, "192.168.0.208" is used:

Request URL	http://<url>/api/devices
Request method	POST
Request payload	<pre> {   "device": {     "applicationID": 1,     "devEUI": '70b3d549912693f3',     "deviceProfileID": 1,     "isDisabled": False,     "name": 'testing',     "skipFCntCheck": False   } } </pre>

Table 4 - Characteristics of the request to create a new end-device in the network.

After testing various scenarios, the Network Server can return the following responses:

- If the DevEUI is already registered or there already is a device with the same name:

```

{
  "error": "object already exists",
  "code": 6,
  "message": "object already exists",
  "details": []
}

```

Code 4 - response after error

- If the application ID or device profile used to create the device don't exist:

```

{
  "error": "object does not exist",
  "code": 5,
  "message": "object does not exist",
  "details": []
}

```

Code 5 - response after error code 5

- If the device is successfully registered, a response without payload is received.

Similarly, an analysis of the device activation request shows the following results:

Request URL	http://<url>/api/devices/<DevEUI>/keys
Request method	POST

Request payload	<pre> {"deviceKeys":{   "nwkKey": "11B0282A189B75B0B4D2D8C7FA38548B" } }                 </pre>
-----------------	---

Table 5 - Analysis of the request to activate an end-device

The possible responses by the Network Server are the same ones as when registering the end-device. That is error code “6” when the device has already been activated and error code “5” when the device does not exist (when the application or the device profile did not exist in the registration).

This means that ChirpStack Application Server API is handling every action, and thus can be accessed by our API to perform the actions that we need and build integration. This API is based on the gRPC (<https://grpc.io/>) remote procedure call (RPC) framework but includes a RESTful JSON API interface provided by an embedded 'Restful HTTP API to gRPC' proxy. The latter is more convenient and easier to use for simple use-cases like our API. Thus, the Gateway Bridge has been configured to encode events using 'JSON', changing the default 'protobuf' configuration.

Using the Application Server, this are the steps and requirements to create and activate a device in a working ChirpStack network:

1. Create an *application*, which in ChirpStack is used as a way of organizing end-devices.
2. Create a *device profile*, which is a set of end-device configuration parameters.
3. Create a *device* inside an available *application*, assigning it to a *device profile*.
4. Activate the *device*.

How each step is performed has been discovered also using reverse engineering. After a thorough testing and analysis using Google Chrome developer tools and Postman, the requirements and formalities of the process are known.

The first and more important parameter is the Grpc-Metadata-Authorization, which is needed for every call to the Network Server’s API. The *limit* parameter is also necessary for every Get Request and is therefore included in Table 6

Parameter	Characteristics	Error message
Grpc-Metadata-Authorization	mandatory, string	‘Authorization failed’
Limit	mandatory, integer	Does not return any item of the requested list.

Table 6 - Necessary parameters, Network Server API

The following code snippet shows the structure needed to perform Get and Post Requests to the Network Server. One thing to notice is that the body of a post request to the Network Server needs to be in JSON format. The response's body is always JSON formatted as well, so it must be reformatted into a normal python list.

```
url = 'chirpstackurl'
API_TOKEN = 'API_TOKEN_EXAMPLE'
auth_token = 'Bearer %s' %API_TOKEN
headers = {'Grpc-Metadata-Authorization':auth_token,
           'Accept':'application/json'}
params = {'limit':999} # return entire list
body = {'var':
        {'sub_var': 'example'}}
# GET request
requests.get(url=url, headers=headers, params=params)
# POST request
requests.post(url=url, headers=headers, data=json.dumps(body))
```

Code 6 - request schema in ChirpStack

To create an application, a post requests needs to be sent to ChirpStack's API. Table 7 shows a list of the parameters that this request can have.

Parameter	Characteristics	Value	Error message
name	mandatory, string	Can't be repeated	'Invalid application name'
serviceprofileID	mandatory, string	A valid ID	'uuid: incorrect UUID length:'
organizationID	mandatory, integer	A valid ID	'Application and server profile must be under the same organization'
description	optional, string	-	-

Table 7 - Parameters in a post request to create an application

The creation of a device profile was more tedious to analyze, as it includes more parameters. Most of them, as Table 8 shows, are optional. This list does not include the hidden parameters or parameters which default value is blank or zero and are not important for this work.

Parameter	Characteristics	Value
-----------	-----------------	-------

adrAlgorithmID	mandatory, string	Defaults to “default”. Can also be left blank, but this is problematic sometimes.
macVersion	optional, string	LoRaWAN MAC version of the end-devices.
maxEIRP	optional, int	Defaults to “0”.
name	mandatory, string	Can be formed by numbers, letters, hyphens, and spaces. Can be repeated but is not recommended.
networkServerID	mandatory, string	ID of the network Server to which the devices will be added.
OrganizationID	mandatory, string	ID of the organization to which the devices will be added.
regParamsRevision	optional, string	“B” in this thesis.
supportsClassB, supportsClassC	optional, boolean	Default to “false”. Depends on the end-device to be added.
supports32BitFCnt	mandatory, boolean	Defaults to “true”.
spportsJoin	optional, boolean	Default is ‘false’, so it must be included as ‘True’.
uplinkInterval	optional, string	‘<seconds>s’. Default value is ‘0s’, so it’s recommended to include this value.

Table 8 - parameters in a post request to create a device profile

The creation of the end-device is the next step. This request can have the parameters listed in and returns just ‘{}’ if the creation of the node was successful.

Parameter	Characteristics	Value	Error message
applicationID	mandatory, string	An available ID	“object does not exist”
devEUI	mandatory, string	16 hex. digits	“object already exists”
deviceprofileID	mandatory, string	An available ID	"uuid: incorrect UUID length: " if not present and "object does not exist" if wrong.
isDisabled	optional, boolean	defaults to “false”	-

name	mandatory, string	cannot be repeated	"object already exists", another node was registered with the same name.
referenceAltitude	optional, string	Not going to be used, defaults to 0	-
skipFCntCheck	optional, Boolean	Used to disable frame counter validation. Defaults to "false", it's not recommended to change it.	-
description	optional, string	Not going to be used	-

Table 9 - parameters in a post request to create a new end-device

The last step is to activate the end-device.

Parameter	Characteristics	Value	Error message
nwkSKey	mandatory, string	32 hexadecimal digits	"lorawan: exactly 16 bytes are expected" if not present.
devEUI	optional, string	16 hexadecimal digits	-

Table 10 - parameters needed to activate an end-device

During testing with the physical node, it has been noted that an end-device can be activated before or after its activation using NodesApp, which opens the door to performing mass activation tasks in office and mass deployment afterwards. This, however, needs further testing. Also, if a device is deleted from the Application Server, it can't re-join it even if the activation process is started from scratch.

Once the end-device has been activated, there are several possibilities to observe how end-device and Network Server communicate in the ChirpStack LoRaWAN network.

1. The first option is to remotely access the ChirpStack OS installed in the Raspberry Pi 3 via a secure SSH connection. Once inside ChirpStack OS, log messages can be checked using the following command (or opening the equivalent document):

```
sudo tail -f /var/log/messages
```

Code 7 - shell input to see log messages

However, log messages are difficult to read and most of them do not provide useful information for us. A sequence of log messages has been captured to support this claim.



```
Jul 23 10:39:08 raspberrypi3 user.info chirpstack-application-server[364]: time="2021-07-23T10:39:08Z" level=info msg="finished unary call with code OK" ctx_id="2021-07-23T10:39:08Z"
Jul 23 10:39:08 raspberrypi3 user.info chirpstack-network-server[376]: time="2021-07-23T10:39:08Z" level=info msg="finished client unary call" ctx_id=5004663e-4730-4000-8000-000000000000
Jul 23 10:39:08 raspberrypi3 local0.notice redis[455]: 100 changes in 60 seconds.
Jul 23 10:39:08 raspberrypi3 local0.notice redis[455]: Background saving started by pid 601
Jul 23 10:39:08 raspberrypi3 local0.notice redis[601]: DB saved on disk
Jul 23 10:39:08 raspberrypi3 local0.notice redis[601]: RDB: 0 MB of memory used by copy-on-write
Jul 23 10:39:08 raspberrypi3 local0.notice redis[455]: Background saving terminated with success
Jul 23 10:41:07 raspberrypi3 user.info chirpstack-application-server[364]: time="2021-07-23T10:41:07Z" level=info msg="integration/logger: logging event" ctx_id="2021-07-23T10:41:07Z"
Jul 23 10:41:07 raspberrypi3 user.info chirpstack-application-server[364]: time="2021-07-23T10:41:07Z" level=info msg="integration/mqtt: publishing event" ctx_id="2021-07-23T10:41:07Z"
```

```
: time="2021-07-23T10:39:08Z" level=info msg="finished unary call with code OK" ctx_id="2021-07-23T10:39:08Z"
me="2021-07-23T10:39:08Z" level=info msg="finished client unary call" ctx_id=5004663e-4730-4000-8000-000000000000
seconds. Saving...
started by pid 601

y used by copy-on-write
terminated with success

: time="2021-07-23T10:41:07Z" level=info msg="integration/logger: logging event" ctx_id="2021-07-23T10:41:07Z"
: time="2021-07-23T10:41:07Z" level=info msg="integration/mqtt: publishing event" ctx_id="2021-07-23T10:41:07Z"
```

*Code 8 - messages captured*

- The second option can be found when accessing an active end-device through the ChirpStack Application Server. It offers the possibility to visualize uplink and downlink LoRaWAN frames in which the end-device is the emissary or the recipient.

An 'unconfirmed data up' is sent by the device to tell the network server that the device is active. The frequency of this status check can be adjusted in the creation of the device profile.

```
{
  "rxInfo": [
    {
      "gatewayID": "b827ebfffe160aa6",
      "time": null,
      "timeSinceGPSEPOCH": null,
      "rssi": -13,
      "loRaSNR": 8,
      "channel": 2,
      "rfChain": 0,
      "board": 0,
      "antenna": 0,
      "location": {
        "latitude": 0,
        "longitude": 0,
        "altitude": 50,
        "source": "UNKNOWN",
        "accuracy": 0
      },
      "fineTimestampType": "NONE",
      "context": "kpDsUw==",
      "uplinkID": "34b11aeb-234f-4b17-b2ce-57d1f60471b7",
      "crcStatus": "CRC_OK"
    }
  ]
}
```

```

"txInfo": {
  "frequency": 868500000,
  "modulation": "LORA",
  "loRaModulationInfo": {
    "bandwidth": 125,
    "spreadingFactor": 7,
    "codeRate": "4/5",
    "polarizationInversion": false
  }
},
"phyPayload": {
  "mhdr": {
    "mType": "UnconfirmedDataUp",
    "major": "LoRaWANR1"
  },
  "macPayload": {
    "fhdr": {
      "devAddr": "002090bc",
      "fCtrl": {
        "adr": false,
        "adrAckReq": false,
        "ack": false,
        "fPending": false,
        "classB": false
      }
    },
    "fCnt": 196,
    "fOpts": null
  },
  "fPort": 2,
  "frmPayload": [
    {
      "bytes": "j8KHz+0gXA=="
    }
  ],
  "mic": "5f4db2eb"
}

```

Code 9 - LoRaWAN frame captured

- Alternatively, the payload of the LoRaWAN messages can be accessed connecting to the MQTT broker installed in the Raspberry Pi 3. This method requires MQTT knowledge to write a simple script to build a MQTT client. Python can also be used for this purpose, and it offers the Paho MQTT library [12] for faster and easier development.

Given that the API will also be written in Python, it's always a good idea to start the project by creating a virtual environment, which can be described as an isolated python installation. This isolation allows the user to localize the installation of a project's dependencies, without forcing a system-wide installation and avoiding conflict in package versions with other projects. It also facilitates the creation of a 'requirements.txt' file, which has been created for this project and includes a list of the python libraries needed to run the project, along with their version number used during testing.

This script creates a MQTT client that, given the IP address and device port of a MQTT broker to connect to, subscribes to every topic and prints the received messages in an easy-to-

understand format. It can be found as an annex under the name “MQTT-client.py” with comments to help its understanding.

After reading the captured messages we learn that the topic in which status messages are published is "gateway/b827ebfffe160aa6/event/stats". An example of a message is the following JSON (with our gateway bridge configuration):

```
{
  "gatewayID": "uCfr//4WCqY=",
  "ip": "",
  "time": "2021-06-18T17:31:38.566061453Z",
  "location": null,
  "configVersion": "",
  "rxPacketsReceived": 0,
  "rxPacketsReceivedOK": 0,
  "txPacketsReceived": 0,
  "txPacketsEmitted": 0,
  "metaData": {
    "concentrator_d_version": "3.2.0",
    "config_version": "97abd112-b296-4009-a4a7-51256ac017a9-r1623770647",
    "hal_version": "Version: 5.0.1;", "model": "generic_eu868"
  },
  "statsID": "Yxb59s2GQli+Rc7i6qT/xQ=="
}
```

Code 10 - MQTT message captured with the MQTT client

### 3.3 Web application design

The functionality aimed by this work can be implemented in several ways. It could be developed as a desktop application, a native mobile application or as a web tool. Considering mainly the broader userbase accessed and the optimization of resources (time and knowledge), the web application is the best option:

- A web application can be accessed using a computer and a mobile phone, allowing the mobility that this thesis is aimed to achieve while supporting desktop usage of the tool. Users can access a web application using a web browser such as Google Chrome, Mozilla Firefox, or Safari.

- A web application can be accessed regardless of the operating system of the mobile phone or desktop app, making the tool as accessible as possible.
- Recently developed technologies like Google's desktop accessible web apps can be used to make a web application as read-to-use as a native app.
- Web applications typically have short development cycles and can be made with small development teams.

A web app needs a web server, application server and a database to work. Web servers manage the incoming requests from the users' clients, the application server completes the requested tasks, and the database can be used to store any needed information. Web apps can be divided in two parts, client-side and server-side. Client-side programming builds the front-end of the application and usually utilizes languages like JavaScript, HTML5, and Cascading Style Sheets (CSS). Server-side programming is done to create the scripts a web app will use. That is, the API that will give the end-device connecting functionality. Languages such as Python, Java, and Ruby are commonly used in server-side programming.

The author has experience in the usage of the front-end technologies HTML5 and CSS and the back-end programming language Python. The high-level Python web framework Django is a great open-source option to achieve a rapid development of a secure and maintainable website. The main advantage is that the Django framework can take care of most of the hassle of web development, such as security, scalability, and portability, which are main ingredients of the application to be developed, from now on called **NodesApp**.

In a traditional website, a web application waits for HTTP requests from the web browser (or another client). When a request is received the application works out what is needed based on the URL and possibly information in POST and GET data. The application will then perform the tasks required to satisfy the request and return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

The programming language used for the back end is Python 3.8.3, using version 3.2.4 of the high-level framework Django specialized in web development. The requirements can be found as an annex in a text file named *Requirements.txt*.

The code in NodesApp is grouped following this logical sequence and Django application's typical architecture (Figure 7).

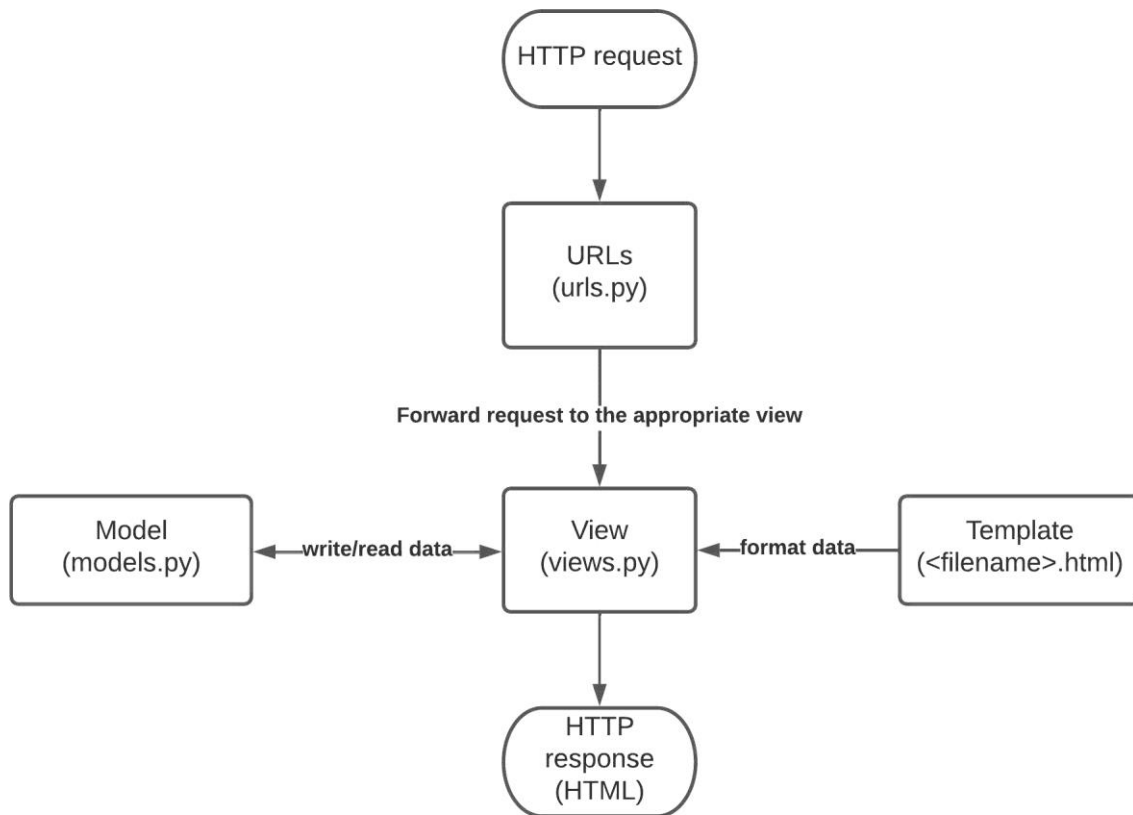


Figure 7 - Organization of code in NodesAPP

### Urls.py

This first task is to send the request to the right view. This task is performed by the 'urls.py' file, which is used to redirect HTTP requests to the appropriate view based on the request URL. This URL mapper can also match patterns of strings or digits that appear in a URL and pass these to a view function as data, making use of the query string discussed in Chapter 2.

NodesApp has been developed in a modular way that allows the implementation of other features under the same base URL. Thus, the main urls.py file redirects to another urls.py file only used for the developed app, allowing the addition of a new part of the application that can be independent while integrated in the same environment.

This file also includes the URLs to media files used in the application.

```
urlpatterns = [
    # path('admin/', admin.site.urls),
    # path to 'connect app' is blank because it's the only one by now.
    path('', include('connect.urls')),
]
# url to media files
urlpatterns += static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)
```

*Code 11 - urls.py of the project, which redirects to the urls.py of the application*

### Views.py

Views are the core of the web application. They receive HTTP requests from web clients and return HTTP responses using resources like the database and the template render.

Views receive an HTTP Request object as a parameter (request) and return an HTTP Response object. In the NodApp, views are kept tidy making use of a helper file, *utils.py*. Functions are scripted in *utils.py* and called in *views.py* to not crowd this file.

### Models.py

Django web applications manage and query data through Python objects referred to as models. Models define structure of stored data, such as its type, default values, etc.

As explained in chapter 2, the developed application is going to be used to host a REST API. Thus, it doesn't need a database. However, an SQLite database has been included to allow the implementation of more features in the future. NodApp makes use of session data to store the needed user information without accessing the database.

### HTML templates

Templates allow the programmer to specify the structure of an output document, using placeholders for data that will be filled in when a page is generated. Templates are used in NodApp to create HTML but could also create other types of documents.

The first thing in the development cycle of any application or API is to know the input and output parameters that it's going to have. The input parameters need to provide enough information about the end-device and the network server and are presented in Table 11.

Context: Name	Required/Optional	Input type	Values
---------------	-------------------	------------	--------

End-device: DevEUI	Required	String	16 hexadecimal characters (usually lowercase, uppercase accepted)
End-device: AppKey	Required	String	32 hexadecimal characters (usually uppercase, lowercase accepted)
End-device: mac version	Optional	String	LoRaWAN version format (e.g. "1.0.2"). Defaults to blank if not present.
End device: regional Parameters	optional	String	LoRaWAN regional parameters revision (e.g. "B"). Defaults to blank if not present
Network Server: IP Address and port	Required	string	IP address, port.
Application Server: Authentication user	Required	String	ASCII, as per ChirpStack
Application Server: Authentication password	Required	string	ASCII, as per ChirpStack

Table 11 - Input needed by the API

As output parameters, the API only needs to reveal if the end-device has been correctly registered and activated or not. In case it's not, it should provide information to help the user fixing it. Therefore, output parameters are the following:

Context: Name	Output type	Description
End-device: DevEUI	String	Information about the end-device.
End-device: AppKey	String	Information about the end-device
Application: d_app	string	Name of the application to which the end-device has been assigned.
Application: id_app	integer	ID of the application to which the end-device has been assigned.
Device profile: d_devprof	String	Name of de device profile used to register the device

Status of creation: <i>info_creation</i>	String	It can be 'successful' or 'unsuccessful'
Status of activation: <i>info_activation</i>	string	It can be 'successful' or 'unsuccessful'

Table 12 - output parameters of NodesApp

The file structure clearly puts *views.py* in the center of the application's operations. It contains the entire process, from the receipt of the HTTP request to the emission of the HTTP response. Therefore, it is here where the architecture of the application must be designed. Five main events can be identified in the process of activating an end-device, where the gathering of information has been split into two events, since the source of that information is completely different. This results in three phases, which will be represented by three *views*:

**Phase 1: Receiving information about the network and the user.** The NodesApp must be available for any user with an active ChirpStack Network. Therefore, the IP address of its application server will always be a variable that can be populated by the user of NodesApp. Furthermore, ChirpStack's Application Server has a user authentication system, only allowing users created in the Application Server to edit network information, such as creating a new device. Every change made in the network must come from a user. Therefore, an authentication token must be included in every call to the Application Server's API, since it needs it to authenticate the respective call to the Network Server. A *global admin user* account is needed to request the creation and activation of a new device. Alternatively, a global admin user can give access to a *regular user*, who by default has zero permissions, to the creation and activation of end-devices.

**Phase 2: Receiving information about the end-device.** As discussed before, this part needs to be as dynamic as possible to accomplish the goal of creating a mobility tool. A QR code scanner needs to be implemented. It must work using a mobile phone and a desktop computer. The QR code reading is started by the users when they are ready to scan an end-device and this phase ends when a valid QR code has been detected and successfully read. This part of the application needs to include the option of logging out to allow users to change the Application Server that they are connecting to and the authentication information that they are connecting with.



**Phase 3: Creating and activating the end-device in the ChirpStack Network Server and showing the results of the operation.** After gathering the required information, the main task must be carried out. This third phase covers everything needed to validate the information read from the QR code, identify the status and the components of the network, and create and activate the end-device.

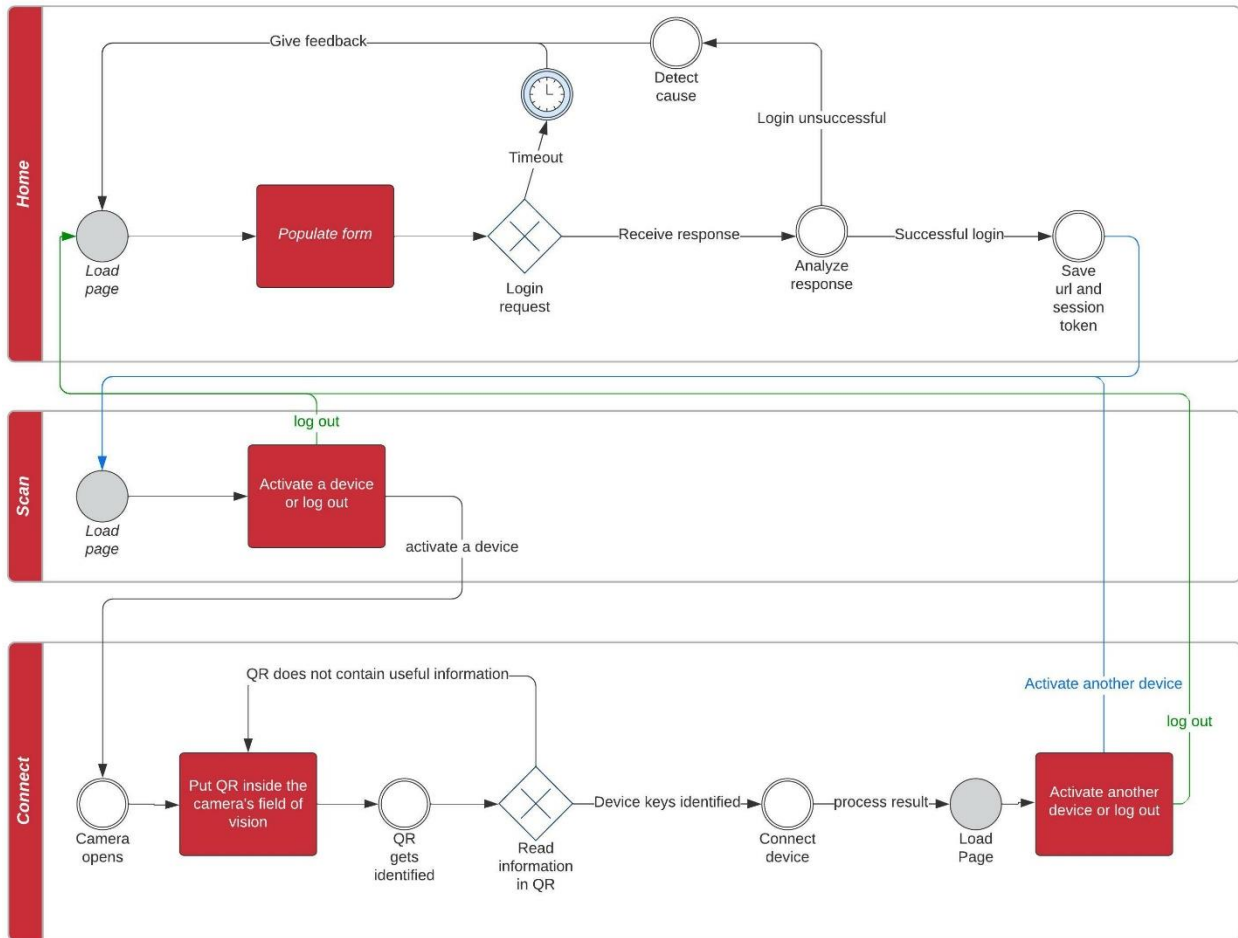


Figure 8 - Schematic of the NodesApp features

Furthermore, extra features are needed to handle errors, timeouts, or user decisions, such as logging out or leaving the application without completing a task and then returning. A flowchart is presented in Figure 9 to illustrate representing the basic process of the app. The tasks that the end user must perform are highlighted in red rectangles. They have been kept to the minimum, as was stated by the objectives of this web tool.

### 3.3.1 The *home* view

The first *view* in our Django app is called *home*, as it is the first one seen by the user. Here, users can authenticate themselves in the network's application server with their network accounts, which must have been created beforehand. For this purpose, a form must be submitted, where the user can also input the URL to the network's Application Server. Then, the path segment leading to the API is always the same and therefore known. Using the login information to perform a login call to the given URL the application gets the authentication token needed for the API calls, called *Grpc-Metadata-Authorization*.

In this case our application is going to handle sensitive information. Therefore, the submit button should do a POST request. GET requests consist of just a string of characters formatted as a URL. If used, that data would appear as plain text in the URL, browser history and server logs. A web application that uses GET requests for admin forms is a security risk: it can be easy for an attacker to mimic a form's request to gain access to sensitive parts of the system, like the Application Server. POST offers more control over access and more security, so we are going to use this request method.

Django offers the possibility of handling a form's data with their form classes. However, when using this technology, the appearance of the website is much more difficult to customize. It would require third-party user-maintained python packages, which would add inconvenient dependencies to our app. Therefore, the form will be manually programmed using HTML5 and bootstrap [13] to make the user interface attractive to the user's eyes. Once the form has been populated, the user can submit it pressing an accordingly labelled button. This will generate a post request to the same 'home' view.

Now the 'home' view will behave differently upon detecting that it has been called by a POST request. The first thing is to try to log in the user to verify the data received and obtain the session token necessary to make changes in the user's ChirpStack network.

- A. If a *RequestException* occurs while performing the request, users are redirected to the standard 'home' view again, since we can't handle the problem. It's probably a connection problem or the URL provided didn't respond, so the request timed out. Any unidentifiable error lands here.
  - B. The URL provided is valid:
    - If the status code received in the response is '200', then the URL does point to a ChirpStack Application Server and the login was successful. Username, URL, and session token (after proper formatting to match ChirpStack's bearer token) are saved as session variables in Django
- 4.1 User interface. The user is redirected to the 'scan' view to continue the process, which will be explained in 3.3.2 The *scan* view.

- If the status code received in the response is '401', the URL points to a ChirpStack Application Server but the login was not authorized. This means that either the username or password are not valid. The form will be loaded again with an extra message explaining the issue.
- If the status code received in the response is '500' or '404', the application server was not addressed correctly. This probably means that an internal server error occurred (500) or that the server was not found at all (404). The reason for this is probably that the URL or the port number have not been correctly entered, which will be explained when the form is loaded again.
- If any other request status code is returned, the form will be loaded again showing that an unexpected error occurred.

The author assumes that the end user of NodesApp is trying to connect an end-device to a working ChirpStack network. This means, that the Application Server is connected to, at least, one Network Server and that a Service Profile was created. An organization (ChirpStack resource) is also needed. Therefore, this is not checked at this point. The registration and activation of the end-device will fail in the last phase of the application if this assumption is not met and NodesApp will show an error message.

### Cross Site Forgery Protection

The 'csrfmiddlewaretoken' is used as Cross Site Request Forgery protection. This type of attack occurs when a malicious website contains a link, a form button or some code intended to perform some action on our website, using the credentials of a logged-in user who visits the malicious site in their browser. A related type of attack, 'login CSRF', where an attacking site tricks a user's browser into logging into a site with someone else's credentials, is also covered. The CSRF protection is based on the following things:

- A CSRF cookie that is based on a random secret value, which other sites will not have access to. To protect against breach attacks, the token is not simply the secret; a random mask is prepended to the secret and used to scramble it. For security reasons, the value of the secret is changed each time a user logs in.

A hidden form field with the name 'csrfmiddlewaretoken' present in all outgoing POST forms. The value of this field is, again, the value of the secret, with a mask which is both added to it and used to scramble it. The mask is regenerated on every call so that the form field value is changed in every such response. This part is done by the template tag `{% csrf_token %}`.

For all incoming requests that are not using HTTP GET, HEAD, OPTIONS or TRACE, a CSRF cookie must be present, and the 'csrfmiddlewaretoken' field must be present and correct. If it isn't, the user will get a 403 error.

When validating the 'csrfmiddlewaretoken' field value, only the secret, not the full token, is compared with the secret in the cookie value. This allows the use of ever-changing tokens. While each request may use its own token, the secret remains common to all.

In addition, for HTTPS requests, strict referrer checking is done. This means that even if a subdomain can set or modify cookies on your domain, it can't force a user to post to your application since that request won't come from your own exact domain.

### 3.3.2 The *scan* view

With the login details provided, our app can get the required authorization token and is now able to perform actions on the user's ChirpStack network. In the 'scan' view, the possibility to scan a QR code from an end device is presented. A log-out option is also available for the users, in case they want to change the Network Server, or the authentication data entered in the previous step.

This view can be seen as a transition between the *home* view and the *connect* view. It also serves as a point of return for users after they have activated a new node.

Another important feature of this view is the introduction of session data management. After authenticating themselves, users get a token which is valid in their ChirpStack network. This token is saved in Django's session data along with the username and the Network Server's IP Address. NodesApp checks after every request if the user is authenticated. That is, if the session data is still valid. ChirpStack's tokens are fungible, and so are the ones in NodesApp. Session data expires before the bearer token would expire in the ChirpStack network to ensure that no connection is interrupted by token lapsing. This adds up to the rest of NodesApp security features.

### 3.3.3 The connect view

This view is the most important part of the application, since it contains the API that has been developed to connect the end devices to the network. Therefore, its architecture is represented in the following flow chart.

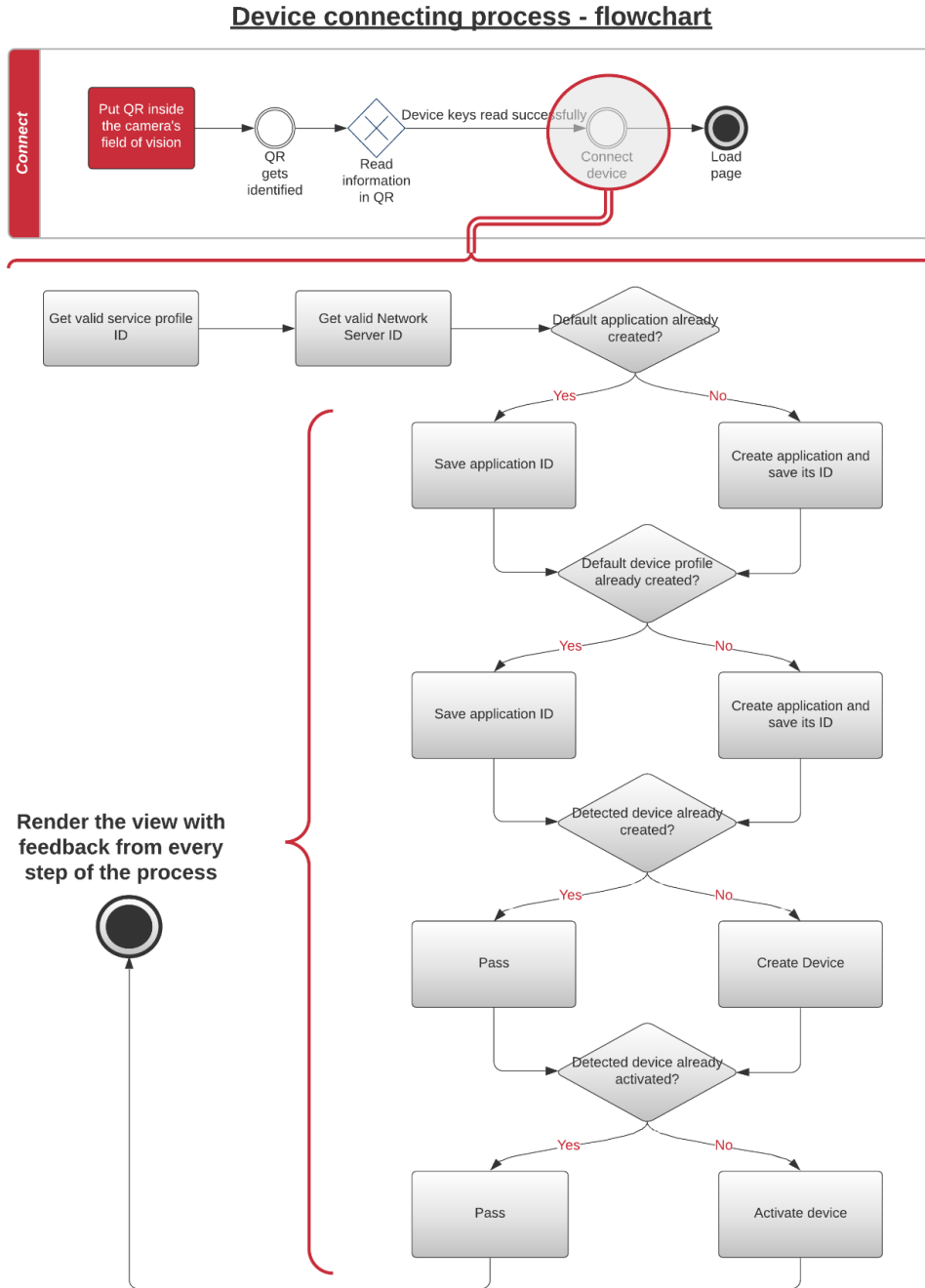


Figure 10 - Flow Chart of the connect view

First, the camera activates and checks every frame to detect any QR or bar code in them. If the user puts the QR code inside the camera's field of vision it gets detected and its content is analyzed. If its data matches the expected format, NodesApp saves the content and closes the camera. After this, the activation process begins. After everything has been completed, the view loads the HTML showing the results of the operation, which are basically divided in three:

- NodesApp has been able to activate the end-device successfully.
- An error has occurred. It was not possible to add the end-device.
- The end-device was already created or even activated.

How NodesApp performs the necessary tasks is thoroughly explained in the following two sections.

### QR code scanner

The following QR code has been generated. It contains a string (plain text) consisting of the DevEUI in first place and the AppKey in second, separated by just a comma. The official QR design recommendation stated by the LoRa Alliance in its Technical Recommendation 005 [1] is different, but for testing purposes, the previously mentioned format has been chosen. The dictionary option has been discarded because variable names could be different between end device developers. This would cause problems when recognizing each variable. This way, the order is what determines their meaning. Here is an example of a QR, which if scanned, returns the following string:



*Figure 11 - QR code. This one contains the device keys '70b3d549912693f3,11B0282A189B75B0B4D2D8C7FA38548B'*

Only the device EUI and application key are going to be stored inside the QR code. Optionally, LoRaWAN version and region parameter can also be included. These will default to '1.0.2' and 'B' if not present.

NodesApp program opens the camera and keep analyzing each frame until it detects a QR code. Once detected, it should decode its contents and save it in a string. After that, the camera should be released.

Python is often used to aid in computer vision programs. Therefore, very good libraries can be easily found in this field.

First, a simpler script only opening the camera, recognizing the barcode in a frame and drawing a rectangle on it. The edited frame will be saved to be checked. Here, two external libraries are used:

'cv2' imports the OpenCV library. The Open-Source Computer Vision Library is an open-source computer vision and machine learning software library, as its name implies. It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.

'pyzbar' imports the pyzbar library, a pure Python library that reads one-dimensional barcodes and QR codes using the zbar library, an open-source software suite for reading bar codes from various sources, such as video streams, image files and raw intensity sensors. zbar supports many popular types of bar codes, including EAN-13/UPC-A, UPC-E, EAN-8, Code 128, Code 39, Interleaved 2 of 5 and QR Code. Therefore, an end-device manufacturer has more flexibility and can generate his preferred barcode symbology.

```
import cv2
from pyzbar import pyzbar

def decode_saved_qr(img): # funciona con mi qr
    image = cv2.imread(img)
    detectedBarcodes = pyzbar.decode(image)
    for barcode in detectedBarcodes:
        (x, y, w, h) = barcode.rect
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 5)
    cv2.imwrite("image.png", image)
```

*Code 12 - script to detect a QR and draw a rectangle around it*

Here is a brief explanation of the code:

1. cv2.imread(image) decodes a saved image to analyze its contents.
2. cv2.rectangle(img, start\_point, end\_point, color, thickness) draws a rectangle given an image to draw on and the desired rectangle's coordinates, color and thickness.
3. pyzbar.decode(image) detects the barcodes in the image. It returns a list of barcode objects if it has detected any.

- a. Barcode class attributes:
  - i. rect: contains the bounding box of the rectangle to which the barcode corresponds, as the top-left coordinates (x, y), the height (h) and width (w).
  - ii. type: Type of barcode detected, as explained before.
  - iii. data: decoded barcode content, as a string.

Passing the saved QR code as argument (img), the function returns a saved "image.png", which is shown below:



*Figure 12 - Proof of successful QR code detection*

Upon encountering this image correctly saved, we know that the QR code gets correctly identified. The next step is to change the saved image for a captured video frame. This is done by the following code:



```
def open_camera_read_qr():
    capture = cv2.VideoCapture(0)
    if not capture.isOpened():
        print('Can\'t open camera')
        exit()
    while True:
        # capture frame
        ret, frame = capture.read() # if frame is read correctly, ret is true.
        if not ret:
            print('Cannot receive frame (stream end?). Exiting...')
            break
        # Operations on the frame
        barcode_info = detect_read_barcode(frame)
        # stop loop if information matches the expected syntax
        if barcode_info:
            pattern = r"^\[('[0-9a-fA-F]{16}'),'([0-9a-fA-F]{32})'\]$"
            match = re.match(pattern, barcode_info)
            if match:
                break
        print(barcode_info)
    capture.release()
```

*Code 13 - QR code detecting feature*

Here is a brief explanation of the code:

- `capture = cv2.VideoCapture()` captures video. Its argument can be either the device index (0 for default camera) or a video file. It captures frame-by-frame (returned capture object) and at the end, the capture should be released.
  - a. The `isOpened()` attribute returns a boolean that can be used to check if the camera has been correctly identified and accessed. An error handling if state has been introduced using this attribute.
  - b. The `read()` object is the main `VideoCapture` use case. It returns a boolean (`ret`) to know if the frame has been correctly decoded and the decoded frame itself. Another error handling if state has been coded to exit the loop if the frames are not being capture correctly, since it would be pointless to continue.
- The `detect_read_barcode(frame)` function will be coded next, after checking if this python definition is working correctly. It should return the barcode's content, which will be . Of course, the possibility

that no barcode has been detected is handled, putting the operations on the barcode info behind an if state which checks if there is any barcode info to analyze. If there is, it will be matched against the expected syntax. This way, we prevent that non-matching QR codes disturb our application. If the syntax matches, the loop will stop and the barcode info saved (by now, printed to check if it works).

The last step is to write the code that will decode the information inside the QR code. As before, the pyzbar library has been used to accomplish our goals. This definition can also return the type of the barcode detected, which could be interesting feedback to the user. For now, we leave it unused.

```
def detect_read_barcode(frame):
    detected_barcodes = pyzbar.decode(frame)
    for barcode in detected_barcodes:
        # decode the information in the detected barcode
        barcode_info = barcode.data.decode('utf-8') # decodes data
        barcode_type = barcode.type
        # return decoded information
    return str(barcode_info)
```

*Code 14 - helper function, extract the information found in the QR code*

**Handling errors in the QR reading process:** This process is highly dependent on the users' hardware and its accessibility, as well as the users' actions. Therefore, a lot of problems can occur while our application is trying to read the keys of a new end device.

1. The camera may not be accessible at all.
2. The captured frames have decoded. If that process fails, the rest of the code won't work.
3. Users may not be able to put the QR code inside the camera's view range or may decide to do something else instead. A timeout is implemented to release the video capture if the code detection takes more than a minute.

Every one of the errors listed above has its error code. If anything happens, users are redirected to the 'scan' view again, where appropriate feedback will be given.

## Activation process

As discovered during the reverse engineering phase of this project [3.2 Understanding the task - Reverse engineering], the process of activating a new end-device in a ChirpStack network has multiple layers and requires some considerations. At this point, NodesApp has already proved that the user's authentication details are correct and, thus, it's able to access the network. A valid service profile and network server ID are also a requirement, since these cannot be created via an HTTP request to the API.

This section starts with the following information:

- About the node: DevEUI, AppKey, MAC version, and regional parameters revision.
- About the network and the user: valid bearer token, Application Server's IP address.

The first thing that NodesApp needs is valid IDs from a service profile and a network server. The application server can give us information about this. NodesApp will take the ID of the first item in both returned lists and use them for the following steps.

The default application that NodesApp uses to register end-devices is called *RHAapplication*. If this application is already present, the ID of the already available application is retrieved for later use. If NodesApp doesn't see an app called *RHAapplication*, it creates one with such name, saving its ID which is a field returned by the API after successfully creating the application.

A similar process is followed with the device profiles. In this case, there isn't a default device profile name. As stated during the testing phase, multiple service profiles can share the same name. However, this would be chaotic for the users. Also, devices will have different hardware components and characteristics. Therefore, a new device profile is created each time an end-device joins the network. The name is composed by *RHA DevProf* and the last four digits of the *Device EUI*. If the device had been previously registered, its device profile will be used. The ID of the device profile gets saved by NodesApp.

With the application ID and the device profile ID the end-device can be registered. The application directly sends the request for this. The possible answers are shown in 3.2 Understanding the task - Reverse engineering. NodesApp analyses receives the confirmation or an error code and saves the respective result. A comprehensive explanation is also created to give feedback to the user.

In the activation process the app follows a similar sequence. In this case, the URL to make the request to needs to be populated by the devEUI, which entails a relative security risk. However, this is how ChirpStack handles device activation and can't be avoided. NodesApp saves the result of this operation to give feedback to the user. While trying to activate the device, a successful activation or an error stating that the device is already active in the network are common. A third option covers the case of a non-

registered device. NodesApp users will never encounter this issue since non-registered devices that are scanned change their status in the previous step. However, if an error has occurred while trying to register the device, this third option gives double feedback to the user.

At the end of this process, the information gathered in each step is presented to the user rendering an HTML template.

# Chapter 4

## Results

The result of this work is the working prototype called NodesApp. How can the creation of a web application be presented and assessed? This app aims to improve time efficiency in LoRaWAN deployments in a mobility environment, so the time saved by using NodesApp needs to be analyzed. Furthermore, the result of this application are its user interface and its user experience. Last but not least, a final testing phase has been performed to ensure that the features that this prototype includes are fully functional. These are the three main topics discussed in this section.

### 4.1 User interface

The application has been designed to fit both smartphone and PC screens. The HTML components used are flexible and adapt to every screen. The three pages are static, which means that they do not contain JavaScript methods to allow interaction with the user on the same page, apart from basic aesthetic components.

There is one page for each view. The web page design stays consistent across them, which is a very important design feature in every kind of application. Consistency can be looked at from two different points of view.

On the one hand, the application is themed to make the design coherent on every page and between pages. The theme has been selected according to ITI's main web page and colors. Thus, orange dominates in every page. Furthermore, Bootstrap technology and its CSS classes allow us to build a modern front-end to enhance the user's experience. Font sizes, page headers and buttons are styled the same. A focus on simplicity and functionality is followed throughout the entire web page. NodesApp is an industrial application, and it should look and feel that way. It has only one purpose, to connect an end device to a

LoRaWAN network. No extra features are needed; thus, they are not included. This optimizes development time and cost.

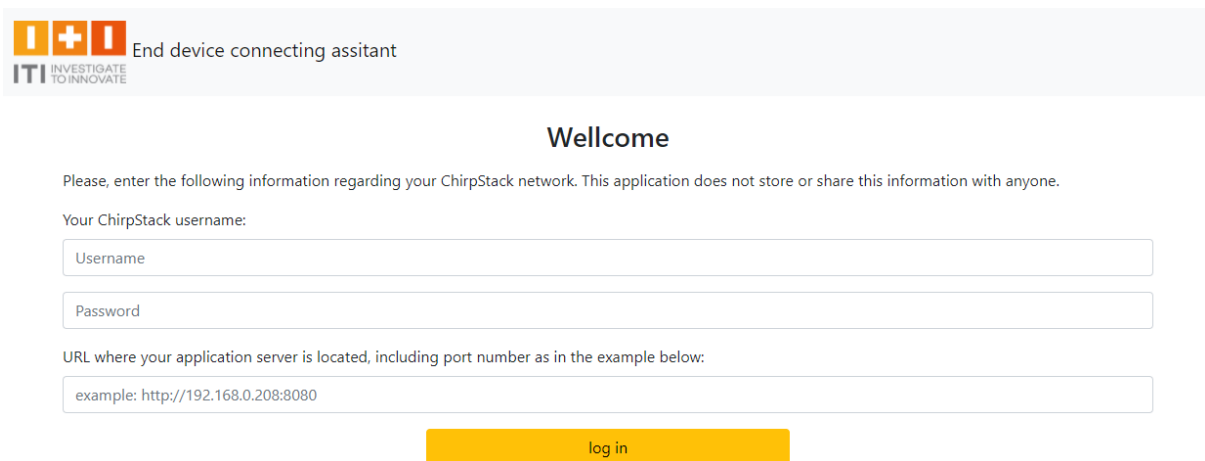
On the other hand, the application can be accessed using a computer as well as a mobile phone and responds the same way on both. The design stays consistent even when changing the platform. NodesApp can be used for mass activation using a computer or single activations in a mobility environment, which gives this app the desired flexibility of use.

### The home view

This is the first page that the user sees when connecting to NodesApp. Because this application uses Django's session data management and not cookies, no warning is required, which makes this first contact smoother. Figure 13 is a screenshot of the PC version of this page.

Starting from the top, the header consists of the ITI logo and the purpose of the application, which is assisting in the connection of end-devices. Then the app welcomes the users and explains the first step that they need to take, which is to populate the form below. This form has placeholders to help users identifying the correct information to input. It is also divided into two segments; authentication details and the IP address of the network's application server to be used.

The 'log in' button is placed on the bottom. After clicking it, NodesApp sends a request to the introduced IP address. The possible results have been analyzed in 3.3.1 The *home view*. The user interface shows the result of that analysis in the HTML template. If the authentication fails, a red message appears over the field that has been deemed as the cause of the error. If the source of the error is unknown, this is also stated when the *home view* is reloaded.



ITI INVESTIGATE TO INNOVATE

End device connecting assistant

## Wellcome

Please, enter the following information regarding your ChirpStack network. This application does not store or share this information with anyone.

Your ChirpStack username:

Username

Password

URL where your application server is located, including port number as in the example below:

example: http://192.168.0.208:8080

log in

Figure 13 - Screenshot of the PC version of the home view user interface

### The scan view

The HTML template of the *scan* view inherits the header of the page. This time, it also includes a *log out* button to return to the *home* view, clearing all session data. This page is the one which is loaded when a user opens NodesApp, and session data is still valid. Therefore, the log out button includes the username to remind the user of the authentication details used when initiating the session. For the same reason, information about the Application Server is also included at the bottom of the page, specifically at the bottom of a Bootstrap *card*. This item has been included to highlight the QR scanning section as a special module. At the top it includes the title 'QR code scanner', then a brief indication to the users, and a QR code to exemplify what needs to be scanned. Under the QR code, users can find the 'Begin scanning' button that allows them to proceed with the activation. Lastly, the aforementioned information about the Application Server is presented below the button.

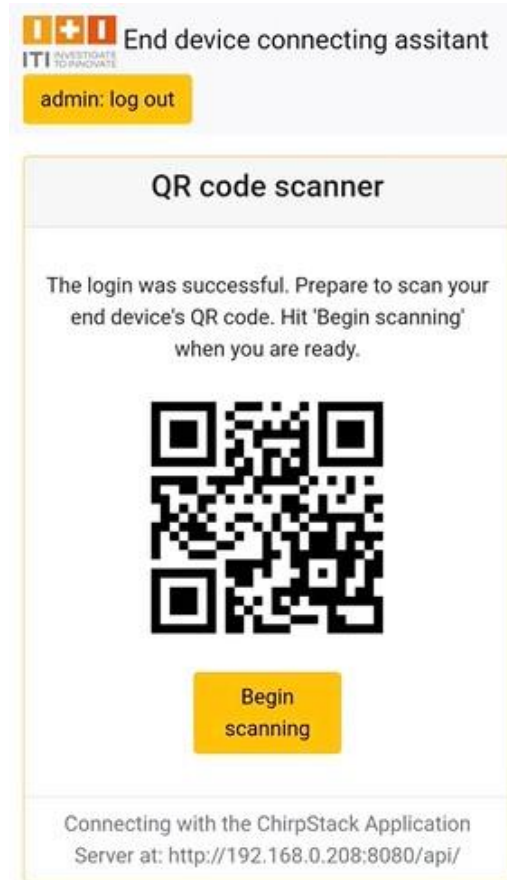


Figure 14 - Screenshot of the mobile version. Scan view HTML template without error messages.

### The connect view

This template also inherits the same header, keeping the design consistent in conjunction with the three phases of an individual process. It also includes the log out button introduced in the *scan view* to allow the user to directly log out, without returning to the *scan view*.

The title of this page reveals the result of the operation since it's the most important thing that the user wants to see. The activation can either be successful or unsuccessful. The body of the page reveals information about the device and its creation and activation.

- The keys read from the QR are shown to provide transparency and usefulness to NodesApp. The user may want to personally check these keys, or copy-paste them in another place.
- The name and ID of the application and device profile assigned to the new end-device are presented in this page. This is useful information for the user.

Finally, a button gives users the ability to return to the *scan* page and activate another end-device.



**End device connecting assistant**  
admin: log out

### End device activation successful

The scanned device has the following information available in the QR code:

Detail	Value
Device EUI of the end device	70b3d549912693f3
Application key of the end device	11B0282A189B7580B4D2D8C7FA3854

Here are the details on the operation:

Detail	Name	Value
Application assigned to the device	RHAApplication	22
Device profile used to register the device	RHA DevProf 93f3	25c4c305-d5cd-4f06-9
Device creation	Has been successful	70b3d549912693f3
Device activation	Has been successful	11B0282A189B7580B4

Scan another device

Figure 15 - Screenshots of the mobile version. Connect view HTML template successful activation



## 4.2 Time saving with NodesApp

The Django Framework used is often used for developing much bigger applications. Our website's calls per minute and required computing power can be handled easily by Django and basic server hardware.

Often, the bottleneck will be how fast the user's deployed ChirpStack Application Server handles the requests that our application sends to it. As NodesApp has no control over that, the speed of activation will not be measured until the physical connection is established, but until the ChirpStack Application Server has the necessary keys to achieve it.

Features of the application that consume time:

- Authentication.
- Entering the IP address.
- Rigid structure. Special cases would require some previous personalization of the app.

The first two tasks could be done only once a day, for the first activation, leaving the rest of the activations just one click away. The third could be problematic. It is going to be included in this analysis, but it's a direct consequence of NodesApp being a prototype of a future application.

The current state of the app allows the elimination of following tasks:

- Creation of an application, including its configuration
- Creation of a device profile, including its configuration
- Creation of a device, including the selection of an application, a device profile, and a few extra parameters.
- Activation of a device, including the input of device keys.

The first two points could be done just one time, leaving only the creation and activation as repetitive and unavoidable tasks.

The analysis of time consuming while deploying a new end-device is not equal among all users. More experienced and prepared users will be able to effortlessly navigate through the Application Server, while unexperienced users will find NodesApp much easier to understand. This means that the time difference between the direct use of the Application Server and NodesApp will be much higher in the first end-device deployment that a user performs. However, the time comparison does not make sense in that situation, as the main advantage of NodesApp at that point is the simplification of the task in terms of technology understanding, not in speed.

Therefore, the following analysis includes only participants who have done at least 5 end-device activations before entering this study, where they do 5 more. However, both apps were returned to the ‘initial’ state after the 5 activations prior to the experiment. That means a new application and a new device profile need to be created.

The graphical analysis presented in Figure 16 shows that participants achieved the first activation 49,8 seconds faster using NodesApp than using ChirpStack’s Application Server. This means, that users would invest around 46% less time in the first deployment with the developed tool. This is explained by the lack of configuration in NodesApp and the ability to scan the keys instead of typing them.

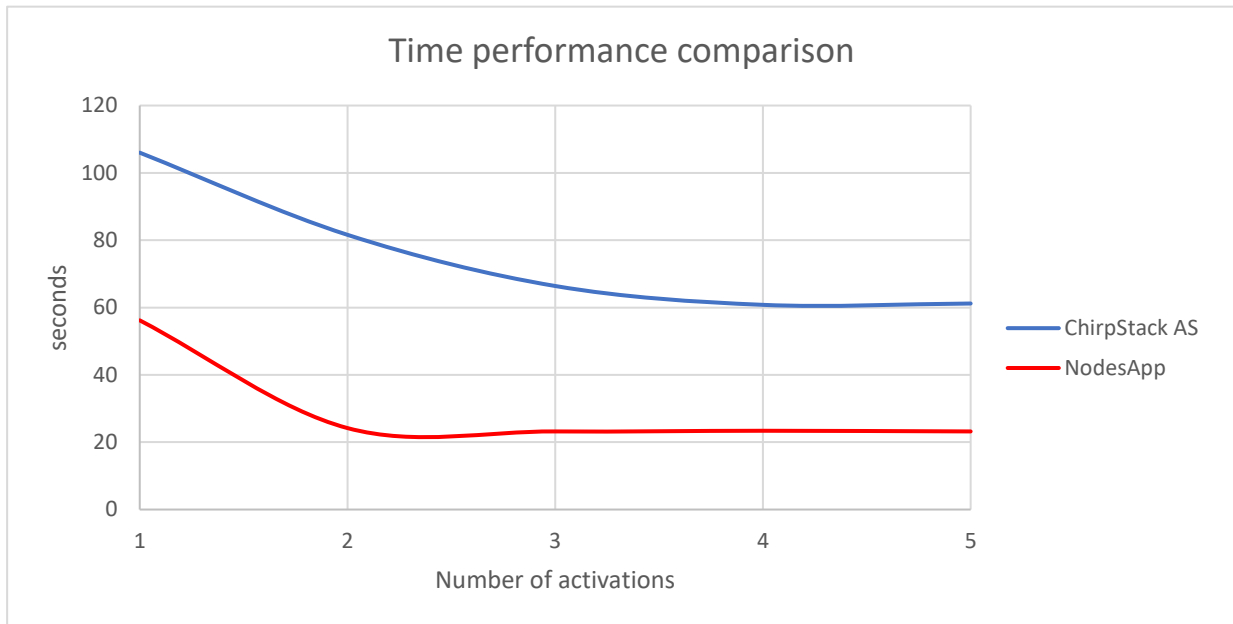


Figure 16 - time performance comparison in 5 consecutive end-device activations between ChirpStack Application Server and NodesApp, 5 subjects of different technical expertise.

The second deployment is where the difference in performance between both methods is at its greatest. Users in ChirpStack still need to type the keys, while NodesApp lets them hit two buttons and scan another device right away. Participant three, as Figure 17 shows, is above the mean in every one of the five deployments. This participant experiments a much greater decrease in time performance between the first and the second deployment when using NodesApp than with ChirpStack’s Application Server. This shows that NodesApp also accomplishes its second goal, to lower the technological barrier in the deployment of a LoRaWAN network.

Lastly, the time spent per deployment in NodesApp stabilizes after this second deployment, while ChirpStack’s Application Server decreases until the fourth deployment and then rises again. This increase could be caused by non-representative data which could be still significant because of the few participants.

Another reason could be fatigue after copying several key pairs for the previous four devices. In this last case, NodesApp could prove even more valuable.

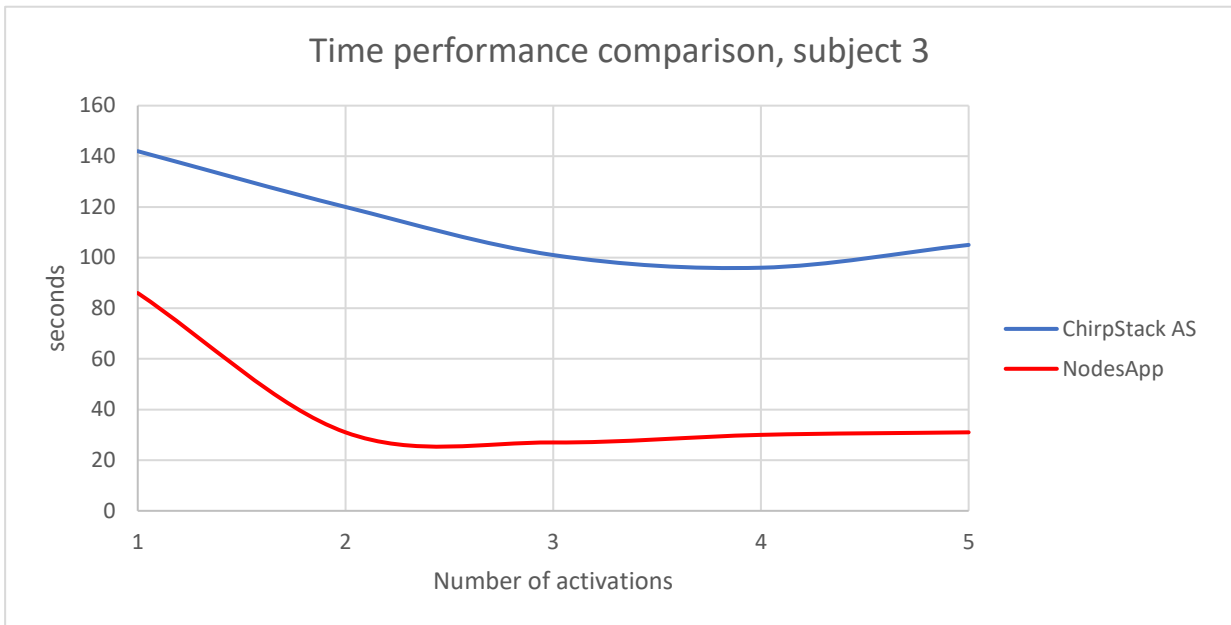


Figure 17 - Time performance comparison in 5 consecutive end-device activations between ChirpStack Application Server and NodesApp, subject with low technical expertise.

### 4.3 Testing NodesApp

NodesApp will encounter a lot of different scenarios while trying to activate an end-device. The state of the network can vary, and the creation of an application and service profile could be troublesome. Error handling techniques keep the user informed about situations that the application can't handle, but the implemented features should work in any situation.

The activation of an end-device has been tested several times in every one of the following scenarios, always being able to successfully activate a valid end-device:

Regarding the presence of service profiles and Network Servers:

- Only one service profile is found.
- Multiple service profiles are found.
- Only one Network Server is found.
- Multiple Network Servers are found.
- Any combination of the above.

Regarding the presence of applications in the Application Server:



- No application is found. Here the critical part is to accept a blank list as a response.
- Only one application is found, the default one in NodesApp called 'RHAapplication'
- One or more applications are found, but none of them is named 'RHAapplication'.
- Several applications are found, one of them called 'RHAapplication'.

Regarding the device profile:

- No device profile is found. Here the critical part is to accept a blank list as a response.
- Only one device profile is found.
- One or more applications are found.
- If the device was already scanned and a device profile with the same name is already present.

## Chapter 6

# Conclusions and future work

The following conclusions can be extracted from the work carried out in this thesis:

- ChirpStack's LoRaWAN open-source technology has been selected as the best choice to bring LPWAN networks closer to the industrial sector, based on the required investment and the technological knowledge needed to deploy it.
- A fully functional web application has been developed to help deploying LoRaWAN nodes in a mobility environment.
- This application has achieved further abstraction of the inherent technology to allow unexperienced users to comfortably deploy LoRaWAN end-devices.
- The deployment of LoRaWAN nodes in a mobility environment has been facilitated by this thesis. Now, any mobile phone with internet connection can do it thanks to an easy-to-use mobile friendly web application.
- A very significant reduction in time while deploying LPWAN nodes has been achieved.
- The technical barriers that people with low technical expertise face while deploying LPWAN networks has been decreased.

These results show that this application has the potential to become an industrial tool which can help experience and unexperienced people to deploy an industrial LPWAN network. The following ideas should be considered when planning future development lines:

- Use the already implemented database to store user, or business data. For example, the IP addresses of Application Servers could be saved and selected afterwards.
- Implement an NFC scanner to support another popular choice for of end-device key equipment.



- Add a feature which enables the user to select the desired application where the node should be registered after getting the full list available in the selected Application Server.
- Further testing needs to be done, primarily to check if NodesApp can escalate.

# Bibliography

- [1] Anbar Mohammed, R. Abdullah, S. Al-Sarai, and A. B. al Hawari, “Internet of Things Market Analysis Forecast 2020-2030,” 2020.
- [2] N. B. Jones and C. M. Graham, “Can the IoT Help Small Businesses?,” *Bulletin of Science, Technology and Society*, vol. 38, no. 1–2, pp. 3–12, Feb. 2018, doi: 10.1177/0270467620902365.
- [3] Orne Brocaar, “ChirpStack, opne-source LoRaWAN® Network Server stack,” 2021. <https://www.chirpstack.io/project/> (accessed Aug. 26, 2021).
- [4] D. Ismail, M. Rahman, and A. Saifullah, “Low-Power Wide-Area Networks: Opportunities, challenges, and directions,” Jan. 2018. doi: 10.1145/3170521.3170529.
- [5] D. Soni and A. Makwana, “A SURVEY ON MQTT: A PROTOCOL OF INTERNET OF THINGS(IOT),” Aug. 2017.
- [6] Semtech Corporation, “LoRa and LoRaWAN: A Technical Overview,” 2020.
- [7] LoRa Alliance Technical Committee, “LoRaWAN Device Identification QR Codes for Automated Onboarding Technical Recommendation (TR005),” 2020.
- [8] W. Ayoub, A. Samhat, F. Nouvel, M. Mroue, and J. Prévotet, “Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs standards and Supported Mobility,” 2018, doi: 10.1109/COMST.2018.2877382i.
- [9] Semtech Corporation, “An In-depth Look at LoRaWAN® Class B Devices,” 2019. Accessed: Mar. 10, 2022. [Online]. Available: <https://lora-developers.semtech.com/library/tech-papers-and-guides/lorawan-class-b-devices/>
- [10] R. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,” 2014. [Online]. Available: <http://www.rfc-editor.org/info/rfc7231>.
- [11] “balenaEtcher.” <https://www.balena.io> (accessed Mar. 10, 2022).
- [12] “The Eclipse Foundation.” <https://www.eclipse.org/org/foundation/> (accessed Mar. 10, 2022).
- [13] “Bootstrap.” <https://getbootstrap.com/> (accessed Mar. 10, 2022).

# Annexes

The following code files:

- MQTT-client.py
- urls.py
- utils.py
- views.py

Helper files to launch the app in a virtual environment:

- README
- Requirements

Information about the experiment:

- Excel file with detailed data about the experiment.