



**School of  
Engineering**

InES Institute of  
Embedded Systems

## **Bachelor Thesis (Computer Science)**

# Autonomous Cloud Solution for Pressure Sensors

---

**Author**

---

Samuel Egger  
Lukas Raschle

---

**Main supervisor**

---

Andreas Rüst

---

**Sub supervisor**

---

Kurt Bleisch

---

**Industrial partner**

---

KELLER AG für Druckmesstechnik

---

**Date**

---

19.06.2020

---

## **Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering**

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinar massnahmen der Hochschulordnung in Kraft.

**Ort, Datum:**

**Name Studierende:**

Winterthur, 18.06.2020

Lukas Raschle

Fischingen, 18.06.2020

Samuel Egger

# Abstract

The company KELLER AG für Druckmesstechnik (KELLER) produces pressure sensors in Winterthur. For a few years now, the Long-Range Wide Area Network (LoRaWAN) network protocol has been used for the transmission of measurement data. KELLER offers its customers the KOLIBRI Cloud ([www.kolibricloud.ch](http://www.kolibricloud.ch)) for operating the pressure sensors with remote transmission functionalities. It acts as a LoRaWAN application server. In the KOLIBRI Cloud, devices are configured and measurement data is received, saved, and provided for further use. To use the KOLIBRI Cloud, a connection to the Internet is required and the data is stored on the KOLIBRI Cloud server. KELLER sales staff reported the customer's need to have the same functionality without having to transfer the measurement data to external servers via the Internet.

The objective of the thesis was to analyse the customer's needs, to define the requirements of the system, to develop a concept and to implement and validate the system as a proof of concept.

In the first step, possible local LoRaWAN solutions were analysed with the aim of covering customer needs which we knew from KELLER development staff. We created an initial broad set of specification and use cases. In the next steps, key requirements were determined together with the KELLER sales staff using a survey. Knowing the requirements and the prioritisation amongst them, we defined which subset of the specification and which use cases are implemented in the proof of concept together with KELLER.

In the proof of concept, hardware and software technologies were evaluated. The complete LoRaWAN stack was operated on a LORIX One device, an inexpensive outdoor LoRaWAN gateway with a Yocto-based Linux system. The freely usable open-source solution ChirpStack was used as LoRaWAN network and application server including data storage in PostgreSQL. For the data retrieval, we created an application called "KIWI Server" which provides an API using Go as programming language. For the visualisation of the data, we implemented the Windows program "KIWI Desktop" (.NET, WPF) to fetch the data from the LORIX One via the API in a local network.

The provided solution enables users to connect multiple low-energy LoRa devices with one cost-efficient server system. It makes it possible to use the benefits of the LoRa sensors without the need of an Internet connection and with the full control and ownership over all data. All produced software is well documented, will become open-source and therefore adaptable to third-party LoRa sensors.

# Zusammenfassung

Die Firma KELLER AG für Druckmesstechnik (KELLER) produziert in Winterthur Drucksensoren. Für die Übertragung der Messdaten wird seit einigen Jahren das Netzwerkprotokoll Long-Range Wide Area Network (LoRaWAN) verwendet. Für den Betrieb der Drucksensoren mit Fernübertragungsfunktionalitäten bietet KELLER seinen Kunden die KOLIBRI Cloud ([www.kolibricloud.ch](http://www.kolibricloud.ch)) an. Sie fungiert als LoRaWAN-Anwendungsserver. In der KOLIBRI Cloud werden Geräte konfiguriert, Messdaten empfangen, gespeichert und zur weiteren Verwendung bereitgestellt. Um die KOLIBRI Cloud zu nutzen, ist eine Verbindung zum Internet erforderlich und die Daten werden auf dem KOLIBRI Cloud-Server gespeichert. Das Verkaufspersonal von KELLER berichtete, dass sich verschiedene Kunden diese Funktionalität wünschen, ohne die Messdaten über das Internet auf externe Server übertragen zu müssen.

Ziel der Arbeit war es, die Bedürfnisse des Kunden zu analysieren, die Anforderungen an das System zu definieren, ein Konzept zu entwickeln, das System als Proof-of-Concept zu implementieren und zu validieren.

Im ersten Schritt wurden mögliche lokale LoRaWAN-Lösungen analysiert mit dem Ziel, Kundenbedürfnisse abzudecken, die wir von KELLER-Entwicklern erfahren hatten. Wir erstellten eine umfassende Spezifikation und eine Vielzahl von Anwendungsfällen. In den nächsten Schritten wurden gemeinsam mit den KELLER-Vertriebsmitarbeitern die wesentlichen Anforderungen ermittelt mit Hilfe einer Umfrage. Als wir die Anforderungen und dessen Priorisierungen kannten, definierten wir gemeinsam mit KELLER, welchen Teil der Spezifikation und welche Anwendungsfälle im Proof-of-Concept umgesetzt werden.

Beim Proof-of-Concept wurden Hardware- und Softwaretechnologien bewertet. Der komplette LoRaWAN-Stack wurde auf einem LORIX One Gerät, einem kostengünstigen Outdoor-LoRaWAN-Gateway mit einem Yocto-basierten Linux-System, betrieben. Als LoRaWAN-Netzwerk- und Applikationsserver einschliesslich Datenspeicherung in PostgreSQL wurde die frei verwendbare Open-Source-Lösung ChirpStack eingesetzt. Für den Datenabruf haben wir eine Anwendung namens "KIWI Server" erstellt, die eine API mit Go als Programmiersprache zur Verfügung stellt. Für die Visualisierung der Daten implementierten wir das Windows-Programm "KIWI Desktop" (.NET, WPF), um die Daten vom LORIX One über die API in einem lokalen Netzwerk abzurufen.

Die bereitgestellte Lösung ermöglicht es den Anwendern, mehrere energiesparende LoRa-Geräte mit einem kosteneffizienten Serversystem zu verbinden. Sie ermöglicht es, die Vorteile der LoRa-Sensoren zu nutzen, ohne dass eine Internetverbindung erforderlich ist und mit voller Kontrolle und Besitz über alle Daten. Die gesamte produzierte Software ist gut dokumentiert, wird Open Source werden und daher an LoRa-Sensoren von Drittanbietern anpassbar sein.

## Authors Note

This bachelor thesis was written by Samuel Egger and Lukas Raschle in the spring semester in 2020. The thesis was done on behalf of KELLER AG für Druckmesstechnik and in cooperation with the Institute of Embedded Systems (InES). The project was supervised by the main supervisor Prof. Andreas Rüst (InES) and the sub supervisor Kurt Bleisch (InIT)

### Note of thanks

We would like to thank Prof. Andreas Rüst and Kurt Bleisch for their valuable feedback throughout the project. Their recommendations and constructive feedback in our weekly meetings helped us to achieve the quality that we were looking for.

We would also like to thank KELLER AG für Druckmesstechnik and especially our main contact Sebastian Mojado for the enjoyable collaboration and important feedback and suggestions in all stages of the project. His know-how of the of KELLER and their customers helped us greatly to reach our goal and complete this project successfully. The fast and uncomplicated provision of required hardware like the LORIX One and the ADT1's also helped us to stick to the schedule and secure a desirable software quality.

### Naming

Since the software of KELLER has bird themed names, we also were looking to name our software bird themed. We thought an IoT network without internet is like a bird that cannot fly, so we chose the flightless bird "Kiwi" as a base name for the different software solutions. Our server software package with the key third-party component ChirpStack can be viewed as the bird nest where the Kiwi mother feeds the kiwi younglings with data. We therefore named it "ChirpNest".

# Table of Contents

1	Introduction .....	10
1.1	KELLER AG für Druckmesstechnik .....	10
1.2	Situation .....	11
1.2.1	KOLIBRI Desktop .....	11
1.2.2	KOLIBRI Mobile .....	12
1.2.3	KOLIBRI Cloud .....	13
1.3	Thesis Objective .....	14
1.4	Methodology .....	14
1.5	Thesis Structure .....	14
2	Theoretical Background .....	15
2.1	LoRa/LoRaWAN .....	15
2.1.1	End Nodes .....	15
2.1.2	Gateway .....	15
2.1.3	Network Server .....	16
2.1.4	Application .....	16
2.2	KELLER Remote Transmission Units .....	17
3	Problem Analysis .....	18
3.1	Problem Domain .....	18
3.1.1	Sensor Use Scenarios .....	18
3.1.2	Potential Customers Unable to Use the Internet .....	18
3.1.3	Potential Customers not Willing to Use the Internet .....	19
3.2	Persona .....	19
3.3	Value Proposition Canvas .....	21
3.3.1	Customer Profiles .....	22
3.3.2	Value Map .....	25
3.3.3	Conclusion .....	25
4	Requirements and Specification .....	26
4.1	System Overview .....	26
4.1.1	ADT1 .....	26
4.1.2	LoRa Stack Unit .....	26
4.1.3	KIWI Desktop .....	26
4.1.4	KIWI Web .....	27
4.2	Survey Results .....	27

4.2.1	System Setup.....	27
4.2.2	Data Storage.....	27
4.2.3	Data Access.....	27
4.2.4	Conclusion .....	27
4.3	Specification.....	28
4.3.1	LoRa Stack Unit .....	28
4.3.2	KIWI Desktop .....	29
4.3.3	KIWI Web.....	30
4.3.4	Delimitations .....	31
4.4	Use Cases .....	32
4.4.1	LoRa Stack Unit .....	34
4.4.2	KIWI Desktop .....	37
4.4.3	KIWI Web.....	47
5	Technical Analysis.....	48
5.1	LoRa Stack Unit Device .....	48
5.1.1	MultiTech Conduit .....	48
5.1.2	Wirnet Station (Kerlink) .....	48
5.1.3	LORIX One (Wifx) .....	48
5.1.4	Conclusion .....	48
5.2	LoRaWAN Stack .....	49
5.2.1	ChirpStack .....	49
5.2.2	The Things Stack .....	49
5.2.3	lorawan-server .....	49
5.2.4	Conclusion .....	49
5.3	Data Storage.....	49
5.3.1	InfluxDB .....	50
5.3.2	PostgreSQL .....	50
5.3.3	ThingsBoard.....	50
5.3.4	Conclusion .....	50
5.4	API Technology.....	50
5.4.1	ASP.NET Core .....	50
5.4.2	Go, gRPC, Protocol Buffers.....	51
5.4.3	Node.js, Express.....	51
5.4.4	Conclusion .....	51
5.5	Windows Application – Charts.....	51

5.5.1	LiveCharts.....	51
5.5.2	Microcharts .....	51
5.5.3	OxyPlot .....	52
5.5.4	Conclusion .....	52
6	Concept.....	53
6.1	Overview .....	53
6.2	ChirpNest.....	54
6.2.1	ChirpStack with Data Storage .....	54
6.2.2	KIWI Server.....	59
6.2.3	Scalability .....	60
6.3	KIWI Desktop .....	61
6.3.1	Design.....	61
6.3.2	Data Storage .....	66
6.3.3	Export .....	66
7	Implementation.....	67
7.1	System Setup.....	67
7.2	ChirpNest.....	68
7.2.1	ChirpNest Image .....	68
7.2.2	KIWI Server API .....	69
7.3	KIWI Desktop .....	74
7.3.1	Project Structure .....	74
7.3.2	NuGet Packages .....	75
7.3.3	Testing .....	75
8	Validation and Verification .....	76
8.1	Validation with KELLER .....	76
8.2	System Test .....	76
8.3	Verification: Check Specification .....	79
9	Results and Conclusions .....	82
9.1	Proof of Concept .....	82
9.2	General Findings.....	82
9.3	Further Work .....	83
	References.....	84
	Sources .....	84
	Figures.....	87
	Tables.....	88



Appendix .....	89
I. List of Abbreviations .....	89
II. Thesis Objective (Aufgabenstellung).....	90
III. Survey with Results.....	92
IV. Project Management .....	106
IV.I. Schedule.....	106
IV.II. Risk Management.....	106
V. Protocol: Verification at KELLER .....	107
VI. Used tools .....	110
VII. Manuals .....	111
VII.I. Manual: Installation of a LORIX One with an ADT1 Device.....	111
VII.II. Manual: Development of KIWI Server .....	120
VII.III. Manual: Create ChirpNest Yocto Image for LORIX One.....	122

# 1 Introduction

This chapter introduces to the bachelor thesis by introducing KELLER, describing the situation stating the thesis objective, explainin the used methodology and the structure of the thesis.

## 1.1 KELLER AG für Druckmesstechnik

KELLER AG für Druckmesstechnik (KELLER) is Europe's leading producer of isolated pressure transducers and transmitters. The family owned company is based in Winterthur, Switzerland where the production and value generation takes place. Most of the 450 employees work at the headquarters in Winterthur and the rest at various branches in Europe, Asia, South America and the USA [1].

With a variety of products ranging from small pressure transducers with analogue output over pressure transmitters to fully functional data loggers with remote transmission functionalities, KELLER sensors are used in many different applications and fields such as oil and gas, mining, aviation, water and environment, food and pharma and notably also on the International Space Station.

KELLER has over 10 years of experience with Internet of Things (IoT) systems based on data transmission over GSM modules to FTP/mail servers. With increasing popularity of IoT protocols, KELLER developed its own cloud solution called "KOLIBRI Cloud". The new data loggers with LoRaWAN transmission functionality supplement the IoT range of KELLER.

## 1.2 Situation

KELLER offers multiple software solutions for its devices. Under the product family name "KOLIBRI" there are three products for different kind of applications.

### 1.2.1 KOLIBRI Desktop

Figure 1 shows the application KOLIBRI Desktop with device connection types.

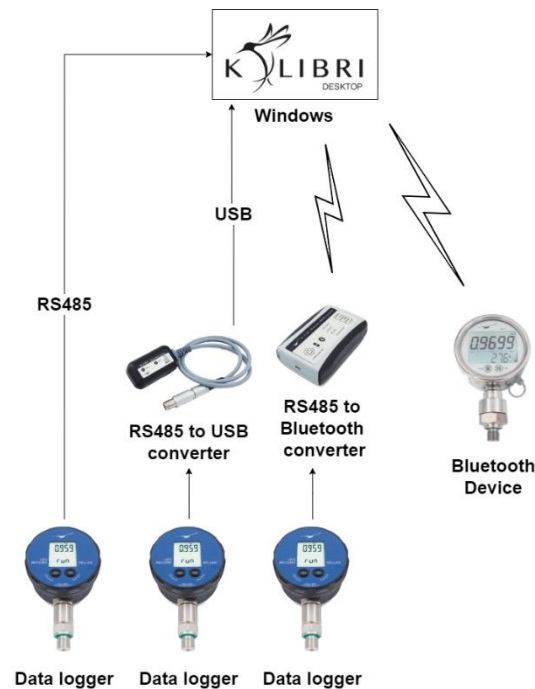


Figure 1: KOLIBRI Desktop overview

KOLIBRI Desktop is a Windows application that supports KELLER data loggers. Supported devices can be connected directly with the serial port. A converter can be used to convert the signal to USB and connect the device with an USB cable.

Some data loggers have Bluetooth classic functionality and are able to connect to the computer directly by using the Bluetooth connection. Alternatively, all data loggers can use the Bluetooth interface when using a serial port to Bluetooth converter.

### 1.2.2 KOLIBRI Mobile

Figure 2 shows the application KOLIBRI Mobile with device connection types.



*Figure 2: KOLIBRI Mobile overview*

KOLIBRI Mobile is a mobile Android app similar to KOLIBRI Desktop with a planned future iOS version. KOLIBRI Mobile supports the same devices as the desktop application but only offers the Bluetooth interface. Data loggers without a Bluetooth interface need a converter to connect as shown in Figure 2.

### 1.2.3 KOLIBRI Cloud

Figure 3 shows the application KOLIBRI Cloud with device connection types.

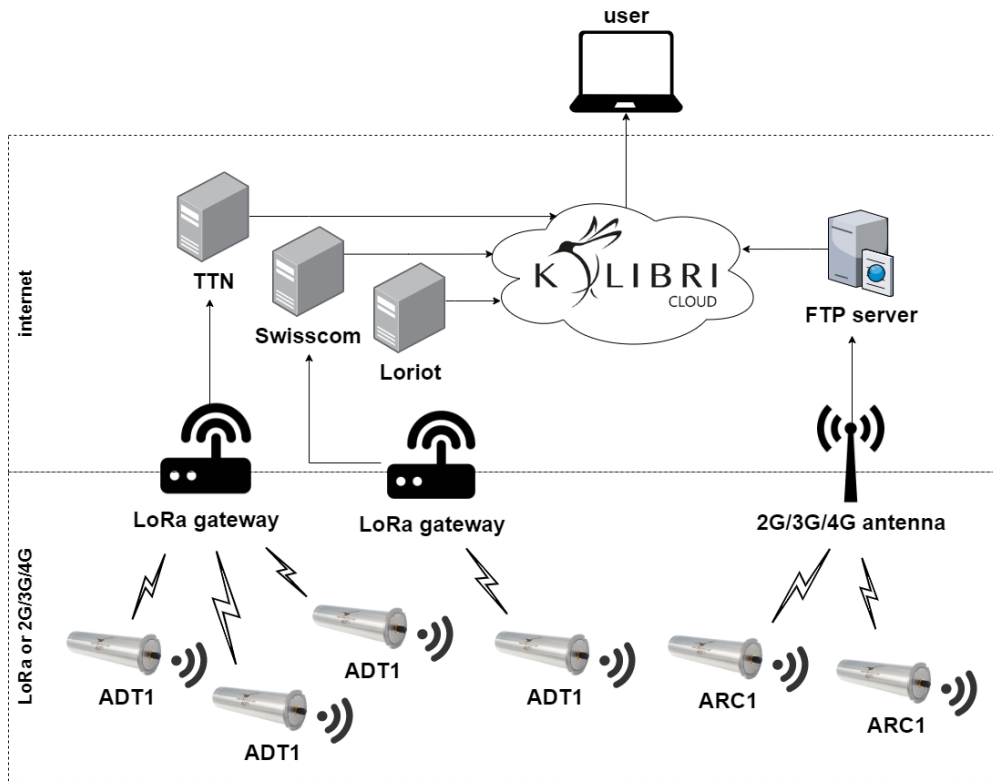


Figure 3: KOLIBRI Cloud overview

KOLIBRI Cloud is the newest member of the KOLIBRI product family. Remote transmission units such as the ADT1 (LoRaWAN) and the ARC1(2G/3G/4G/LoRaWAN) are able to send measurement data directly to KOLIBRI Cloud where the data will be stored online.

Currently, the cloud solution is only an option for the customers of KELLER if their location of operation has Internet access and when they are willing to transmit the data to the servers of KELLER where the cloud is hosted. KELLER wants to close this gap in their product range and offer a solution for the LoRaWAN devices that can be used without Internet access and where the data will never leave the infrastructure of the customer.

### 1.3 Thesis Objective

The goal is the development of an open-source solution that enables customers which are not willing or able to use the current cloud-based solution to use KELLER remote transmission units. The goal of this solution is not to build a new service provided by KELLER but to promote sales of KELLER remote transmission units to said customers by empowering them to use the benefits of these devices independently from KELLER hosted solutions.

The objective is to analyse the situation in cooperation with KELLER to understand what the needs are. With this information we shall define requirements and use cases to develop a concept that fits those needs. Based on the concept we shall implement a proof of concept as basis for further development and verify and validate the implemented solution [2].

### 1.4 Methodology

To start this project, we had a meeting with the responsible person from KELLER. We got a rough overview of the needs, ideas and their vision. Based on this information we compiled a list of requirements and use cases for which we created personas to personify those needs. The personas are used in the business model canvas to improve understanding requirements for a product.

We surveyed the sales staff of KELLER to collect their view of the needs of their customers and their ideas for solutions. Based on the results, prioritized the requirements and created a concept that covers the most important needs and requirements.

To implement the concept, we started to evaluate different hardware and software components. We first focussed on the interaction of the different components. Hence, we set up the whole stack and implemented our own components rudimentarily to ensure the chosen components would work together. Based on this, we implemented the proof of concept. We then presented the finished proof of concept to KELLER and validated it based on the requirements.

During the whole project, weekly meetings with the supervisors took place (with few exceptions). The regular exchange with KELLER was ensured due to project member Lukas Raschle working at KELLER and regularly communicating with the representative of the company. We also established contact with the creators of central components of our project, namely the manufacturer of the used LoRa gateway (Wifx) and the creator of the central open-source project ChirpStack to overcome obstacles quicker. During the project, we kept an up-to-date schedule and a list of risks to ensure the project could be implemented on time.

### 1.5 Thesis Structure

The thesis is structured according to the Methodology. We start with the explanation of basic knowledge required for the thesis in chapter “2 Theoretical Background”. In chapter “3 Problem Analysis” we analyse the problem the thesis is based upon. The results are used in chapter “4 Requirements and Specification” to create a specification and use cases. In chapter “5 Technical Analysis” we analyse and compare technical components to use them in the concept in chapter “6 Concept”. The important aspects of the implementation are explained in chapter “7 Implementation” followed by the chapter “8 Validation and Verification”. In chapter “9 Results and Conclusions” we explain the result in this thesis and propose further actions for KELLER.

## 2 Theoretical Background

In this chapter, the basics of LoRa/LoRaWAN and KELLER remote transmission units are described.

### 2.1 LoRa/LoRaWAN

Long Range Wide Area Network (LoRaWAN) is a low power wireless protocol. The specification is defined by the LoRa Alliance [3], a non-profit association. LoRaWAN is focused on energy efficiency and for ranges of up to 10 km.

Although LoRa and LoRaWAN are often used interchangeably there is a difference. LoRa describes the physical layer or the wireless modulization while LoRaWAN is the communication protocol or basically the network.

This architecture is portrayed in Figure 4.

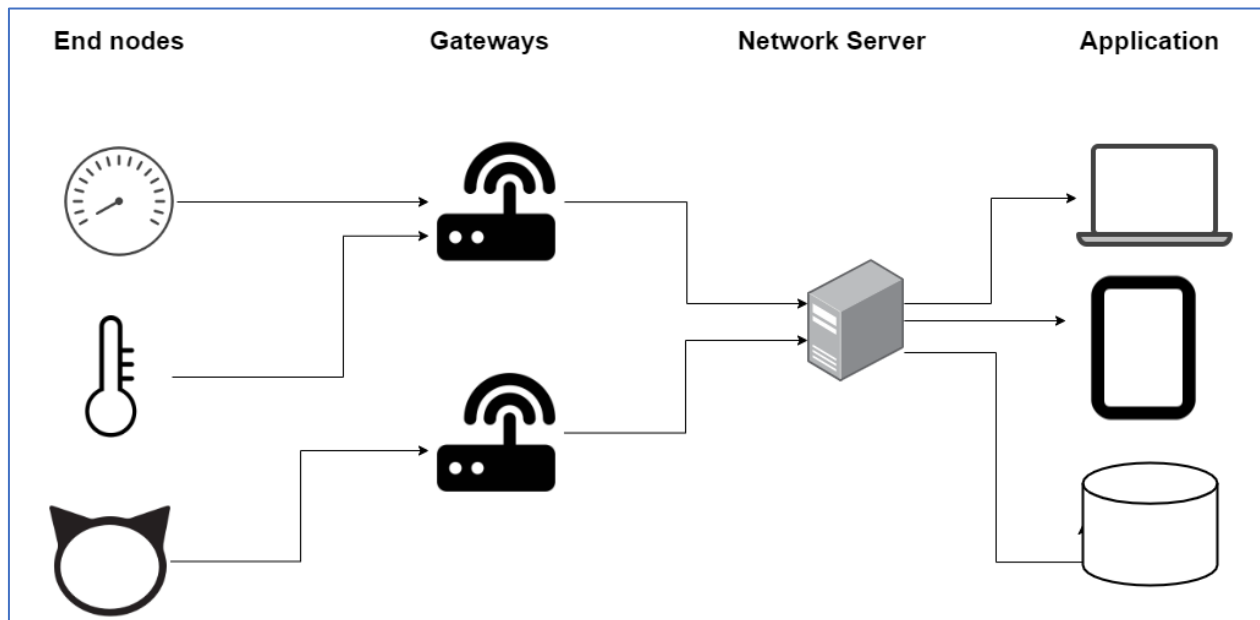


Figure 4: LoRaWAN Network

#### 2.1.1 End Nodes

The LoRa end nodes are IoT devices that send the data to the LoRa gateway. This can be a broad selection of IoT devices like temperature sensors, pressure sensors, smoke detectors, water meters, pet tracking devices and so on. In our projects the end nodes are the remote transmission unit ADT1 from KELLER.

#### 2.1.2 Gateway

The LoRa gateway is an antenna that receives the broadcast LoRa messages from the end nodes and forwards them to a network server with TCP/IP or send data back to the end nodes. Anyone can set up a gateway and every end node send the messages to every gateway in range.

### 2.1.3 Network Server

The network routes the messages to the right application and back. The network server is also responsible to eliminate duplicate messages, does a security check and sends the acknowledgement to the end nodes.

The most common network server providers are:

- The Things Network [4]
- Loriot [5]
- Swisscom (Switzerland) [6]

### 2.1.4 Application

The application is a software running on a server that handles the incoming data. This can be a range of applications that are integrated by the network server, like databases, visualizing tools or web APIs.



## 2.2 KELLER Remote Transmission Units

KELLER offers two data loggers with a LoRa module in several variations.

### ADT1

The ADT1 pictured in Figure 5 is a datalogger that comes in a tube or in a box case. The ADT1 provides remote transmission functionalities with LoRa and is a low cost and energy-saving device while being powered by three conventional AA batteries [7]. The ADT1 will be used to develop the proof of concept in this project.



*Figure 5: ADT1-tube [7]*

### ARC1

While the ARC1 pictured in Figure 6 offers all the functionality the ADT1 offers, the ARC1 can additionally transmit data via 2G, 3G and 4G. The ARC1 also comes in a tube or in a box variation. In contrast to the ADT1, the ARC1 is powered by a single replaceable DD lithium battery [8].



*Figure 6: ARC1-tube [8]*

### 3 Problem Analysis

In this chapter, we analysed and the problem from the customer's perspective.

#### 3.1 Problem Domain

This section describes the problem domain in detail.

##### 3.1.1 Sensor Use Scenarios

For non-IoT capable loggers, KELLER offers "KOLIBRI Desktop" and "KOLIBRI Mobile" for reading, visualizing and exporting the data (see "case C" in Figure 7).

Regarding remote data loggers (meaning they are IoT capable using GSM or LoRa), the KOLIBRI Cloud solution fits the needs of customers that use IoT capable sensors well and are able and willing to transmit the data via the Internet (see "case A" in Figure 7). However, sales also encountered potential customers who were unable or unwilling to transmit measurement data via the Internet (see "case B" in Figure 7). Since KELLER does not have an IoT solution for this case, the customers will have to resort to competitors or more basic solutions with less revenue for KELLER.

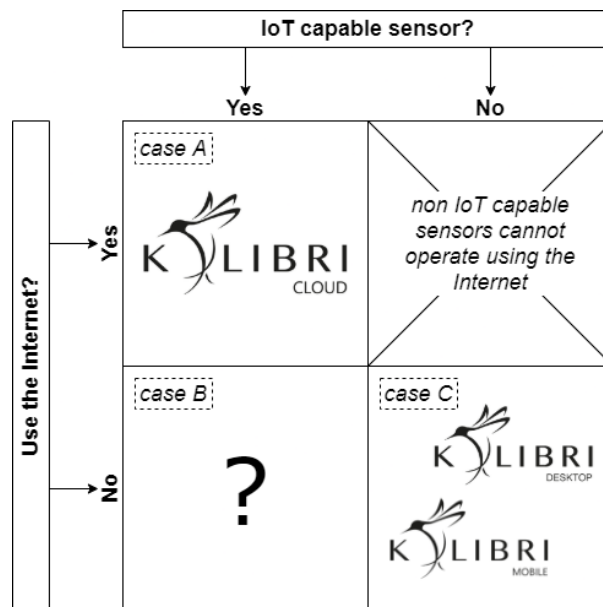


Figure 7: Scenarios of having IoT capable sensors or not and of using the Internet for data transmission or not

##### 3.1.2 Potential Customers Unable to Use the Internet

KELLER sales staff encountered potential customers that are unable to use the Internet for data transmission. The main reason is that they operate in places where there no Internet connection is possible or reasonably affordable (no cable Internet, no mobile Internet provider).

### 3.1.3 Potential Customers not Willing to Use the Internet


Sales of KELLER also encountered potential customers who had the possibility to transmit the data via the Internet but did not want to do so. The main reason for this is security concerns:

- Customers with sensitive or confidential data do not feel comfortable to save the data on external servers.
- Customers from countries with heavily regulated Internet fear the data could be compromised during transmission.
- There are customers with a basic caution with everything regarding Internet and especially with the term "Cloud"

## 3.2 Persona


To better identify the goals and needs of the customers, we created four personas. Every persona represents a potential customer with different goals and needs which are not met by the current IoT solutions offered by KELLER. All details are invented but the customer profiles have been described in conversations by KELLER employees. These Personas are then used in chapter 3.3 to find and define the requirements with a value proposition canvas.

Table 1 describes a representative of a mine in Peru.

Name	Miguel López
Photo	
Operation	Copper Mine in Peru
Details	Miguel López represents a company that has a copper mine in Peru. There is a lake near the mine and groundwater levels are sometimes high, which means the mineshafts are in danger of being flooded. To prevent this from happening, the company monitors water levels to be able to respond. The mine is in a remote location so there is no cable Internet and also no mobile Internet provider.


*Table 1: Persona "Miguel López" representing a mine in Peru*

In Table 2, a German representative of local authorities is described.

Name	Walter Schmidt
Photo	
Operation	Local Authorities, Germany
Details	Walter Schmidt is the representative of certain local authorities. They monitor the moor water level and the groundwater level in an extensive raised bog area where no mobile Internet connection is available.

*Table 2: Persona “Walter Schmidt” representing local authorities (moorland in Germany)*

Table 3 illustrates a Polish fuel company representative.

Name	Ivan Kowalski
Photo	
Operation	Fuel Oil Distribution, Poland
Details	Igor Kowalski is the representative of a company that distributes fuel oil with oil trucks. They have big storage tanks whose filling levels they monitor. Transmitting this data through the Internet into a cloud-hosted environment is not an option due to security and data protection concerns.

*Table 3: Persona “Ivan Kowalski” representing a Polish fuel company with security concerns*

Table 4 describes a representative of a Spanish art museum.


Name	Michelle González
Photo	
Operation	Art Exhibition, Spain
Details	Michelle González is the representative of an art museum. The museum wants to take measurements for a special exhibition where they want to be completely independent of any external services because visualizing the measurements is also part of the art exhibition.

Table 4: Persona “Michelle González” representing a museum in Spain

### 3.3 Value Proposition Canvas

To better determine the customer needs and in order not to miss important features we used value proposition canvases (see Figure 8) [9]. Value proposition canvases help identifying the customer needs and developing products and services that customers actually want.

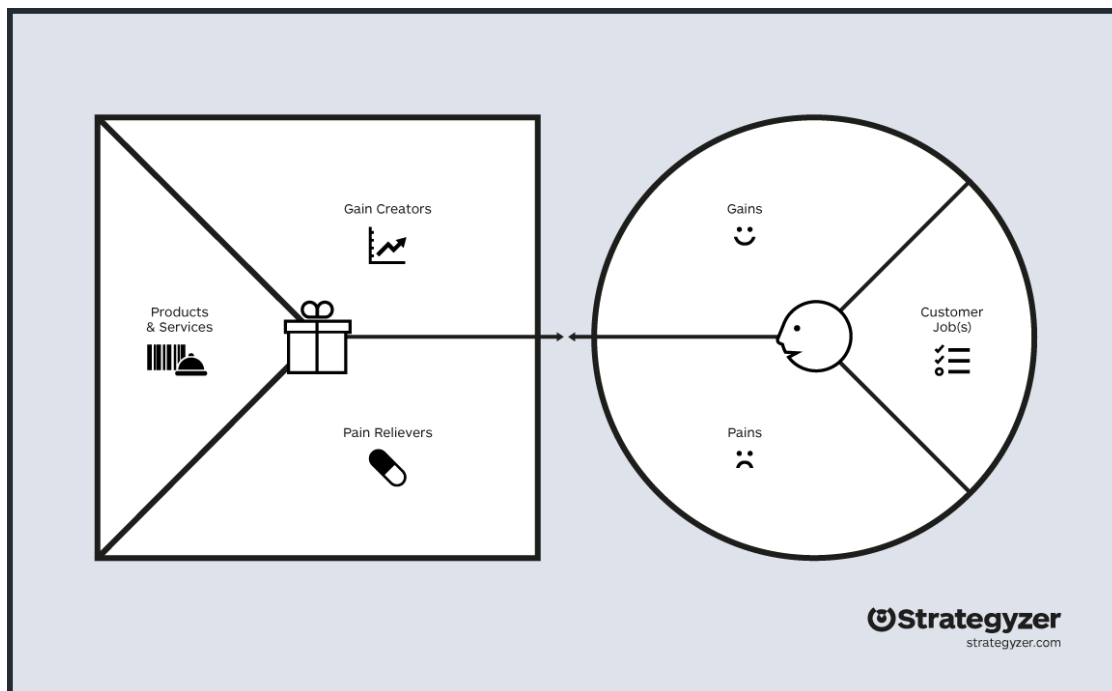


Figure 8: Value proposition canvas by Strategyzer [10]

### 3.3.1 Customer Profiles

We created a customer profile (jobs, gains, pains) for each persona since their jobs, gains and pains are different. The customer profiles are the right part of the value proposition canvas (see Figure 8 above). The blue numbers in gains and pains (e.g. 1.1) are further used to map these gains and pains to gain creators and pain relievers from the value map (see “3.3.2 Value Map”).

#### Mine in Peru, Miguel López

Figure 9 shows the customer profile part of the value proposition canvas for the persona “Miguel López” (see Table 1 above).

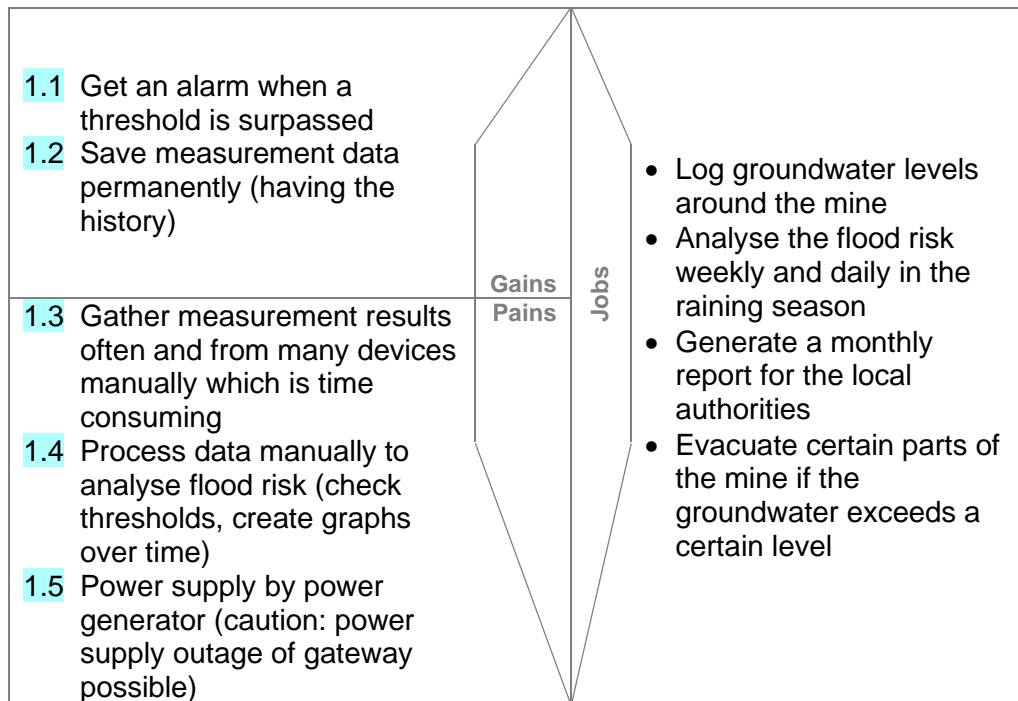


Figure 9: Value proposition canvas customer profile for “Miguel López” (mine in Peru)

### Moorland in Germany, Walter Schmidt

Figure 10 shows the customer profile part of the value proposition canvas for the persona “Walter Schmidt” (see Table 2 above).

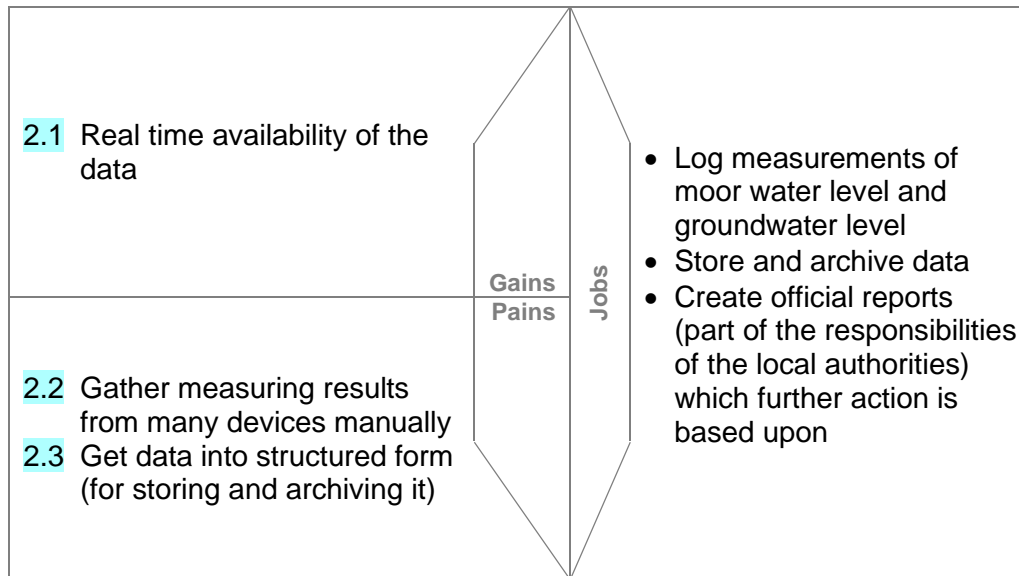


Figure 10: Value proposition canvas customer profile for “Walter Schmidt” (moorland in Germany)

### Poland, security concerns, Ivan Kowalski

Figure 11 shows the customer profile part of the value proposition canvas for the persona “Ivan Kowalski” (see Table 3 above).

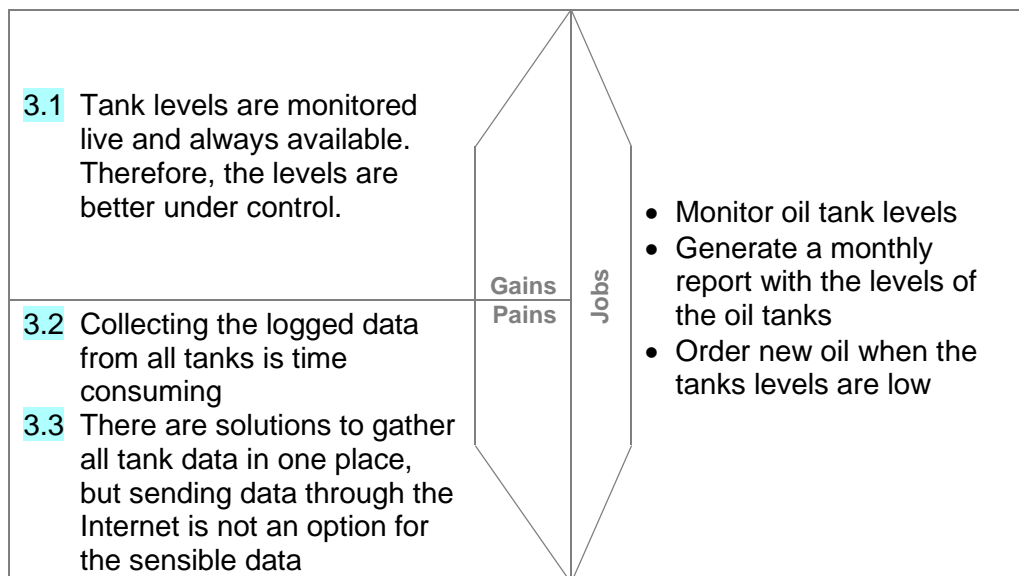


Figure 11: Value proposition canvas customer profile for “Ivan Kowalski” (Poland, security concerns)

### Museum in Spain, Michelle González

Figure 12 shows the customer profile part of the value proposition canvas for the persona “Michelle González” (see Table 4 above).

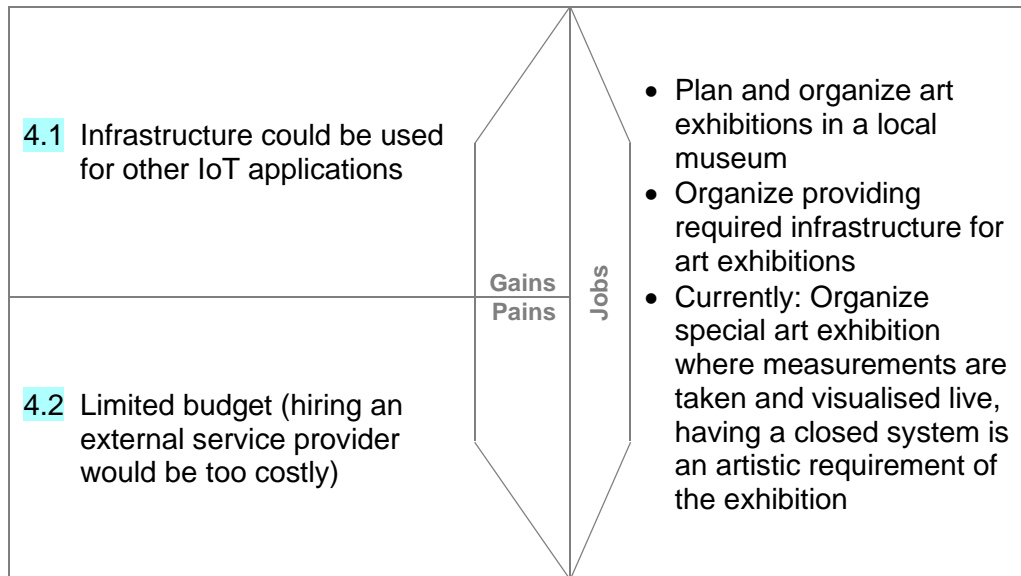


Figure 12: Value proposition canvas customer profile for “Michelle González” (museum in Spain)



### 3.3.2 Value Map

The value map is the same for all personas and is shown in Figure 13.

The blue numbers in gain creators and pain relievers (e.g. [2.1]) show which gains and pains from the customer profiles above are mapped to these gain creators and pain relievers.

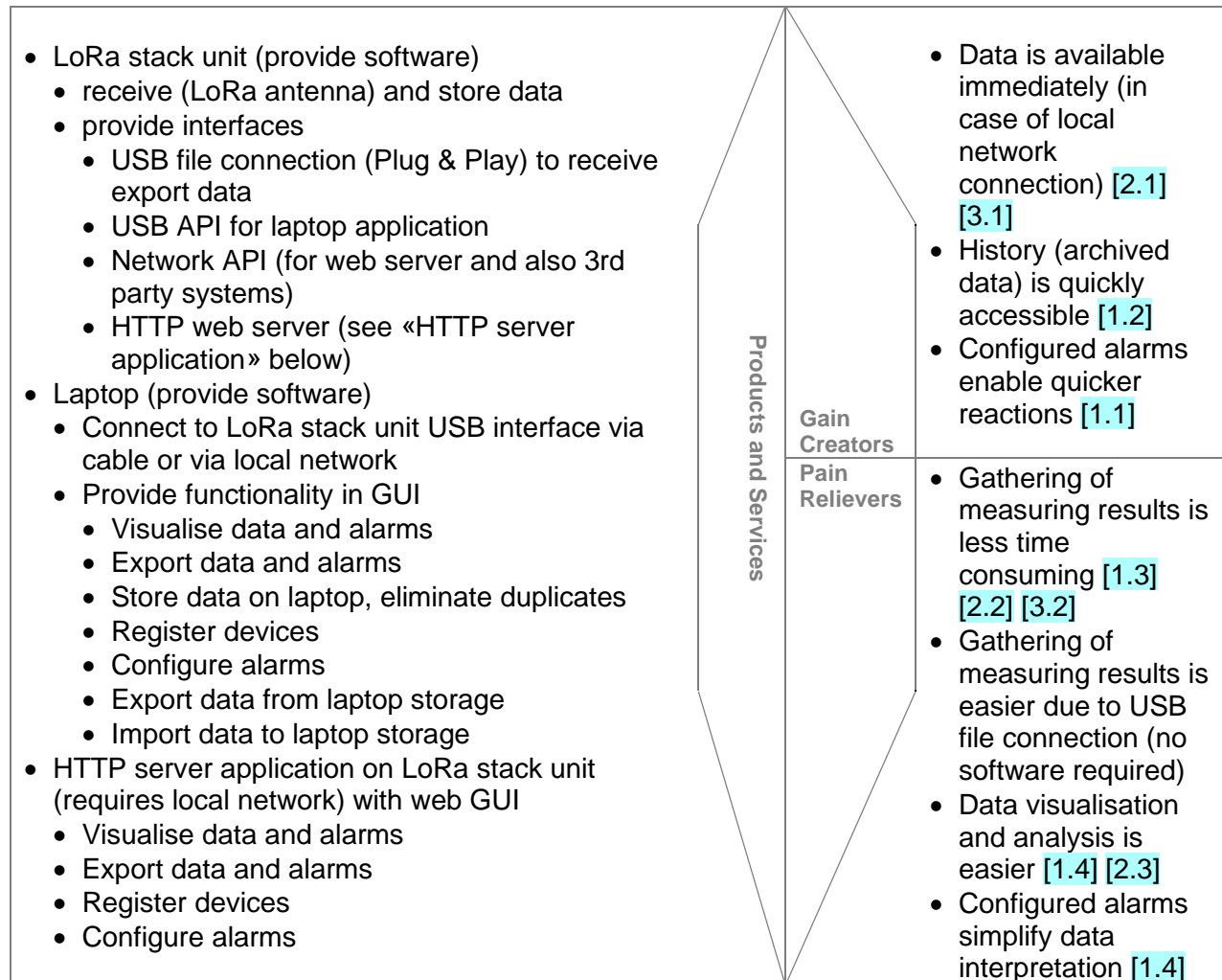


Figure 13: Value map of the value proposition canvas for all four personas

### 3.3.3 Conclusion

Creating the Value Proposition Canvases helped to examine the customer needs with their pains and gains and how a new product can help. This deeper understanding is helpful for creating the survey and the specification and use cases.

## 4 Requirements and Specification

In this chapter, an overview over the system is provided and the survey results are presented. After that, the requirements are listed in form of specifications and use cases.

### 4.1 System Overview

Based on our findings from the exchange with KELLER we designed a system overview to illustrate which components are involved in the final product and how they interact with each other.

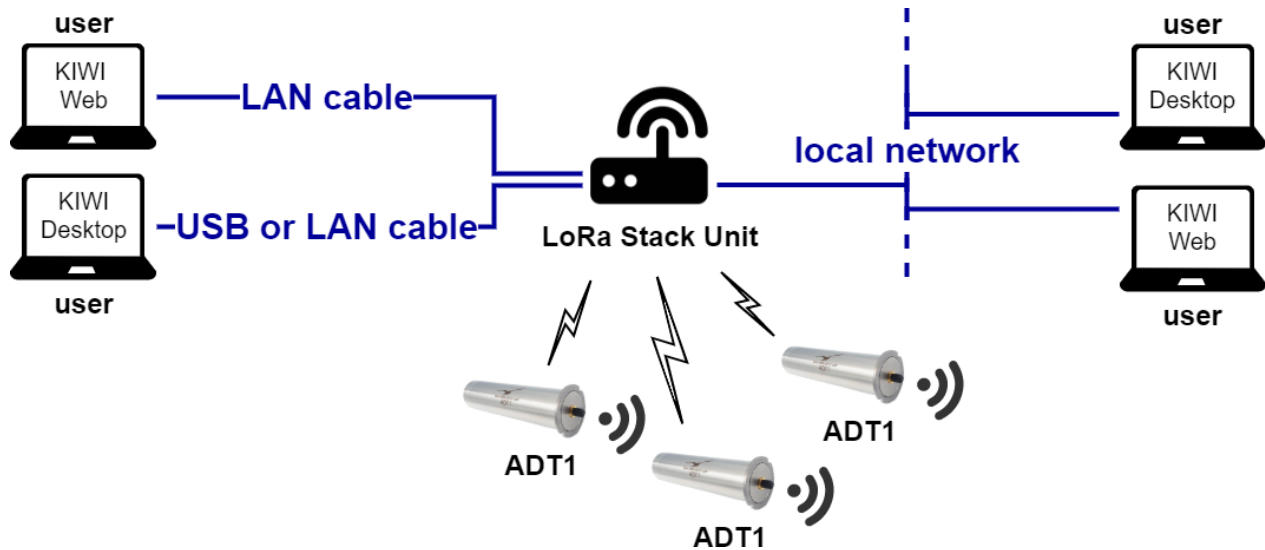


Figure 14: System Overview

#### 4.1.1 ADT1

The ADT1 is a level measuring device from KELLER. In this overview it represents any LoRa end node device from KELLER [11].

#### 4.1.2 LoRa Stack Unit

At the centre is the LoRa stack unit. The LoRa stack unit runs independently from any infrastructure except power must be provided. The LoRa stack unit receives measurements via LoRaWAN from LoRa end nodes ("ADT1" in Figure 14) and stores them locally. It provides an API for accessing the data via local network or directly via a USB or LAN cable.

#### 4.1.3 KIWI Desktop

KIWI Desktop is a software that needs to be installed on a Windows computer. Using KIWI Desktop, the user can access the measurement data from the LoRa stack unit via a local network or via a direct cable connection like a USB or LAN cable.

#### 4.1.4 KIWI Web

KIWI Web is a web application that is hosted on the LoRa stack unit. The user can access the measurement data from the LoRa stack unit using KIWI Web via a local network or via a direct LAN cable.

## 4.2 Survey Results

To narrow the scope of this project to the most useful and most important features we created a survey. This survey was aimed at the sales staff and representatives of KELLER. The goal was to find out the following:

- Were there specific projects in the past where a solution like this would have been an option?
- What is the most common setup / infrastructure of locations where this solution would come into action?
- What are the minimal features this project needs to provide to be the minimum viable product (MVP)?
- How would customers like to access the data that are gathered with the LoRa stack unit and where should the long-term storage of the data be.

The survey with the responses is in the appendix (see “III. Survey with Results”).

The key findings that were gained from this survey are listed below. The impact they have on the concept for our MVP is also stated.

#### 4.2.1 System Setup

While there is a demand for an unconnected LoRa stack unit where data is gathered with temporary cable connections, it is more likely to be used in a permanent local network.

#### 4.2.2 Data Storage

Regardless of the system setup, the majority voted for downloading the data to the laptop and working with the data stored locally. This means the LoRa stack unit should only serve as mid-term storage until the data is downloaded by a user who will be responsible for the long-term storage.

#### 4.2.3 Data Access

Since the survey indicated a clear preference for working with local data the proof of concept will focus on developing a local application intended for Windows computer.

#### 4.2.4 Conclusion

Due to the results of the survey the proof of concept will be limited as follows:

- The LoRa stack unit is permanently connected to a local network.
- The user will access the gateway with an installed Windows client application.
- The functionality of a web access will not be implemented in the proof of concept.

### 4.3 Specification

The following tables show the overall specification. The specifications that will be implemented within the proof of concept are marked with a tick (✓) in column “Implementation”. The selection of which specifications are to be implemented was discussed with KELLER.

The validation and verification is described in chapter “8 Validation and Verification” but also indicated in the following tables in the column “met”.

#### 4.3.1 LoRa Stack Unit

The LoRa stack unit specification is listed in Table 5.


Title	Implement- tation	met?
GS-01: The LoRa stack unit shall receive measurements via LoRaWAN	✓	✓
GS-02: The LoRa stack unit shall store the received measurements. The measurements shall be persisted so that the data is still available after reboot	✓	✓
GS-03: The LoRa stack unit shall be suitable for outdoor use with at least IP65 rating	✓	✓
GS-04: The LoRa stack unit offers the following hardware interfaces:		
• GS-04.1: a RJ45 interface	✓	✓
• GS-04.2: at least one common USB interface (USB-A, USB-C, USB-Mini, USB-Micro)	✓	✓
• GS-04.3: an interface for an external alarm (e.g. light or siren)	✓	✓
<i>NON-FUNCTIONAL REQUIREMENTS</i>		
GS-05: The LoRa stack unit shall be able to serve at least 10 devices sending a measurement per 30 minutes	✓	?
GS-06: When operating with the devices as stated in GS-05, the LoRa stack unit can store all the measurements for at least one year	✓	✓
GS-07: The LoRa stack unit shall run stable for several months	✓	?
GS-08: The APIs of the LoRa stack unit shall be documented in detail	✓	✓
GS-09: All 3rd party software that is used in the LoRa stack unit shall provide a license that allows free commercial use	✓	✓
GS-10: The LoRa stack unit shall be easy to set up	✗	
GS-11: The LoRa stack unit set up shall be documented in a manual	✓	✓

Table 5: LoRa stack unit specification



### 4.3.2 KIWI Desktop

KIWI Desktop will be implemented in the proof of concept. We are only able to implement a small subset of the specifications.

Since many survey results can be mapped to specifications of KIWI Desktop, we added the column “Survey importance” to the following table to better visualise how decisions were made. It contains 1 to 3 stars (☆) where more stars mean that the survey results suggest that customers favour the specification

A column “KELLER weighting” was added which contains  (meaning it having low priority) twice. The reason for the low priority of “DS-04.1: Add/remove devices” is because this can be configured on the LoRa application server (being ChirpStack Application Server, see “5.2 LoRaWAN Stack”). “DS-04.2: Add/remove alarms” has low priority because according to the experience from KELLER with their cloud, implementing alarms is extensive and thus would break the scope of this bachelor thesis.

The KIWI Desktop specification is listed in Table 6.

Title	Survey importance	KELLER weighting	Implementation	met?
DS-01: KIWI Desktop requires installation and shall run on computers with the following operating systems:				
• DS-01.1: Windows	☆☆☆		✓	✓
• DS-01.2: Linux	☆		✗	
• DS-01.3: Mac OS	☆		✗	
DS-02: The user shall be able to connect KIWI Desktop to the LoRa stack unit with a direct wired connection				
• DS-02.1: using USB	☆		✗	
• DS-02.2: using LAN	☆☆		✗	
DS-03: The user shall be able to connect KIWI Desktop to the LoRa stack unit in a local network	☆☆☆		✓	✓
DS-04: KIWI Desktop shall provide the following features while being connected to the LoRa stack unit:				
• DS-04.1: Add/remove devices	☆		✗	
• DS-04.2: Add/remove alarms	☆☆		✗	
• DS-04.3: Show measurement data in a table	☆☆		✗	
• DS-04.4: Show measurement data in charts	☆☆		✗	
• DS-04.5: Show/dismiss triggered alarms	☆☆		✗	

Title	Survey importance	KELLER weighting	Implementation	met?
• DS-04.6: Export measurement data in the following formats				
• DS-04.6.1: Excel	☆☆		✗	
• DS-04.6.2: CSV	☆☆		✗	
• DS-04.6.3: KOLIBRI Format	☆☆		✗	
• DS-04.7: Transfer measurement data directly into KIWI Desktop (data is saved locally on the computer)	☆☆☆		✓	✓
DS-05: KIWI Desktop shall provide following features with measurement data that was transferred to the computer as in DS-04.7				
• DS-05.1: Show measurement data in a table	☆☆		✓	(✓)
• DS-05.2: Show measurement data in charts	☆☆☆		✓	✓
• DS-05.3: Export measurement data in the following formats				
• DS-05.3.1: Excel	☆☆☆		✓	✓
• DS-05.3.2: CSV	☆☆☆		✓	✓
• DS-05.3.3: KOLIBRI Format	☆☆☆		✓	✓
• DS-06: Upload measurement data to the KOLIBRI Cloud	☆		✗	
<i>NON-FUNCTIONAL REQUIREMENTS</i>				
DS-07: KIWI Desktop shall be easy to use, even for non-tech-savvy users			✓	✓
DS-08: All 3rd party software that is used in KIWI Desktop shall provide a license that allows free commercial use			✓	✓
DS-09: KIWI Desktop shall be easy to set up			✗	

Table 6: KIWI Desktop specification

#### 4.3.3 KIWI Web

KIWI Web is not implemented in the proof of concept because the survey results suggest that having a desktop program (KIWI Desktop) is more important than having a web interface. Hence, the “Implementation” column is filled with “✗” for all specifications.

The KIWI Web specification is listed in Table 7.

Title	Implementation	met?
WS-01: The user shall be able to connect to the LoRa stack unit via HTTP in a local network and be able to operate the web interface using an up-to-date browser	×	
WS-02: The web interface shall provide the following features while being connected to the LoRa stack unit:		
• WS-02.1: Add/remove devices	×	
• WS-02.2: Add/remove alarms	×	
• WS-02.3: Show measurement data in a table	×	
• WS-02.4: Show measurement data in charts	×	
• WS-02.5: Show/dismiss triggered alarms	×	
• WS-02.6: Export measurement data in the following formats	×	
• WS-02.6.1: Excel	×	
• WS-02.6.2: CSV	×	
• WS-02.6.3: KOLIBRI Format	×	
<i>NON-FUNCTIONAL REQUIREMENTS</i>		
WS-03: KIWI Web shall be easy to use, even for non-tech-savvy users	×	
WS-04: All 3rd party software that is used in KIWI Web shall provide a license that allows free commercial use	×	
WS-05: KIWI Web shall be automatically set up with the setup of the LoRa stack unit	×	

Table 7: KIWI Web specification

#### 4.3.4 Delimitations

The following enumeration specifies what else is *not* part of the proof of concept:

- The API of the LoRa stack unit as well as KIWI Web shall not require authentication and encryption and hence KIWI Desktop shall also not require authentication and encryption when connecting to the LoRa stack unit.
- The LoRa stack unit and KIWI Desktop being easy to set up (GS-10 and DS-09) are kept in mind but are not prioritized within the proof of concept.
- When multiple LoRa gateways are operated simultaneously, each of them has to be a LoRa stack unit, meaning each of them has a LoRa network and application server and each of them stores the measurements. Having other LoRa gateways connecting to the network server of another LoRa stack unit is not intended in the proof of concept.

## 4.4 Use Cases

We have defined a lot of use cases that aim to cover how users of potential customers eventually interact with the system. As well as in the specification (see previous chapter “4.3 Specification”), we also have defined more use cases than we implemented in the proof of concept. The use cases which will be implemented in the proof of concept are marked green in the use case diagram (see Figure 15) and the note “will be implemented ✓” is added to the detailed use case description. Use cases that will *not* be implemented in the proof of concept are marked red in the use case diagram (see Figure 15) and the note “will not be implemented \* ✗” is added to the detailed use case description together with a “\* Proof of concept note” where the reason for not implementing the use case is stated.

The use cases are performed by three different actors:

- System manager: An advanced user with extensive technical knowledge
- Operator: An average user who carries out the daily business
- LoRa device: A LoRa end node device, e.g. an ADT1

The use cases are divided up into three chapters “4.4.1 LoRa Stack Unit”, “4.4.2 KIWI Desktop” and “4.4.3 KIWI Web”.



Figure 15 provides an overall overview over the use cases.

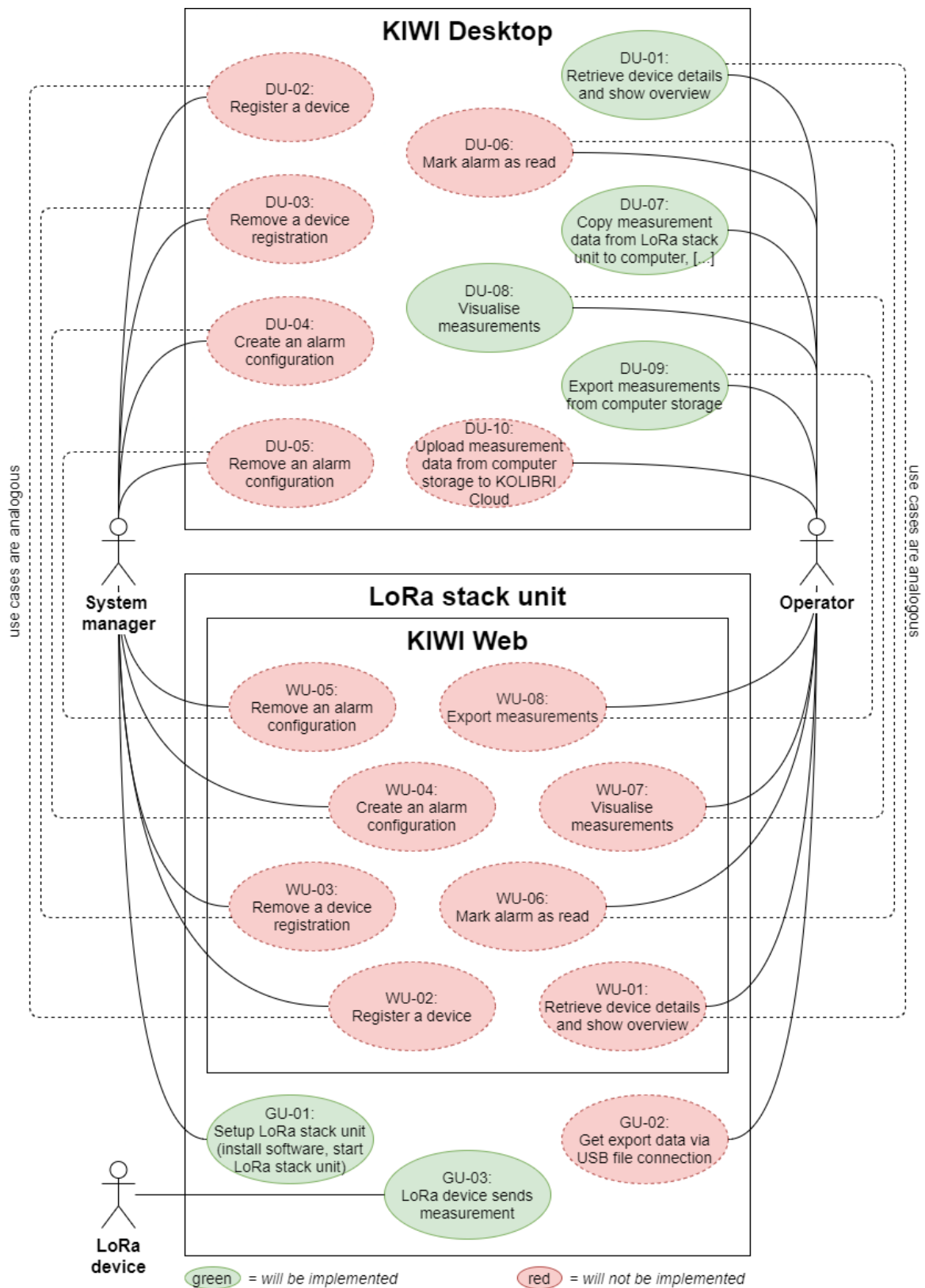


Figure 15: Use case diagram

#### 4.4.1 LoRa Stack Unit

This section contains the use cases where an actor interacts with the LoRa stack unit.

<b>Use Case GU-01: Setup LoRa stack unit (install software, start LoRa stack unit)</b>	<i>will be implemented</i> ✓
<i>Description</i> A system manager sets up and starts a LoRa stack unit.	
<i>Actors</i> <ul style="list-style-type: none"><li>• System manager</li></ul>	
<i>Precondition</i> <ul style="list-style-type: none"><li>• The system manager has the required resources:<ul style="list-style-type: none"><li>• Local network where the computer can connect to the LoRa stack unit device</li><li>• LoRa stack unit device<ul style="list-style-type: none"><li>• with power supply (over ethernet), initially not powered</li><li>• with ethernet connection to the local network</li></ul></li><li>• SD card that meets the minimum required capacity</li><li>• Computer with SD card reader<ul style="list-style-type: none"><li>• with SD card reader</li><li>• with third party software to flash on an SD card</li><li>• with SSH client software</li></ul></li><li>• Newest version of according software (later called ChirpNest)</li></ul></li><li>• The system manager is able to find out the IP address of the LoRa stack unit as soon as is available in the local network after booting.</li></ul>	
<i>Basic flow</i> <ol style="list-style-type: none"><li>1. The system manager inserts the SD card into the SD card reader connected to or integrated in the computer.</li><li>2. The system manager flashes the software on the SD card using a third-party software.</li><li>3. The system manager inserts the SD card into the SD slot of the unpowered LoRa stack unit device.</li><li>4. The system manager powers the LoRa stack unit device.</li><li>5. The system manager finds out the IP address of the LoRa stack unit (see preconditions), connects to the LoRa stack unit with SSH and later with a browser and carries out the required configuration steps which are listed in “VII.I Manual: Installation of a LORIX One with an ADT1 Device” in the appendix (page 111)</li></ol>	
<i>Alternate flow (rainy day)</i> —	
<i>Postcondition</i> The LoRa stack unit is operational. Received data is stored as soon as devices are configured.	

<b>Use Case GU-02: Get export data via USB file connection</b>		<i>will not be implemented * X</i>
<i>Description</i> An operator gathers measurement data in form of an export file.	<i>* Proof of concept note</i> This use case is not implemented because this feature has low priority according to the survey.	
<i>Actors</i> <ul style="list-style-type: none"><li>• Operator</li></ul>		
<i>Precondition</i> <ul style="list-style-type: none"><li>• The operator has the required resources:<ul style="list-style-type: none"><li>• USB cable</li><li>• Computer</li></ul></li><li>• The operator has physical access to the LoRa stack unit</li></ul>		
<i>Basic flow</i> <ol style="list-style-type: none"><li>1. The operator connects the computer and the LoRa stack unit via USB.</li><li>2. The LoRa stack unit creates a temporary filesystem containing an information text file which states that the LoRa stack unit will export all data into a file next to the information text file and that it might take some time until the export file appears.</li><li>3. The operator navigates to the mass storage device (flash drive) which is provided by the LoRa stack unit and recognized by the operating system of the computer.</li><li>4. The operator might or might not open the information text file.</li><li>5. The LoRa stack unit creates the export file (started parallel to step 3).</li><li>6. The operator finds the export file.</li><li>7. The operator copies the export file to the computer.</li><li>8. The operator terminates the USB connection.</li><li>9. The LoRa stack unit cleans up the created filesystem.</li></ol>		
<i>Alternate flow (rainy day)</i> 5a. The LoRa stack unit runs into a problem while creating the export file. The LoRa stack unit creates an error text file that is found by the user and that contains information about the problem that occurred (e.g. insufficient storage space or unexpected error with error message).		
<i>Postcondition</i> The operator has the export file on the computer.		

**Use Case GU-03: LoRa device sends measurement***will be implemented* ✓*Description*

A LoRa device sends a measurement to the LoRa stack unit which stores it locally.

*Actors*

- LoRa device

*Precondition*

- Use case GU-01 (setup of LoRa stack unit)
- Use case DU-02 (device is registered)

*Basic flow*

1. The LoRa device sends a packet containing measurement data.
2. The LoRa stack unit receives the LoRa packet and stores the measurement data in its local database (including the current time).

*Alternate flow (rainy day)*

- 2a. The LoRa stack unit does not receive the LoRa packet. In this case, the packet with the measurement data is lost and nothing else happens.

*Postcondition*

The LoRa stack unit has stored the received measurement data in its local database.

#### 4.4.2 KIWI Desktop

This section contains the use cases where an actor interacts with KIWI Desktop.

<b>Use Case DU-01: Retrieve device details and show overview</b>	<i>will be implemented</i> ✓
<i>Description</i> An operator retrieves device details from the LoRa stack unit and sees an overview.	
<i>Actors</i> <ul style="list-style-type: none"><li>• Operator</li></ul>	
<i>Precondition</i> <ul style="list-style-type: none"><li>• Use case GU-01 (setup of LoRa stack unit)</li><li>• KIWI Desktop is installed on the operator's computer.</li><li>• The computer of the operator and the LoRa stack unit are in the same local network and the operator knows the IP address of the LoRa stack unit.</li></ul>	
<i>Basic flow</i> <ol style="list-style-type: none"><li>1. The operator enters the IP address of the LoRa stack unit to establish a connection.</li><li>2. KIWI Desktop requests the device details from the LoRa stack unit containing:<ol style="list-style-type: none"><li>a. Date and time of the oldest and the newest measurement</li><li>b. Number of available measurements</li><li>c. Device name</li><li>d. Unique identifier (device EUI)</li><li>e. Possibly more available details</li></ol></li><li>3. KIWI Desktop shows the above device details in an overview.</li></ol>	
<i>Alternate flow (rainy day)</i> <ol style="list-style-type: none"><li>1a. When KIWI Desktop cannot connect to the LoRa stack unit, a corresponding information is displayed.</li></ol>	
<i>Postcondition</i> The operator sees the device details overview in KIWI Desktop.	

Use Case DU-02: Register a device		will not be implemented * ✕
<div>Description</div> <div>An operator registers a new device on the LoRa stack unit.</div>		<div>* Proof of concept note</div> <div>This use case is not implemented because the ChirpStack Application Server provides a web interface where devices can be configured (see “VII.I Manual: Installation of a LORIX One with an ADT1 Device” in the appendix on page 111) which is sufficient.</div>
<div>Actors</div> <div><ul style="list-style-type: none"><li>• Operator</li></ul></div>		
<div>Precondition</div> <div><ul style="list-style-type: none"><li>• Use case DU-01 (retrieved device details, show overview)</li><li>• Connection to the LoRa stack unit can be established</li></ul></div>		
<div>Basic flow</div> <div><ol style="list-style-type: none"><li>1. The operator navigates to the device register form.</li><li>2. The operator fills in at least the following data:<ol style="list-style-type: none"><li>a. Device address</li><li>b. Device name</li><li>c. Network session key</li><li>d. App session key</li></ol></li><li>3. The operator submits the data which is then validated by KIWI Desktop (allowed symbols and required number of characters in address and keys) and sent to the LoRa stack unit.</li><li>4. The LoRa stack unit receives the data and internally registers and activates the device and sends a confirmation to KIWI Desktop.</li><li>5. The operator sees the confirmation displayed by KIWI Desktop.</li></ol></div>		
<div>Alternate flow (rainy day)</div> <div><ol style="list-style-type: none"><li>3a. When one or more fields are not valid (containing false symbols or not the required number of characters), KIWI Desktop keeps the filled form and shows a corresponding hint where the input is invalid. The operator can change the input and submit it again (go back to step 2).</li><li>3b. If the connection to the LoRa stack unit cannot be established or times out or an error occurred when registering the device on the LoRa stack unit, the error message is displayed to the operator. Go back to step 2 (form is still filled).</li></ol></div>		
<div>Postcondition</div> <div>The device is registered and the LoRa stack unit can receive measurements from the device.</div>		

<b>Use Case DU-03: Remove a device registration</b>		<i>will not be implemented * X</i>
<i>Description</i> An operator removes a registered device from the LoRa stack unit.	<i>* Proof of concept note</i> This use case is not implemented because the ChirpStack Application Server provides a web interface where devices can be removed (see “VII.I Manual: Installation of a LORIX One with an ADT1 Device” in the appendix on page 111) which is sufficient.	
<i>Actors</i> <ul style="list-style-type: none"><li>• Operator</li></ul>		
<i>Precondition</i> <ul style="list-style-type: none"><li>• Use case DU-01 (retrieved device details, show overview)</li><li>• Connection to the LoRa stack unit can be established</li></ul>		
<i>Basic flow</i> <ol style="list-style-type: none"><li>1. The operator triggers the “remove device” action on a device which is accessible in the overview from use case DU-01.</li><li>2. KIWI Desktop shows a hint that this action cannot be undone and that all measurements of this device will also be permanently removed.</li><li>3. The operator confirms the removal.</li><li>4. The device and all affiliated measurements are removed from the LoRa stack unit and a confirmation is sent to KIWI Desktop.</li><li>5. The operator sees the confirmation displayed by KIWI Desktop.</li></ol>		
<i>Alternate flow (rainy day)</i> <ol style="list-style-type: none"><li>2a. The operator decides not to remove the device with its measurements and cancels the action.</li><li>3a. If the connection to the LoRa stack unit cannot be established or times out or an error occurred when removing device and measurements on the LoRa stack unit, the error message is displayed to the operator.</li></ol>		
<i>Postcondition</i> The device and all affiliated measurements are removed from the LoRa stack unit.		

Use Case DU-04: Create an alarm configuration		will not be implemented * ✕
<p>Description</p> <p>An operator configures a new alarm on the LoRa stack unit.</p>		<p>* Proof of concept note</p> <p>This use case is not implemented because all alarm features are not part of the proof of concept because it would break the scope of this bachelor thesis (see chapter “4.3.2 KIWI Desktop”).</p>
<p>Actors</p> <ul style="list-style-type: none"><li>• Operator</li></ul>		
<p>Precondition</p> <ul style="list-style-type: none"><li>• Use case DU-01 (retrieved device details, show overview)</li><li>• Use case DU-02 (device is registered)</li><li>• Connection to the LoRa stack unit can be established</li></ul>		
<p>Basic flow</p> <ol style="list-style-type: none"><li>1. The operator navigates to the alarm creation form by selecting the “create alarm” action on a device which is accessible in the overview from use case DU-01.</li><li>2. The operator fills in the following data:<ol style="list-style-type: none"><li>a. Device</li><li>b. Channel (e.g. pressure)</li><li>c. Mode<ol style="list-style-type: none"><li>i. Pass threshold (threshold value required)</li><li>ii. Fall below threshold (threshold value required)</li><li>iii. Last measurement is more than a specific duration ago (duration value required, e.g. 1 day)</li></ol></li></ol></li><li>3. The operator submits the data which is then validated by KIWI Desktop (device, channel and mode are selected, required value is provided) and sent to the LoRa stack unit.</li><li>4. The LoRa stack unit receives the data, internally saves the alarm configuration and sends a confirmation to KIWI Desktop.</li><li>5. The operator sees the confirmation displayed by KIWI Desktop.</li><li>6. The operator fills out the form and sends it.</li></ol>		
<p>Alternate flow (rainy day)</p> <ol style="list-style-type: none"><li>3a. When one or more fields are not valid, KIWI Desktop keeps the filled form and shows a corresponding hint where the input is invalid. The operator can change the input and submit it again (go back to step 2).</li><li>3b. If the connection to the LoRa stack unit cannot be established or times out or an error occurred when creating the alarm configuration on the LoRa stack unit, the error message is displayed to the operator. Go back to step 2 (form is still filled).</li></ol>		
<p>Postcondition</p> <p>The alarm is configured on the LORIX One.</p>		



<b>Use Case DU-05: Remove an alarm configuration</b>		<i>will not be implemented * ✕</i>
<i>Description</i> An operator removes a configured alarm from the LoRa stack unit.	<i>* Proof of concept note</i> This use case is not implemented because all alarm features are not part of the proof of concept because it would break the scope of this bachelor thesis (see chapter “4.3.2 KIWI Desktop”).	
<i>Actors</i> <ul style="list-style-type: none"><li>• Operator</li></ul>		
<i>Precondition</i> <ul style="list-style-type: none"><li>• Use case DU-01 (retrieved device details, show overview)</li><li>• Use case DU-04 (alarm is configured)</li><li>• Connection to the LoRa stack unit can be established</li></ul>		
<i>Basic flow</i> <ol style="list-style-type: none"><li>1. The operator navigates to the alarms overview of a device.</li><li>2. The operator triggers the “remove alarm configuration” action on an alarm.</li><li>3. KIWI Desktop shows a hint that this action cannot be undone.</li><li>4. The operator confirms the removal.</li><li>5. The alarm configuration is removed from the LoRa stack unit and a confirmation is sent to KIWI Desktop.</li><li>6. The operator sees the confirmation displayed by KIWI Desktop.</li></ol>		
<i>Alternate flow (rainy day)</i> <ol style="list-style-type: none"><li>3a. The operator decides not to remove the alarm configuration and cancels the action.</li><li>5a. If the connection to the LoRa stack unit cannot be established or times out or an error occurred when removing the alarm configuration on the LoRa stack unit, the error message is displayed to the operator.</li></ol>		
<i>Postcondition</i> The alarm configuration is removed from the LoRa stack unit.		

Use Case DU-06: Mark alarm as read		will not be implemented * ✕
<p>Description</p> <p>An operator marks an alarm that was triggered before as read.</p>		<p>* Proof of concept note</p> <p>This use case is not implemented because all alarm features are not part of the proof of concept because it would break the scope of this bachelor thesis (see chapter “4.3.2 KIWI Desktop”).</p>
<p>Actors</p> <p>Operator</p>		
<p>Precondition</p> <ul style="list-style-type: none"><li>• Use case DU-04 (alarm is configured)</li><li>• Use case GU-03 (LoRa device sends measurement) → an alarm must have been triggered by the measurement</li><li>• Use case DU-01 (retrieved device details, show overview)</li><li>• Connection to the LoRa stack unit can be established</li></ul>		
<p>Basic flow</p> <ol style="list-style-type: none"><li>1. The operator triggers the “mark alarm as read” action on a displayed triggered alarm in the overview.</li><li>2. The triggered alarm is marked as read on the LoRa stack unit and a confirmation is sent to KIWI Desktop.</li><li>3. The operator sees the confirmation displayed by KIWI Desktop.</li></ol>		
<p>Alternate flow (rainy day)</p> <ol style="list-style-type: none"><li>2a. If the connection to the LoRa stack unit cannot be established or times out or an error occurred when marking the triggered alarm as read on the LoRa stack unit, the error message is displayed to the operator.</li></ol>		
<p>Postcondition</p> <p>The triggered alarm is marked as read on the LoRa stack unit.</p>		

**Use Case DU-07: Copy measurement data from LoRa stack unit to computer, eliminate duplicates**

*will be implemented* ✓

*Description*

An operator copies the measurement data from a LoRa stack unit to the computer. In case there already exist measurements of same devices on the computer gathered from another LoRa stack unit, duplicates are eliminated.

*Actors*

- Operator

*Precondition*

- Use case DU-02 (device is registered)
- Use case GU-03 (LoRa device sends measurement)
- Use case DU-01 (retrieved device details, show overview)
- Connection to the LoRa stack unit can be established

*Basic flow*

1. The operator triggers the “download measurements” action on a device which is accessible in the overview from use case DU-01.
2. The measurements of the device are transmitted to KIWI Desktop. In case there already exist measurements of the according device which might come from another LoRa stack unit, these duplicates are eliminated and not stored multiple times.
3. The operator sees a confirmation as soon as the transmission has finished.

*Alternate flow (rainy day)*

- 2a. If the connection to the LoRa stack unit cannot be established or times out or an error occurred on the LoRa stack unit when transmitting measurements, the error message is displayed to the operator.

*Postcondition*

The measurements are stored on the computer within KIWI Desktop.

<b>Use Case DU-08: Visualise measurements</b>	<i>will be implemented</i> ✓
<i>Description</i> An operator views visualised measurements of a device.	
<i>Actors</i> <ul style="list-style-type: none"> <li>• Operator</li> </ul>	
<i>Precondition</i> <ul style="list-style-type: none"> <li>• Use case DU-07 (measurements are stored on the computer)</li> </ul>	
<i>Basic flow</i> <ol style="list-style-type: none"> <li>1. The operator navigates to the according locally stored measurement data of a device.</li> <li>2. KIWI Desktop displays a diagram containing all measurements of all channels.</li> </ol>	
<i>Alternate flow (rainy day)</i> —	
<i>Postcondition</i> The operator sees the visualised measurements.	

**Use Case DU-09: Export measurements from computer storage***will be implemented* ✓*Description*

An operator obtains an export file containing all locally stored measurement data of a device.

*Actors*

- Operator

*Precondition*

- Use case DU-07 (measurements are stored on the computer)

*Basic flow*

1. The operator navigates to the according locally stored measurement data of a device and triggers the “export” action.
2. The operator can select the format (Excel, CSV or KOLIBRI Format), the save path and the filename.
3. The operator submits the form.
4. KIWI Desktop creates the according file containing the measurement data.
5. KIWI Desktop displays a confirmation.

*Alternate flow (rainy day)*

- 4a. When the file cannot be created, KIWI Desktop displays an according message. Go back to step 2 (form is still filled).

*Postcondition*

The operator has the export file containing the measurement data in the file system.

<b>Use Case DU-10: Upload measurement data from computer storage to KOLIBRI Cloud</b>		<i>will not be implemented * ✕</i>
<i>Description</i> An operator uploads measurement data from the computer to the KOLIBRI Cloud via the provided API.	<i>* Proof of concept note</i> This use case is not implemented because this feature has low priority according to the survey (see chapter “4.3.2 KIWI Desktop”).	
<i>Actors</i> <ul style="list-style-type: none"><li>• Operator</li></ul>		
<i>Precondition</i> <ul style="list-style-type: none"><li>• Use case DU-07 (measurements are stored on the computer)</li><li>• Connection to the KOLIBRI Cloud (<a href="http://www.kolibricloud.ch">www.kolibricloud.ch</a>) can be established (Internet connection required)</li><li>• The operator is registered in the KOLIBRI Cloud and is able to authenticate himself</li></ul>		
<i>Basic flow</i> <ol style="list-style-type: none"><li>1. The operator navigates to the according locally stored measurement data of a device and triggers the “upload to KOLIBRI Cloud” action.</li><li>2. The operator authenticates against the KOLIBRI Cloud.</li><li>3. KIWI Desktop uploads the measurements.</li><li>4. KIWI Desktop displays a confirmation.</li></ol>		
<i>Alternate flow (rainy day)</i> 2a. When authentication failed, an error message is displayed and KIWI Desktop offers to try it again or cancel the upload action. 3a. When an error occurred during the upload, the error message is displayed to the operator.		
<i>Postcondition</i> The measurement is uploaded and can be accessed in KOLIBRI Cloud.		

### 4.4.3 KIWI Web

This section contains the use cases where an actor interacts with KIWI Web.

For each use case, an analogous use case is already described in detail in the previous section “4.4.2 KIWI Desktop”. In this section, only the key differences to the analogous use cases are described to avoid redundancies.

#### **Use Case WU-01: Retrieve device details and show overview**

is analogous to DU-01

Different precondition:

The operator does not require an installation of KIWI Desktop but a browser.

#### **Use Case WU-02: WU-02: Register a device**

is analogous to DU-02

#### **Use Case WU-03: Remove a device registration**

is analogous to DU-03

#### **Use Case WU-04: Create an alarm configuration**

is analogous to DU-04

#### **Use Case WU-05: Remove an alarm configuration**

is analogous to DU-05

#### **Use Case WU-06: Mark alarm as read**

is analogous to DU-06

#### **Use Case WU-07: Visualise measurements**

is analogous to DU-08

Different precondition:

Measurements do not need to be stored on the computer (DU-07) but a connection to the LoRa stack unit is required instead.

#### **Use Case WU-08: Export measurements**

is analogous to DU-09

Different precondition:

Measurements do not need to be stored on the computer (DU-07) but a connection to the LoRa stack unit is required instead.

## 5 Technical Analysis

In this chapter we analyse what components we plan to use in the proof of concept.

### 5.1 LoRa Stack Unit Device

The central part of this project is the LoRa stack unit device on which the following software will run: The LoRa stack (gateway bridge, network server and application server), the data storage and the API for accessing the data. The requirements for the LoRa stack unit are:

- Waterproof least ingress protection code: IP65
- Interfaces:
  - USB-mini or USB-micro
  - RJ45 (Ethernet)

For these requirements we found the three options listed in Table 8.

<b>Device</b>	MultiTech Conduit	Wirnet Station	LORIX One
<b>Producer</b>	MultiTech	Kerlink	Wifx
<b>Approximate price</b>	CHF 700.-	CHF 1800.-	CHF 550.-
<b>Interfaces</b>	LAN/GSM	LAN/GSM	LAN

*Table 8: LoRa gateway comparison [12] [13] [14]*

#### 5.1.1 MultiTech Conduit

The MultiTech Conduit is a mid-priced and highly configurable and scalable LoRa gateway for industrial IoT applications. The device has an IP67 rating and has optional interface extensions for 2G, 3G, LTE and 4G [12].

#### 5.1.2 Wirnet Station (Kerlink)

The Wirnet Station is a robust outdoor gateway in the high-priced segment. The device has an IP67 rating and is made for harsh conditions and supports a SIM-card for a GSM connection [13].

#### 5.1.3 LORIX One (Wifx)

The LORIX One is a mid- to low-priced gateway with an IP65 rating. The gateway offers a slot for an SD card where it can boot from [14].

#### 5.1.4 Conclusion

While the Wirnet Station would probably be the most robust choice, it may be too high priced for some customers. The LORIX One and the MultiTech Conduit are in a similar price segment. Since KELLER has experience with all three gateways, they prefer the LORIX One since the MultiTech conduit is very complex to configure and manage. Therefore, we selected the LORIX One as gateway for this project.



## 5.2 LoRaWAN Stack

This section contains the evaluation of the software that provides the LoRaWAN stack (gateway bridge, network server and application server) which must run on the LoRa stack unit that is the LORIX One. The main criteria for the software components are them to be able to run on the LORIX One, allowed free commercial use, open-source and a good documentation.

### 5.2.1 ChirpStack

ChirpStack includes all required components, is open-source and maintained and allows the free commercial use under the MIT license [15]. With “ChirpStack Gateway OS”, an operation system is provided where that contains the components and requires little effort for setup [16]. There is even a ChirpStack Gateway OS image provided for the LORIX One.

### 5.2.2 The Things Stack

The Things Stack provides a LoRaWAN network server and application server. The gateway bridge is not included. The provided software is open-source, actively maintained and the free commercial use is allowed under the Apache-2.0 license [17]. If running The Things Stack on the LORIX One is possible would need to be tested.

### 5.2.3 lorawan-server

The lorawan-server (“compact server for private LoRaWAN networks”) provides a LoRaWAN network server and application but no gateway bridge like The Things Stack. The software is open-source, actively maintained and free commercial use is allowed under the MIT license [18]. For this software it also would need to be tested if it runs on the LORIX One.

### 5.2.4 Conclusion

ChirpStack provides a complete solution that meets all the criteria and even provides an image that runs on the LORIX One. We could have looked for other software that provides the gateway functionality and combine it with The Things Stack or lorawan-server, but we have refrained from doing so because ChirpStack as a whole is really suitable.

## 5.3 Data Storage

The measurements received by the ChirpStack Application Server must be persisted on the LORIX One. We evaluated several possibilities. The most important criteria are the capability of running smoothly on the LORIX One (128 MB RAM), a simple and quick integration with the ChirpStack Application Server and a short time to set it up. Of course, the standard requirements to most database systems must be met (i.e. atomicity, consistency, isolation, durability).

The evaluated database systems are limited to the ones that are directly integrated by ChirpStack Application Server [19].

### 5.3.1 InfluxDB

InfluxDB is a time series database. The hardware sizing guidelines suggest providing at least 2 GB of RAM to run InfluxDB OSS (the open-source variant) [20]. However, it fits well to save measurements since it is a time series database and provides useful IoT functions.

### 5.3.2 PostgreSQL

The relational database management system PostgreSQL is already used by ChirpStack to operate. It therefore is certainly available and running on the LORIX One. The PostgreSQL integration in the ChirpStack Application Server is global as opposed to InfluxDB and ThingsBoard which are to be configured per application. While there are no specific minimum hardware requirements specified, the numbers which are stated under "Resource Consumption" and "Memory" are about in the realm of what the LORIX One can provide [21].

### 5.3.3 ThingsBoard

ThingsBoard is an IoT platform that among other features supports data collection. It has a wide range of features and requires Java 8 to run. The manual for installing it on Ubuntu Server states that for a usage together with PostgreSQL, at least 1 GB of RAM is required [22]. The available installation guides suggest it is not designed to run on embedded systems.

### 5.3.4 Conclusion

PostgreSQL has the big advantage of already being installed and operational on the LORIX One since ChirpStack also uses it. InfluxDB and ThingsBoard would both need to be added to the Yocto image, which might be very time consuming. Also, InfluxDB and ThingsBoard seem to require too many resources to operate so they both will not be able to run on the LORIX One. Since PostgreSQL also satisfies the standard requirements to databases and is widely used and for the reasons above, we decided to use PostgreSQL to store the measurements.

## 5.4 API Technology

To retrieve the measurement data which is stored in the PostgreSQL database we need to create an application (called "KIWI Server") which provides an API. The technology used for this application is evaluated here.

The main criteria for KIWI Server are the ability to run on the LORIX One, robustness, performance and a short implementation time which does not break the scope of this bachelor thesis.

### 5.4.1 ASP.NET Core

ASP.NET Core is our first choice in terms of our knowledge and experience. Getting an ASP.NET Core application to run on the LORIX One seems to be hard if not impossible. We were not able to find a ready to use Yocto recipe that provides ASP.NET Core and trying to achieve doing it ourselves is likely to be very time consuming.

Also, a .NET Core application with the .NET Common Language Runtime (CLR) is rather resource intensive.

### 5.4.2 Go, gRPC, Protocol Buffers

The ChirpStack Application Server uses Go as programming language and provides an API using gRPC and protocol buffers [23]. Since the ChirpStack application server runs on the LORIX One, it has already been shown that using this technology works with the hardware. The robustness is the same as for the ChirpStack application server since it is the same technology. The implementation time should fit the boundaries of the bachelor thesis since the open-sourced code of ChirpStack application server can be used as reference.

### 5.4.3 Node.js, Express

Using Node.js and Express (expressjs.com) on the LORIX One would probably be possible since there are lots of Yocto recipes out there for that. Running Node.js is quite resource intensive since the code is interpreted rather than it being compiled in advance. This might also compromise robustness and performance.

### 5.4.4 Conclusion

Go, gRPC and protocol buffers fulfil the criteria the best since it is already shown to work well on the LORIX One. Robustness and performance are certainly not worse than in ChirpStack Application Server and the implementation time can be kept low even though we do not have experience in it since ChirpStack Application Server functions as a good code example. Using ASP.NET Core or Node.js with express is too risky because the performance and robustness might be insufficient and setting it up would be far more time intensive.

## 5.5 Windows Application – Charts

For the visualisation of the measurements in KIWI Desktop we are looking for a simple WPF library that provides basic chart functionalities with the ability to handle more complex requirements for further development. Since the project will be open-source, we are looking for a free solution with an according license and preferably open-source.

### 5.5.1 LiveCharts

According to the GitHub statistics, LiveCharts is the most used open-source chart library in WPF projects. While it has many stars and forks on GitHub, there are no new commits since over a year, which suggests it is no longer worked on [24].

### 5.5.2 Microcharts

Microcharts is an open-source chart library under an MIT license originally built for Xamarin. Since the built-in views are compatible with .NET Standard 1.4 it would also work with our WPF application. The GitHub statistics suggest that the activity on the Microcharts project is decreasing with no new commits in the last few months [25].

### 5.5.3 OxyPlot

OxyPlot is a cross-platform .NET chart library. The project is open-source and under an MIT license. The GitHub project has a very active community with many pull requests and daily commits [26].

### 5.5.4 Conclusion

While all three libraries have a similar set of functions it looks like Microcharts is mainly focused on Xamarin and relying on the WPF compatibility could pose a risk. Even though the OxyPlot project has a smaller community it was way more active in the last year and therefore looks more promising for future developments and support. For this reason, we choose the OxyPlot library to visualise our measurement data.

## 6 Concept

This chapter contains the concept that was developed based on the requirements and use cases. It is mainly focussed on the parts that are implemented in the proof of concept, but possible future components are also described.

Components that already exist are described in detail in this chapter because they do not belong to the next chapter “7 Implementation”. For components which we have created, this chapter only contains the conceptual aspects and the implementation details follow in the next chapter.

### 6.1 Overview

The centre of the architecture is ChirpNest which runs on the LoRa stack unit. ChirpNest includes the whole LoRa stack which is provided by ChirpStack. LoRa packets from LoRa devices (ADT1 in the proof of concept) are received and stored in the PostgreSQL database by ChirpNest. Clients then are able to access the API of KIWI Server within ChirpNest to retrieve this data.

Figure 16 shows an overview over the components of the concept. The data flow is also indicated by arrows.

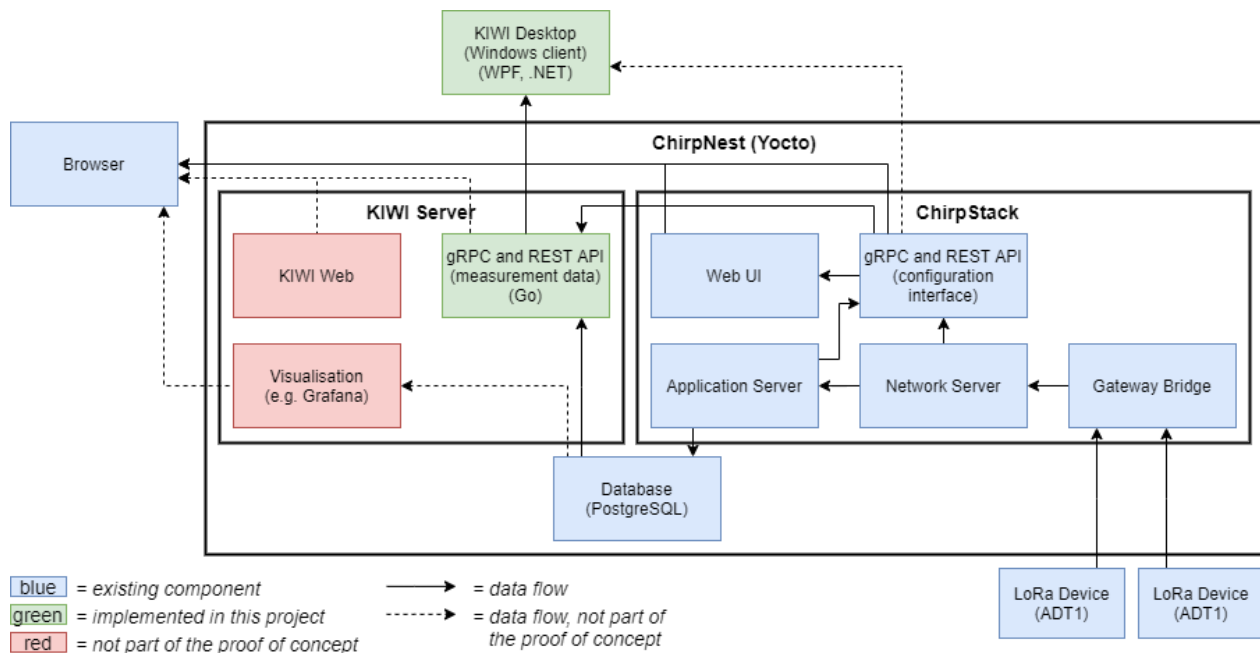


Figure 16: Concept overview

Figure 17 illustrates the path that measurements take from the ADT1 to the computer of the user.

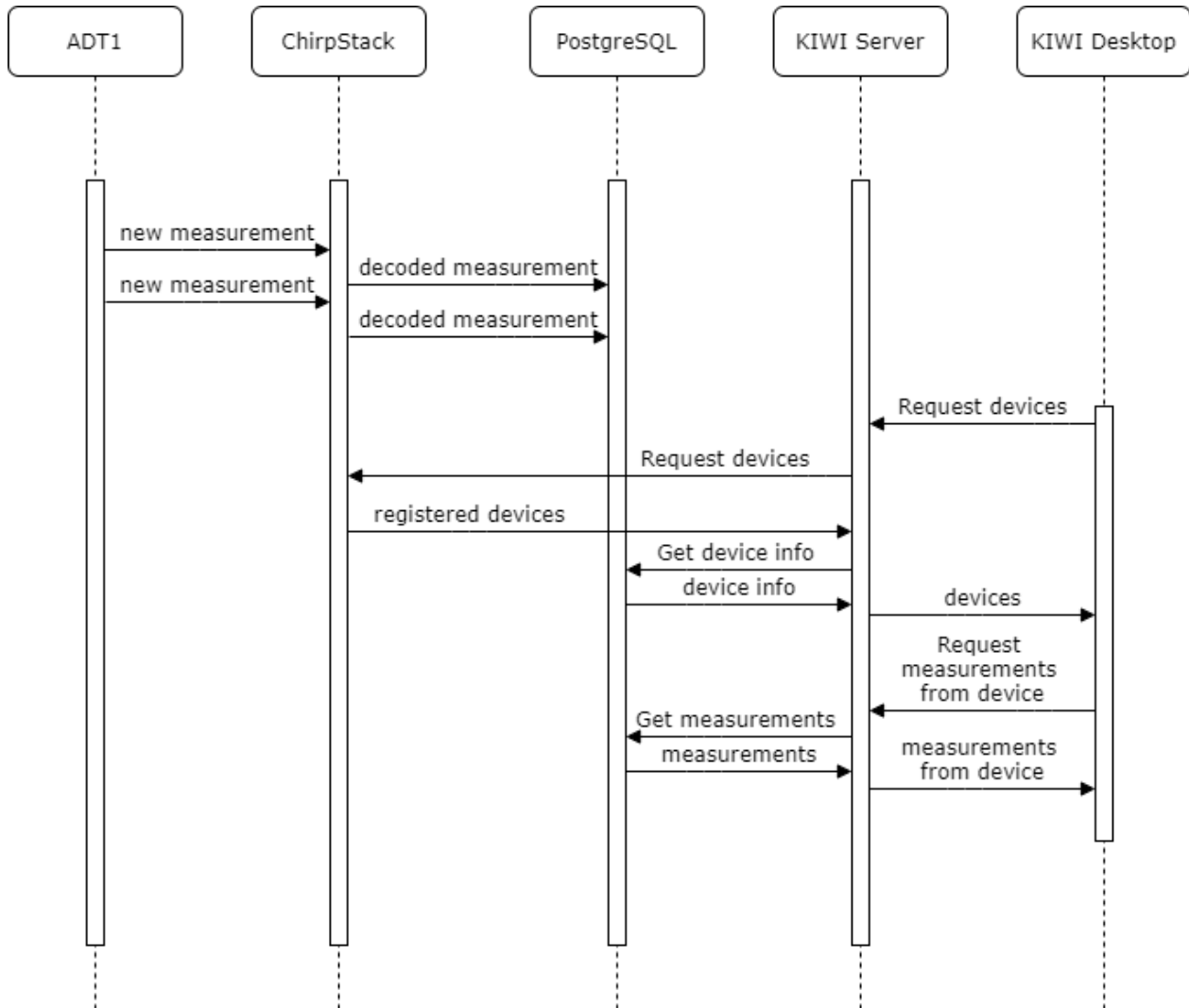


Figure 17: Measurements sequence diagram

## 6.2 ChirpNest

The concept ChirpNest consisting of ChirpStack with data storage and KIWI Server is described in this section.

### 6.2.1 ChirpStack with Data Storage

ChirpStack provides all required LoRa components starting with the gateway bridge, going on with the network server and also the application server [27]. The application server offers a gRPC API and also a REST API (swagger UI is hosted) having exactly the same functionality. These APIs allow the configuration of the application server [28]. The web interface, which is also hosted by the application server, offers an easy to use interface for interacting with the REST API. In our scenario, an application has to be configured on the ChirpStack Application Server and also a

device (including device profile) which has to be activated as well. Authentication with username and password is required to interact with the API or web interface.

#### 6.2.1.1 Device Setup

Table 9 lists the steps that are required to set up a device on a newly created ChirpStack instance. These steps can also be carried out using the web interface of ChirpStack which then handles creating the listed API requests. The steps when using said web interface are explained in the manual “VII.I. Manual: Installation of a LORIX One with an ADT1 Device” below “Configure ChirpNest through the ChirpStack Web Interface” in the appendix on page 114.

Action	REST API method	Description
login	/api/internal/login	login using username and password, receiving a json web token (JWT which is required for all the following calls
add network server	/api/network-servers	add the network server which is available under “localhost:8000”
add service profile	/api/service-profiles	add a service profile to the network server
add device profile	/api/device-profiles	add device profile suitable to the ADT1, also containing the decode function (see “6.2.1.3 KELLER Decoder function”)
add application	/api/applications	add an application referencing the service profile
add device	/api/devices	add a device to an application
activate device	/api/devices/{dev_eui}/activate	activate the device with the according key

Table 9: Steps required to setup a device on a new ChirpStack instance

#### 6.2.1.2 PostgreSQL Integration for Data Storage

The PostgreSQL integration of the ChirpStack Application Server can be activated globally by modifying the single configuration file “chirpstack-application-server.toml” [19]. It being activated globally means that it is automatically enabled for every application that is configured in the ChirpStack application server.

For the PostgreSQL integration to work, the database tables (“device\_up”, “device\_status”, “device\_join”, “device\_ack”, “device\_error”, “device\_location”) have to be created with the correct schema as laid out in [29]. Also, the extension “hstore” needs to be activated.

The only relevant table in this project is “device\_up” which stores uplink data and therefore measurements. The fields of the “device\_up” table are listed in Table 10.

Column	Type	Relevance
id	uuid	-
received_at	timestamp with time zone	timestamp of the measurement
dev_eui	bytea	identifier of the device
device_name	character varying(100)	-
application_id	bigint	-
application_name	character varying(100)	-
frequency	bigint	-
dr	smallint	-
adr	boolean	-
f_cnt	bigint	-
f_port	smallint	-
tags	hstore	-
data	bytea	-
rx_info	jsonb	-
object	jsonb	contains the decoded fields

Table 10: PostgreSQL table schema of “device\_up”

### 6.2.1.3 KELLER Decoder function

The payload of received LoRa packet mostly containing measurements is decoded by ChirpStack in our project. To achieve this, ChirpStack allows to configure “Custom JavaScript codec functions” where JavaScript (specifically ES5) code can be run to extract the data from the payload and fill it into fields by creating a JavaScript object [30]. The decode function is provided by KELLER [31]. It is intended for The Things Network but it also works on ChirpStack when the entry function signature is changed from `Decoder(bytes, port)` (The Things Network style) to `Decode(port, bytes)` (ChirpStack style). Listing 1 shows an excerpt from the decoder JavaScript code as provided by KELLER with the changed entry function.

```
function Decode(port, bytes) {
  var functionCode = bytes[0];
  var result = {
    func: functionCode,
    port: port,
    payload: bytes.map(function (byte) { return pad(byte.toString(16).toUpperCase(), 2); }).join('')
  };

  if (functionCode === 12) {
    // Device sends information package, has to be decoded differently
    fillupDecodedDeviceInformation(bytes, result);
  } else {
    // Device sends measurement package
    fillupDecodedMeasurements(bytes, result);
  }

  return result;
}
```

Listing 1: Excerpt from the decoder JavaScript code [31]



The result is ultimately written into the “object” column of the uplink table “device\_up” in JSON format. For illustration, Listing 2 shows an example of a decoded measurement uplink packet and Listing 3 shows an example of a decoded device information uplink packet. Both were sent by an ADT1, decoded with said decoder function and the examples were obtained from the “object” column in the uplink table (“device\_up”).

```
{
  "P1": 0.9503183364868164,
  "ct": 3,
  "TOB1": 22.42431640625,
  "func": 1,
  "port": 1,
  "channel": "0000000000010010",
  "payload": "010300123F73481041B36500",
  "channelCount": 2
}
```

*Listing 2: Example of a decoded measurement uplink packet*

```
{
  "func": 12,
  "port": 4,
  "payload": "0C011300132F0000005C266B9C16409EA0D1502B",
  "serial_number": 92,
  "battery_voltage": 4.9571309089660645,
  "sw_version_text": "19.47",
  "class_group_text": "19.00",
  "humidity_percentage": 43,
  "device_local_datetime": "2020-06-04 11:48:38",
  "battery_capacity_percentage": 80
}
```

*Listing 3: Example of a decoded device information uplink packet*

The resulting fields are described in the following tables. Table 11 lists the fields that are provided in both decoded uplink and device information packets. Table 12 lists the fields that are only provided only in decoded uplink packets and Table 13 the fields that are only provided in device information packets.

Field Name	Meaning, details
“port” ( <i>int</i> )	Port of the LoRa packet provides the following ports: <ul style="list-style-type: none"> <li>• 1: Measurements</li> <li>• 2: Alarm</li> <li>• 3 Configuration</li> <li>• 4 Info (Battery voltage, Humidity, Time ...)</li> <li>• 5 Answer on a request</li> </ul>
“func” ( <i>int</i> )	Function code, retrieved from the first byte of the payload: <ul style="list-style-type: none"> <li>• 1: Measured values in float format</li> <li>• 12: “Info” message</li> <li>• 31: 1 Byte variable, 32: 1 Byte variable - stream</li> <li>• 51: 4 Byte variable, 52: 4 Byte variable - stream</li> <li>• 61: Float variable, 62: Float variable – stream</li> <li>• 71: ASCII characters, 72: ASCII characters – stream</li> <li>• 81: KELLER Sensor information</li> <li>• 90: Command / Configuration</li> </ul> Only function code 1 and 12 are relevant in this project.
“payload” ( <i>string</i> )	Payload of the LoRa packet as a hexadecimal string.

Table 11: General fields resulting from the decoder function [32]

Field Name	Meaning, details
“ct” ( <i>int</i> )	Connection type / device type, retrieved from the 2 <sup>nd</sup> byte of the payload: Contains a value ranging from 0 to 13 and determines which channels are available (see variable “deviceTypesToChannelNames” in [31] and see “Device Type Overview” in [32])
“channel” ( <i>string</i> )	“The ‘Channel’ shows you how many and which channels are transmitted in the message. Each bit stands for one channel, thus channel 1 ... 16 are selectable”. [32] “Example: Channel: 0000'0000 1000'0101 = Values 1 + 3 + 7 are contained in the message. The least significant bit of the channel is always the first value”. [32] The field contains the value as a binary string.
“channelCount” ( <i>int</i> )	Number of transmitted channels (corresponds to the number of “1” characters in the channel field)
Channel name ( <i>float</i> ) e.g.: <ul style="list-style-type: none"> <li>• “P1” (<i>float</i>)</li> <li>• “TOB1” (<i>float</i>)</li> </ul>	Value of the channels in float IEEE 754 format. The channel names are determined by “ct” and “channel” (deviceTypesToChannelNames contains a name for every possible channel for every device type). The example in Listing 2 contains “P1” (1 <sup>st</sup> pressure channel) and “TOB1” (1 <sup>st</sup> temperature channel).

Table 12: Measurement packet fields resulting from the decoder function [32]

Field Name	Meaning
"battery_voltage" ( <i>float</i> )	Voltage of the battery in volt
"battery_capacity_percentage" ( <i>int</i> )	Calculated battery capacity in percent
"humidity_percentage" ( <i>int</i> )	Relative humidity in percent
"class_group_text" ( <i>string</i> )	KELLER class and group of the device
"sw_version_text" ( <i>string</i> )	Year and week of the software version on the device
"serial_number" ( <i>int</i> )	Serial number
"device_local_datetime" ( <i>string</i> )	Local time (format "year-month-day hour-minute-second")

Table 13: Device information packet fields resulting from the decoder function [32]

## 6.2.2 KIWI Server

KIWI Server is the application within ChirpNest that is responsible for providing the required data for KIWI Desktop and KIWI Web to function. This includes measurement data and device information which is provided via the API which is explained in section "6.2.2.1 API". KIWI Server also has to serve the static files for KIWI Web to run in browsers (see section "6.2.2.2 KIWI Web").

KIWI Server is implemented using Go as evaluated in "5.4 API Technology".

### 6.2.2.1 API

The API of KIWI Server ("KIWI Server API") offers these three functionalities:

- Provide measurement data
- Enable measurement data to be deleted
- Provide device information

Note that in order to fulfil the specification of KIWI Web (see "4.3.3 KIWI Web"), the KIWI Server API needs to offer many more functionalities (e.g. adding/removing devices, alarms). These however are not dealt with in this section and only the functionality which is required for the proof of concept is described in the concept of the API.

The measurement data is directly retrieved from the local PostgreSQL database from the above mentioned "device\_up" table (see 6.2.1.2 PostgreSQL Integration for Data Storage). Information about the configured devices is retrieved from the ChirpStack application server via its gRPC API and also from the PostgreSQL when a device information uplink packet containing fields as described in Table 13 has been sent by the device.

The data is provided by a gRPC interface using protocol buffers as evaluated in "5.4 API Technology". In addition, the same functionality is also provided by a RESTful JSON interface. Offering these interfaces is inspired by and analogous to the API of the ChirpStack application server [28].

Figure 18 illustrates the described concept of the KIWI Server API.

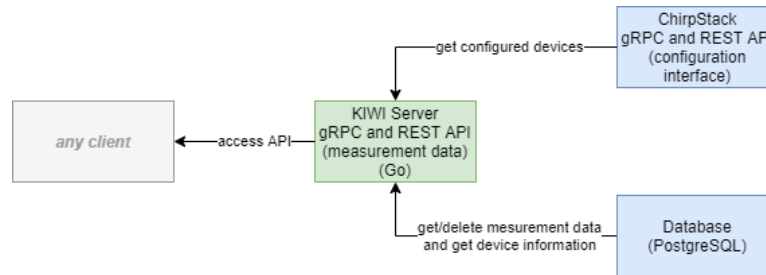


Figure 18: Concept illustration of the KIWI Server API

#### 6.2.2.2 KIWI Web

Since KIWI Web is not part of the proof of concept, the concept of it is limited to only stating the role and one possible technology that could be used.

KIWI Web is the front-end web interface provided by KIWI Server and therefore by ChirpNest. It provides easy to use access to the functionality of the KIWI Server API. The functionality is described in previous chapters (see “4.3 Specification” and “4.4 Use Cases”).

Since the KIWI Server API uses the same technology as the ChirpStack Application Server API (see “5.4 API Technology”), a possible choice for the technology of KIWI Web would be to use React as the ChirpStack Application Server front-end uses React [33].

#### 6.2.2.3 Visualisation

The idea of the “Visualisation” component (see Figure 16 on page 53) was to take an existing, freely usable software (e.g. Grafana) that visualises the measurements in browsers requiring little configuration effort and is hosted by KIWI Server within ChirpNest. However, due to the RAM of the LORIX One being only 128 MB [14] and the hardware recommendations for Grafana being at least 255 MB in the Grafana documentation [34], a detailed technical analysis was omitted and the idea was not pursued further.

### 6.2.3 Scalability

If the area where the LoRa end nodes are located is too large for a single LoRa gateway, the option to set up multiple gateways with ChirpNest is an easy way to extend the area with LoRa coverage. Each KELLER remote transmission unit then has to be registered to the nearest ChirpNest and the user then gathers the data from each gateway individually.

If multiple gateways are in the same network its even simpler. The user only has to set up one gateway with ChirpNest and further gateways only have to redirect the message to the ChirpNest. Since ChirpNest contains a network server this can be done with any gateway the same way as in a regular LoRa setup.

## 6.3 KIWI Desktop

KIWI Desktop is a windows application to connect to ChirpNest, download measurements and visualize and export them.

### 6.3.1 Design

This section contains the mockups of the core functionalities.

#### Overview

The overview page in Figure 19 will show a list of the local measurements that can be selected to display in a chart. There is also a textbox to connect to the KIWI Server running on a LORIX One in the same network.

KIWI Desktop

Back Overview

IP Address

Connect

**Local measurements**

SHOW	EUI-1234	01.01.2020 08:40 - 02.05.2020 19:30
SHOW	EUI-1122	02.02.2020 20:20 - 03.03.2020 03:03
SHOW	EUI-3456	02.02.2020 20:20 - 04.04.2020 04:04

Figure 19: Mockup overview unconnected

## Overview connected

When the connection to the KIWI server could be established, the list of the KELLER remote transmission units shows up with the option to show the measurements or directly download them to the computer as shown in Figure 20.

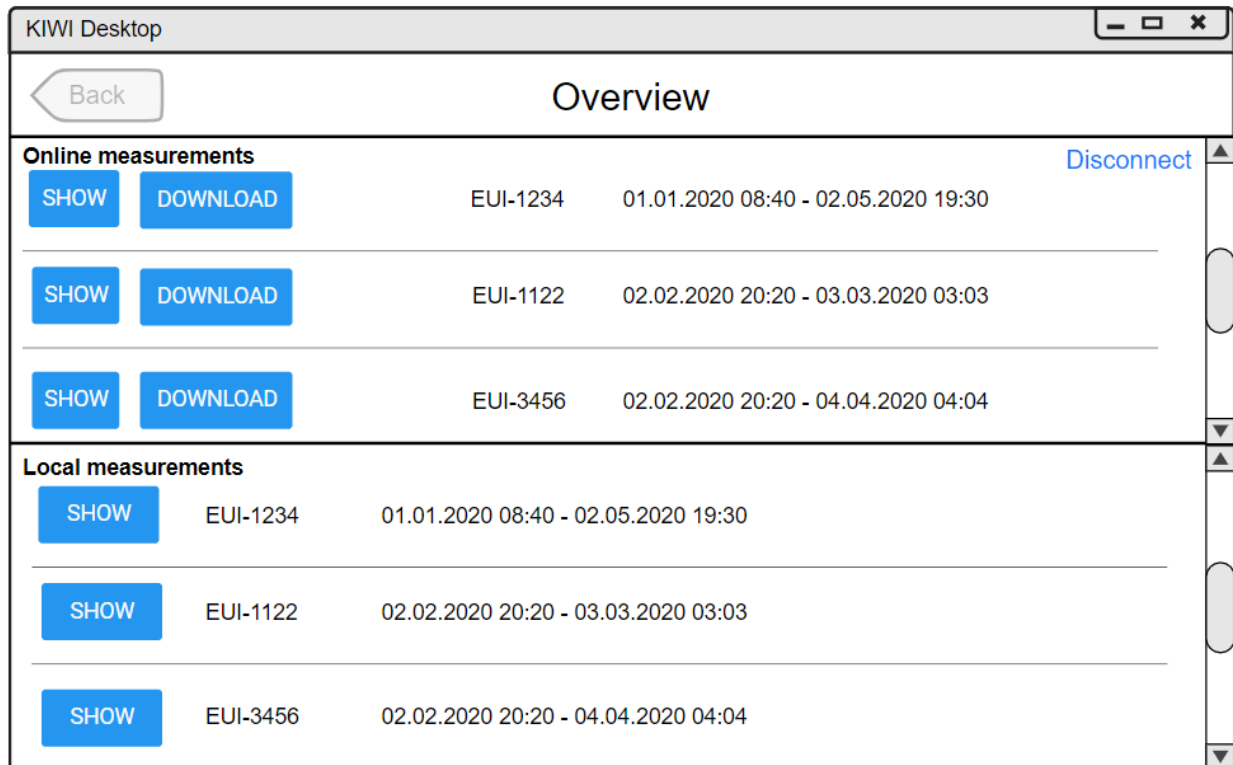


Figure 20: Mockup overview connected

## Measurement online

When a device is selected the measurements shows up in a chart with the options to download, export and delete them. Figure 21 pictures how that will look like.

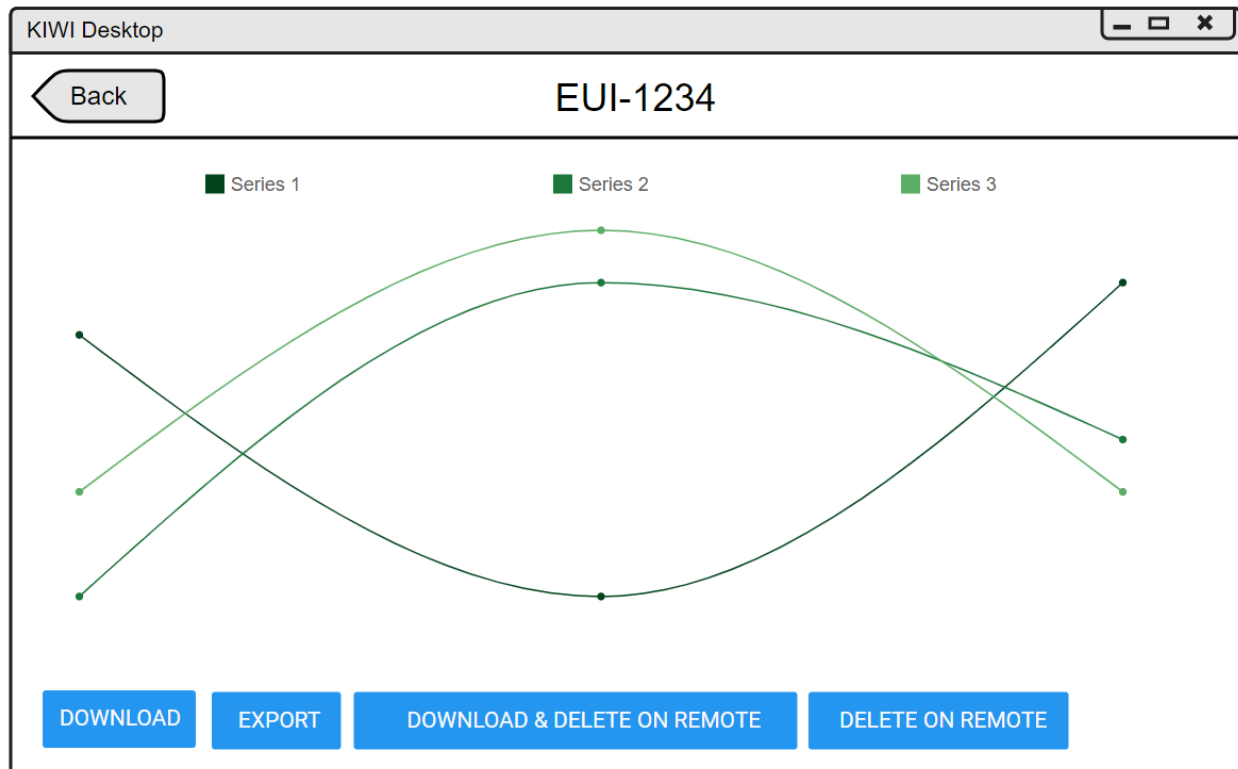


Figure 21: Mockup measurement online

## Measurement local

Figure 22 shows a local measurement that is selected. The measurement is visualised in a chart with the options to export and delete the data.

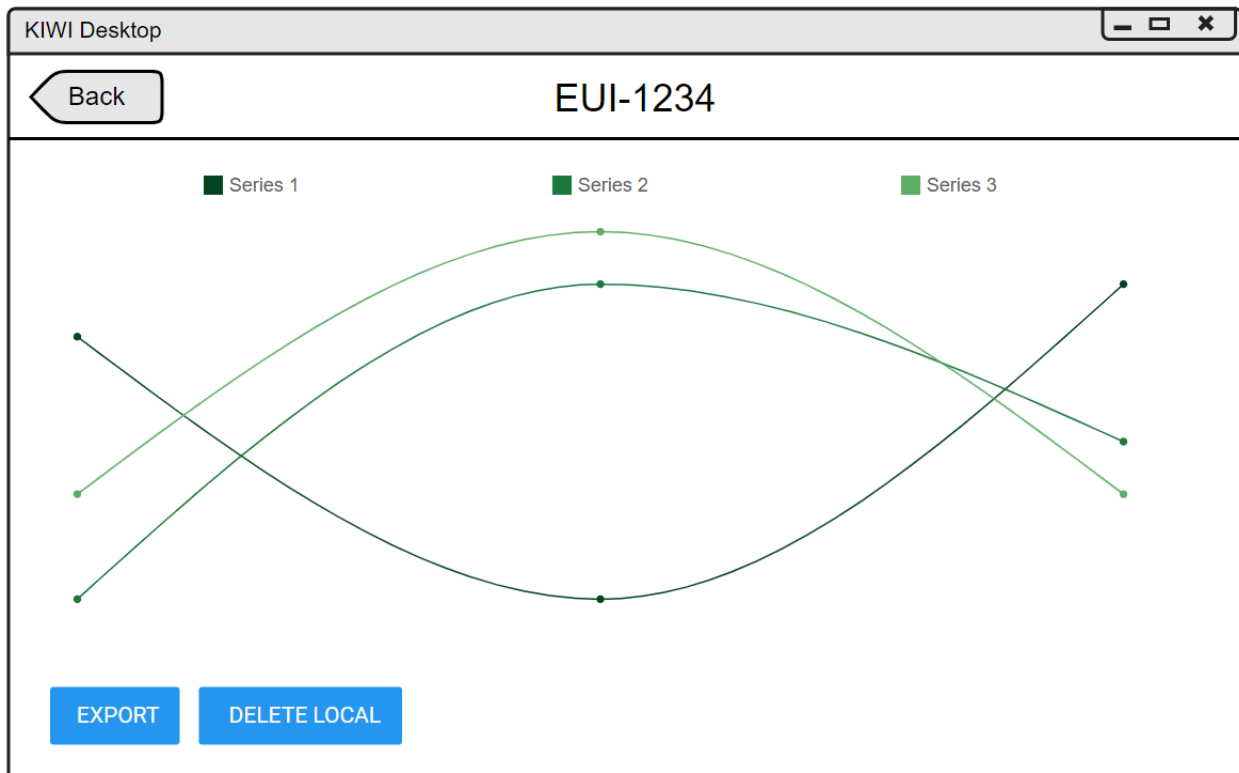


Figure 22: Mockup measurement local



## Download confirmation

When downloading a measurement, the option in Figure 23 is given to combine the measurement with existing local measurements of the same device into a single measurement.

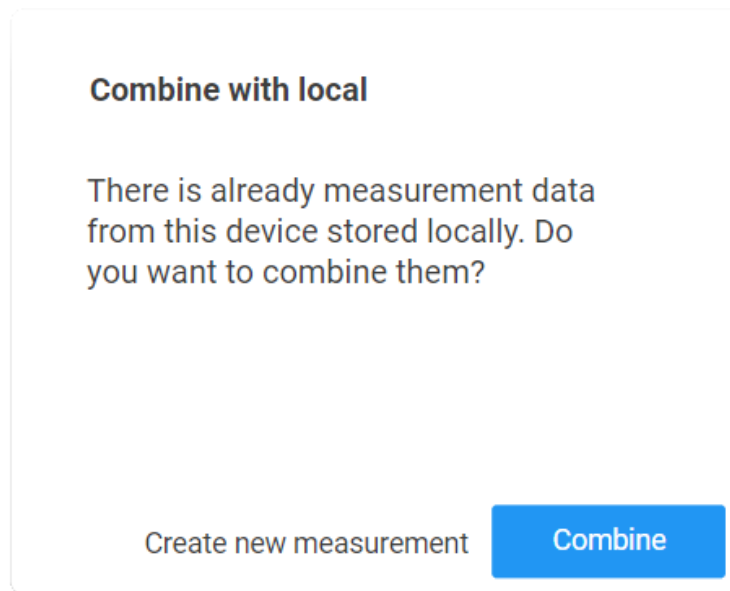


Figure 23: Mockup combine measurements

The state diagram in Figure 24 visualizes the interaction of the user interfaces.

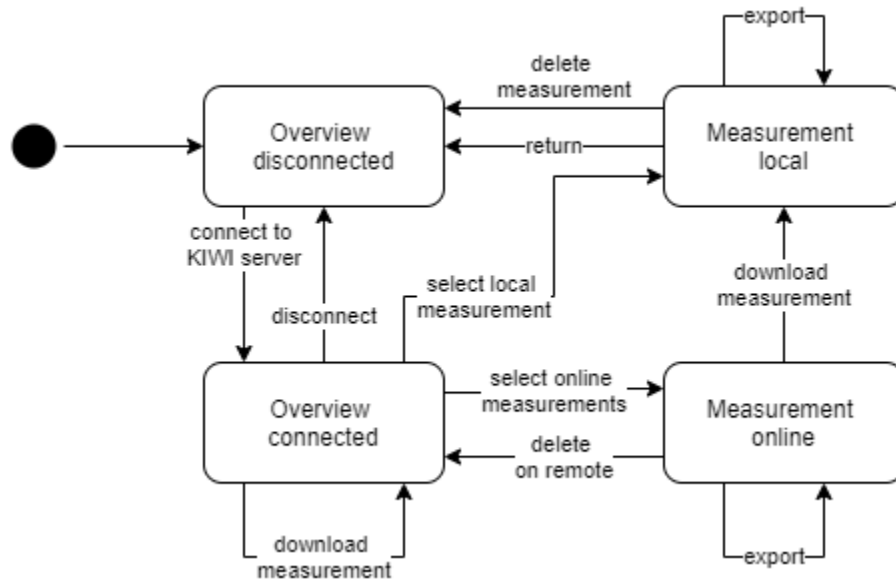


Figure 24: KIWI Desktop state diagram

### 6.3.2 Data Storage

KIWI Desktop stores measurement data in JSON files. The JSON structure is a KELLER standard for measurement data which is shared between all KOLIBRI products.

Storing the measurements in this format offers several benefits:

- The format is known by KELLER and therefore simplifies further development by KELLER.
- Easy exchange of data between KOLIBRI products and KIWI Desktop on a file basis.
- Does not require the installation of a database.
- Existing functionalities of KELLER code could be used with the KELLER format, e.g. export to CSV/Excel.

### 6.3.3 Export

To export the KOLIBRI JSON format KELLER provides a library that generates the required Excel and CSV files. This library uses Syncfusion which requires licensing [35]. This is why we do not use the export library of KELLER but implement the functionality with the open-source library ClosedXML that helps with the creation of Excel sheets [36]. This will give KELLER the option to distribute this KIWI Desktop without the need of Syncfusion licensing while leaving the option to replace the export functionalities with their own library open.

## 7 Implementation

This chapter explains the details of the implementation. Since used existing components are already explained in detail in the previous chapter (“6 Concept”), this chapter is focused on the components that were implemented by us, namely the KIWI Server API and KIWI Desktop.

### 7.1 System Setup

In Figure 25, the setup of the running system is pictured. We see the LORIX One connected to a switch and power. A Laptop is connected to the same switch and therefore in the same local network. On the laptop KIWI Desktop is running and connected to the ChirpNest running on the LORIX One. On the right side is the ADT1 with the pressure sensor in the water continuously sending the pressure of the water height.

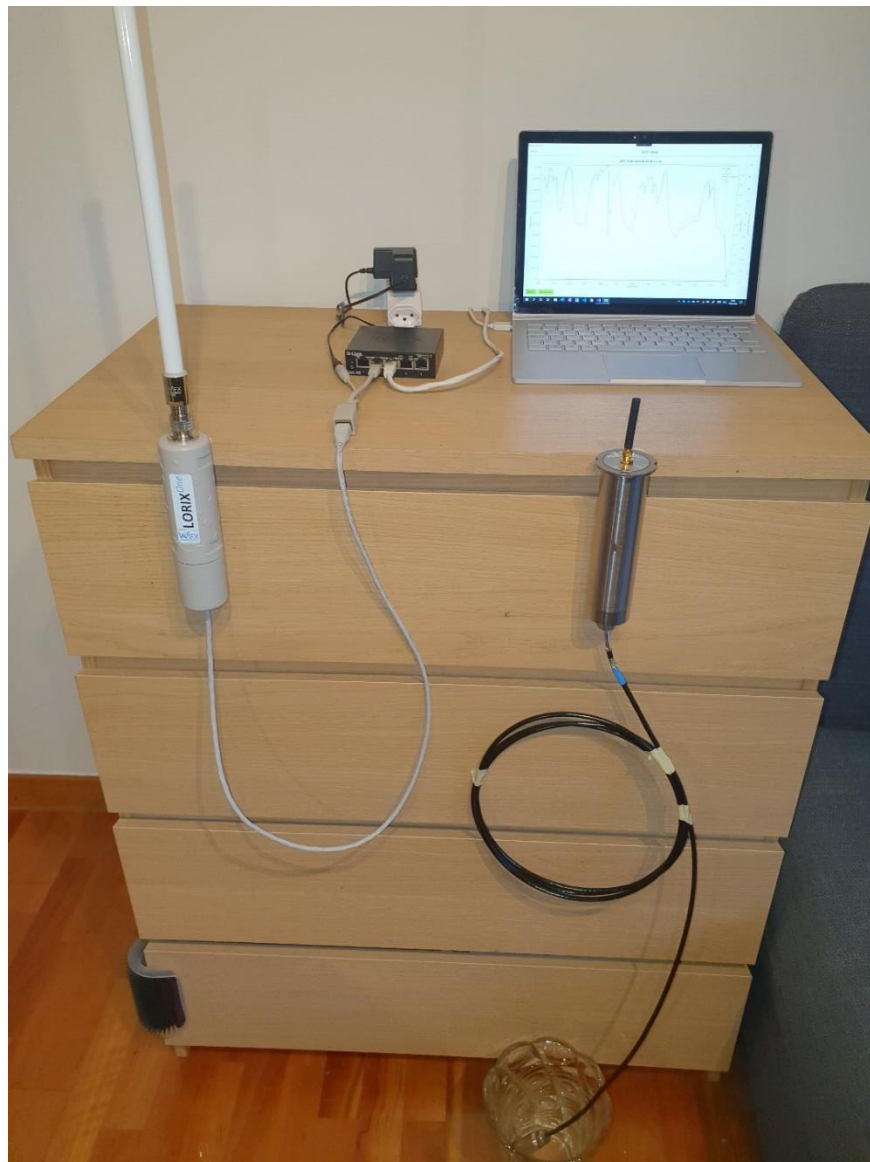


Figure 25: System setup

## 7.2 ChirpNest

ChirpNest includes all ChirpStack components which cover the whole LoRaWAN functionality. These ChirpStack components are preconfigured as described in “7.2.1.1 ChirpStack Configuration within the ChirpNest Image”. ChirpNest also includes KIWI Server which hosts an API that is described in “7.2.2 KIWI Server API”. Said components are packed into an image that boots on the LORIX One (see “7.2.1 ChirpNest Image”).

Note that ChirpNest is not yet ready to be published because currently, the described changes are just added to the ChirpStack Gateway OS repository into the existing image “chirpstack-gateway-os-full.bb”. A sensible implementation that also enables easier maintainability (update to newer ChirpStack version trouble-free) is to strictly separate all additions from the existing components by creating a new image next to chirpstack-gateway-os-full but which includes and extends chirpstack-gateway-os-full. Also, the handling of licenses in “kiwi-server\_BETA.bb” should be ensured to be correct. For this reason, the text that appears when a user logs in using SSH writes “ChirpNest BETA version”. This text was added to the file called “motd”.

### 7.2.1 ChirpNest Image

ChirpNest is built using the ChirpStack Gateway OS. ChirpStack Gateway OS is an embedded Linux system that is open-source based and built with Yocto [16]. The code to build ChirpStack Gateway OS is open-source and under the MIT license and can therefore be used as a basis for ChirpNest.

#### 7.2.1.1 ChirpStack Configuration within the ChirpNest Image

The file “chirpstack-application-server.toml” is changed by adding these configurations:

- The PostgreSQL integration was activated (alongside with the MQTT integration which is activated by default). Also, the connection to the local data source is provided in the `dsn` option.

```
[application_server.integration]
enabled=["mqtt", "postgresql"]
```

```
[application_server.integration.postgresql]
dsn="postgres://chirpstack_as_events:dbpassword@localhost/chirpstack_as_events?sslmode=disable"
```

- The maximum execution time for the decoder in the ChirpStack Application Server is set to 200 ms. The reason for this configuration is because with the default setting, the received uplink packets triggered an error when ChirpStack Application Server decoded them using the decoder function. This way, the packets were not decoded and hence the information of the packets was not properly written into the PostgreSQL database.

```
[application_server.codec.js]
max_execution_time="200ms"
```

#### 7.2.1.2 Set up PostgreSQL Database for ChirpStack Integration

The PostgreSQL integration of ChirpStack requires a database with a certain schema to be available as stated in [29]. This is where the table “device\_up” is created amongst other tables. The creation of this database is executed automatically during the first boot of ChirpNest. For this purpose, the following files were modified or created:

- firstbootinit\_1.2.0.bb
- firstbootinit.sh
- setup\_tables.sql

#### 7.2.1.3 Integration of KIWI Server

The ChirpStack components are added to the ChirpStack Gateway OS and therefore also to ChirpNest. These components are configured to automatically be started on after booting and to be monitored with Monit [37]. KIWI Server is added to and monitored by ChirpNest exactly the same way.

The following files were created for that purpose:

- kiwi-server\_BETA.bb
- kiwi-server.init
- kiwi-server.monit
- kiwi-server.toml

Also, the file “chirpstack-gateway-os-full.bb” needed to be changed so KIWI Server is included in the image creation process.

#### 7.2.1.4 Image Build Configuration

The repository was also modified so that the settings are correct for the LORIX One when building an image is triggered. For this purpose, `VENDOR = "wifx"` was written into the file “bblayers.conf.sample” and `MACHINE = "lorix-one-512-sd"` was written into the file “local.conf.sample”

### 7.2.2 KIWI Server API

The KIWI Server API offers two types of APIs:

- gRPC interface
- JSON REST API

Both APIs are listening on port 8081 and not require authentication and there is no encryption involved.

#### 7.2.2.1 gRPC interface

The gRPC interface is defined in “.proto” files using the protocol buffer language, specifically the proto3 syntax [38]. Listing 4 shows an excerpt from the “measurement.proto” file and defines the “Get Measurements” API Endpoint (see “7.2.2.3 API Endpoints”).

```

// MeasurementService is for managing and retrieving the measurements.
service MeasurementService {
    // Get returns the requested measurements.
    rpc Get (GetMeasurementsRequest) returns (GetMeasurementsResponse) {
        option(google.api.http) = {
            get: "/api/measurements/{dev_eui}"
        };
    };
    [...]
}
[...]
message GetMeasurementsRequest {
    // Device EUI (HEX encoded).
    string dev_eui = 1 [json_name = "devEUI"];

    // Get only measurements after this timestamp.
    google.protobuf.Timestamp start = 2;

    // Get only measurements before this timestamp.
    google.protobuf.Timestamp end = 3;
}

message GetMeasurementsResponse {
    // Total number of measurements available within the result-set.
    int64 number_of_measurements = 1 [json_name = "numberOfMeasurements"];

    // Device EUI (HEX encoded).
    string dev_eui = 2 [json_name = "devEUI"];

    // All devices.
    repeated MeasurementListItem measurements = 3;
}

message MeasurementListItem {
    // Time of the measurement.
    google.protobuf.Timestamp time = 1;

    // Port.
    int64 port = 2;

    // Channel (in last 2 bytes, e.g. 00000000 00000000 10100000 00010010).
    fixed32 channel = 3;

    // Number of channels.
    int64 channel_count = 4 [json_name = "channelCount"];

    // ct (as decoded)
    int64 ct = 5;

    // func (as decoded)
    int64 func = 6;

    // Measured value of each channel.
    map<string, double> channel_values = 7 [json_name = "channelValues"];
}

```

*Listing 4: Excerpt from the “measurement.proto” file*

This definition of the gRPC interface in the proto3 syntax is used to automatically generate server code as explained in “VII.II. Manual: Development of KIWI Server” below “Generate API Code from .proto Files” in the appendix on page 121.

The .proto files are also used to generate the C# data access classes in KIWI Desktop (see “7.3 KIWI Desktop”).

### 7.2.2.2 JSON REST API

The JSON REST API is just a gateway forwarding the requests to the gRPC interface. For that reason, the two APIs offer exactly the same functionality and the same fields.

Note that the variable names in the response model of the JSON REST API are in camel case and without underscores (“\_”), e.g. `number_of_devices` becomes `numberOfDevices` and `channel_count` becomes `channelCount`. Also, in the JSON REST API, the type `uint` is the same as `int` and the type `timestamp` is a string representing UTC date and time with the “Z” zone designator, e.g. “2020-06-04T15:14:03.242229Z”. All timestamps are UTC, also on the gRPC interface.

### 7.2.2.3 API Endpoints

The three implemented API endpoints of KIWI Server are shown below.

#### Get Devices

This endpoint delivers a list of all devices that are configured in ChirpNest. The field `device_info_available` is true when the information for the fields `serial_number` and `device_type` could be found in an information packet.

Method	GET
REST endpoint	/api/devices
gRPC service, method	DeviceService, List
Request parameters	–
Response model	<pre>int number_of_devices repeated devices [   string dev_eui   string name   string description   int serial_number   string device_type   bool device_info_available   timestamp first_measurement_time   timestamp last_measurement_time   int number_of_measurements ]</pre>

## Get Measurements

This endpoint delivers the measurements of one device. The time frame can be limited by specifying the parameters `start` and/or `end`.

Method	GET
REST endpoint	/api/measurements/{dev_eui}
gRPC service, method	MeasurementService, Get
Request parameters	string dev_eui ( <i>part of the path in JSON</i> ) timestamp start <i>optional</i> timestamp end <i>optional</i>
Response model	<pre>int number_of_measurements string dev_eui repeated measurements [   timestamp time   int port   uint channel   int channel_count   int ct   int func   map&lt;string, double&gt; channel_values ]</pre>

## Delete Measurements

This endpoint allows measurements of one device to be deleted. The time frame can be limited by specifying the parameters `start` and/or `end`.

Method	DELETE
REST endpoint	/api/measurements/{dev_eui}
gRPC service, method	MeasurementService, Delete
Request parameters	string dev_eui ( <i>part of the path in JSON</i> ) timestamp start ( <i>optional</i> ) timestamp end ( <i>optional</i> )
Response model	—



#### 7.2.2.4 Data Access

The KIWI Server API delivers measurement data and device data (see “7.2.2.3 API Endpoints”) and allows deleting measurements. The access of said data is described below.

##### Measurement Data

KIWI Server retrieves the measurement data from the “device\_up” table of the database that is integrated by ChirpStack (see “6.2.1.2 PostgreSQL Integration for Data Storage”). The database is accessed using the Go package “database/sql” [39]. The PostgreSQL statements are in the file “db\_queries.go” and parameters such as a Device EUI are included in the statement using the functionality of said “database/sql” package.

The SQL statements that are used for retrieving and deleting measurements are KELLER specific. The query for retrieving measurements of a device for example filters for rows in the “device\_up” table that have “1” in the “f\_port” column and that have a “func” property with the value “1” in the JSON object of the “object” column (“measurement packets” as described in “6.2.1.3 KELLER Decoder function”). The following excerpt from the SQL statement does the described filtering: `WHERE f_port = 1 AND object @> '{"func": 1}'`

##### Device Data

The device data is partially retrieved from ChirpStack Application Server API and partially also from the “device\_up” table of the database.

The ChirpStack Application Server API is called using the package imported with `"github.com/brocaar/chirpstack-api/go/as/external/api"` and documented on [40]. The authentication is performed by calling the “Login” method of the “InternalService” service with hard coded username and password to obtain the required JWT. The device list is retrieved by calling the “List” method of the “DeviceService” (the previously obtained JWT is added to the request to prove authentication). This returns a list of devices including device EUI, name and description.

The rest of the device data is retrieved from the “device\_up” table which is accessed the same way as for measurement data. The table is filtered for the newest device information packet (see “6.2.1.3 KELLER Decoder function”) by the following excerpt from the SQL statement: `WHERE [...] AND object @> '{"func": 12}'`. Further details are retrieved by querying the “device\_up” table for the timestamp (column “received\_at”) of the oldest and newest measurement packet and for the number of available measurement packets for the corresponding device.

## 7.3 KIWI Desktop

### 7.3.1 Project Structure

The KIWI Desktop project consists of three .Net projects shown in Figure 26.

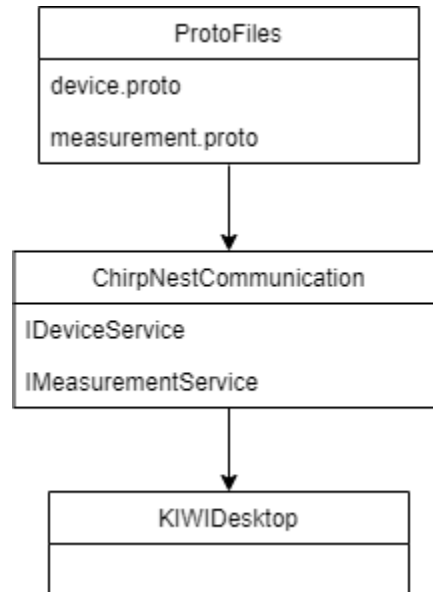


Figure 26: KIWI Desktop project structure

#### ProtoFiles

ProtoFiles is a .Net Standard 2.0 class library that contains the .proto files from the ChirpNest API. By using the official gRPC NuGet packages this class library automatically generates C# classes representing the data structure defined in the .proto files. These classes are used by the ChirpNestCommunication project.

#### ChirpNestCommunication

ChirpNestCommunication is a .Net Standard 2.0 project that provides Interfaces and functions to fetch data from the ChirpNest AP. The fetched measurements then are mapped to the KELLER classes. This functionality is separated from the KIWI Desktop project so that the project could be turned into a NuGet package by KELLER and the communication with ChirpNest could be included into other KELLER software or third-party software.

#### KIWIDesktop

KIWIDesktop is a WPF project with the target .Net Framework 4.7.2. This project contains the views and the view models.

### 7.3.2 NuGet Packages

We used several NuGet packages in KIWI Desktop. In Table 14: NuGet packages used in KIWI Desktop the used packages including their licenses are described.

<b>Package</b>	<b>Author</b>	<b>Version</b>	<b>Licence</b>
AutoMapper	Jimmy Bogard	9.0.0	MIT
ClosedXML	Francois Botha, Alekssei Pankratev, Manuel de Leon, Amir Ghezlbash	0.95.3	MIT
CommonServiceLocator	Microsoft	2.0.5	MS-PL
Grpc	Grpc	2.28.1	Apache-2.0
MaterialDesignThemes	James Willock	2.6.0	MIT
MVVMLight	GalaSoft	5.4.1.1	MIT
Newtonsoft.Json	James Newton-King	12.0.3	MIT
NLog	Jarek Kowalski, Kim Christensen, Julian Verdurmen	4.7.2	BSD 3
NodaTime	Jon Skeet	2.4.7	Apache-2.0
OxyPlot	OxyPlot	2.0.0	MIT
KellerAg.Shared.Entities	KELLER AG für Druckmesstechnik	3.2.150	unknown
KellerAg.Shared.Export	KELLER AG für Druckmesstechnik	3.2.150	unknown

*Table 14: NuGet packages used in KIWI Desktop*

### 7.3.3 Testing

In this proof of concept, we focused on the ChirpNestCommunication project for testing. With Unit-tests the mapping from the API object data to the KELLER measurement format is ensured to work as intended.

## 8 Validation and Verification

In this chapter, the validation and verification of the proof of concept system is documented to ensure it meets the specification. This includes a validation with KELLER as well as a separate verification of each specification.

### 8.1 Validation with KELLER

The validation with KELLER was conducted on the 9th of June 2020 at the KELLER headquarter in Winterthur. ChirpNest was set up on a LORIX One and two ADT1 devices were configured. A laptop with KIWI Desktop was in the same local network as the LORIX One and visualized the measurements. After a presentation for interested KELLER employees of the development department, the actual validation was conducted with Sebastian Mojado representing KELLER. We provided an overview over the components and their dependencies, answered all questions and went through the specification. The protocol of the validation is in the appendix (see “IV. Protocol: Verification at KELLER”).

### 8.2 System Test

To verify the majority of the specifications (see next section “8.3 Verification: Check Specification”), we conducted a system test. The system test is very similar to what was set up at during the validation with KELLER, but it is documented including with screenshots. The procedure is described step by step in the following paragraphs.

We tested setting up a LORIX One running ChirpNest, configured an ADT1 device and connected KIWI Desktop running on a laptop in the same network to ChirpNest. The setup was done by providing the requirements and following the steps from “VII.I. Manual: Installation of a LORIX One with an ADT1 Device” in the appendix. The following list provides a brief overview over these steps:

- Prepare the LORIX One device
- Configure ChirpNest via SSH
- Configure ChirpNest through the ChirpStack Web Interface
- Connect KIWI Desktop with ChirpNest

After multiple measurements have been sent, the following screenshot in Figure 27 was taken from KIWI Desktop.

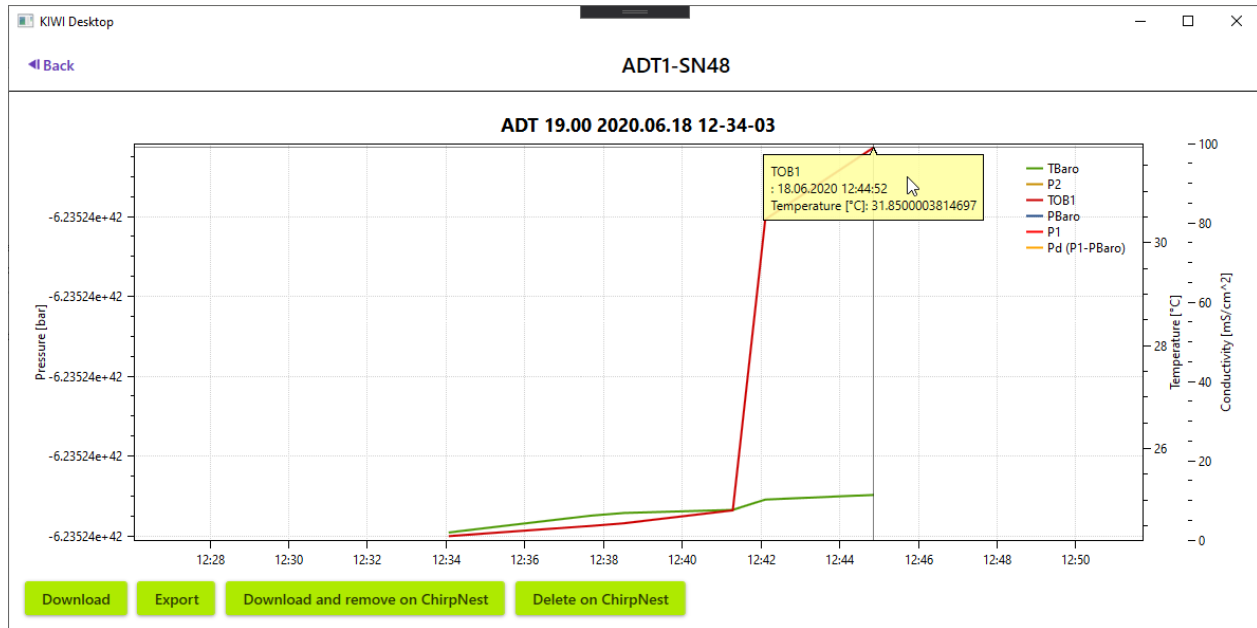


Figure 27: Screenshot of KIWI Desktop visualising measurements

Afterwards, we removed the power supply of the LORIX One by unplugging the ethernet cable which supplies the LORIX One with electricity (power over ethernet) and powered the LORIX One again after ten seconds.

KIWI Desktop was restarted and connected again. Then, the screenshot in Figure 28 was taken.

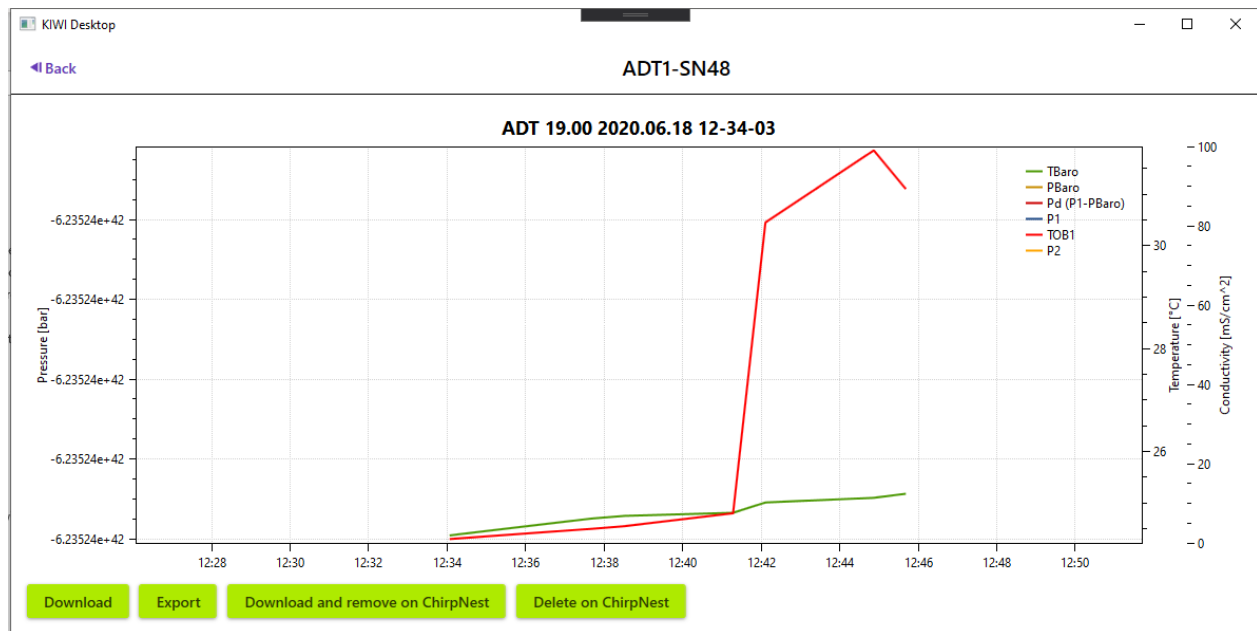


Figure 28: Screenshot of KIWI Desktop visualising measurements after LORIX One reboot

The measurements were then stored locally on the laptop where KIWI Desktop is running by pressing the button “Download & delete on remote”. After reloading the device overview, there were 0 measurements available as shown in Figure 29.

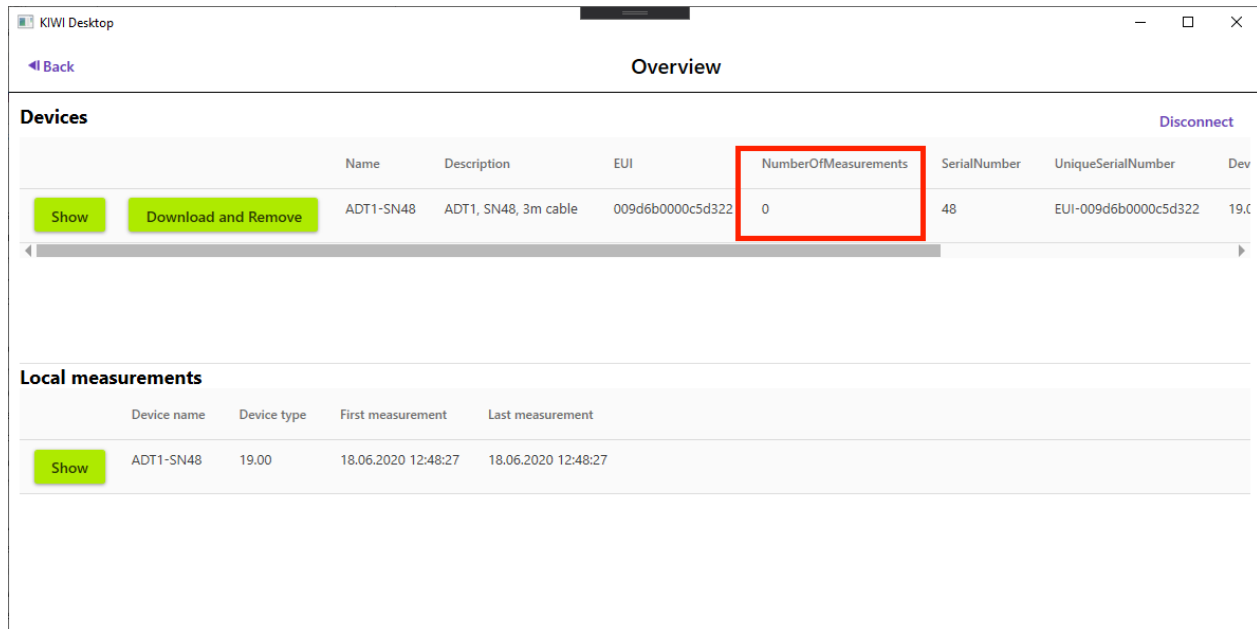


Figure 29: Screenshot of the device overview KIWI Desktop after deleting measurements

The connection to the LORIX One was cut by removing the ethernet cable. Afterwards, the locally saved measurements were visualised as shown in Figure 30.

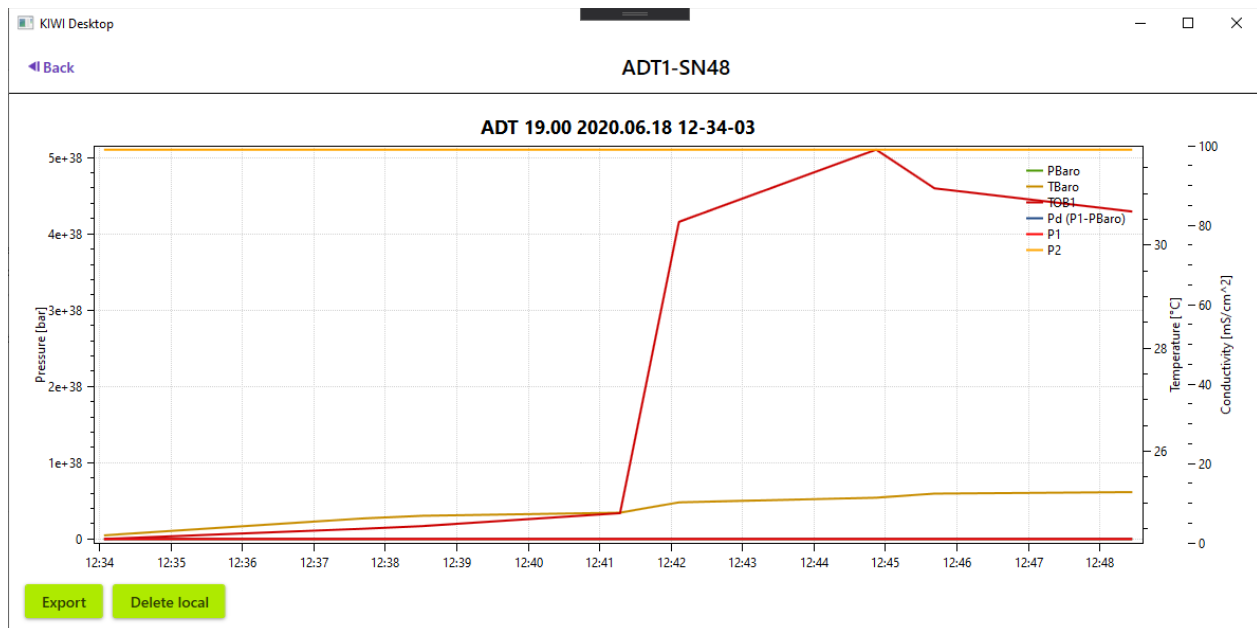


Figure 30: Screenshot of KIWI Desktop visualising locally saved measurements

Finally, the locally saved measurements were exported by pressing the button “Export” and by selecting a path and inserting a file name. Three exports were performed this way into all three available formats:

- Excel
- CSV
- KOLIBRI Format

The screenshot in Figure 31 shows the results of said exports.

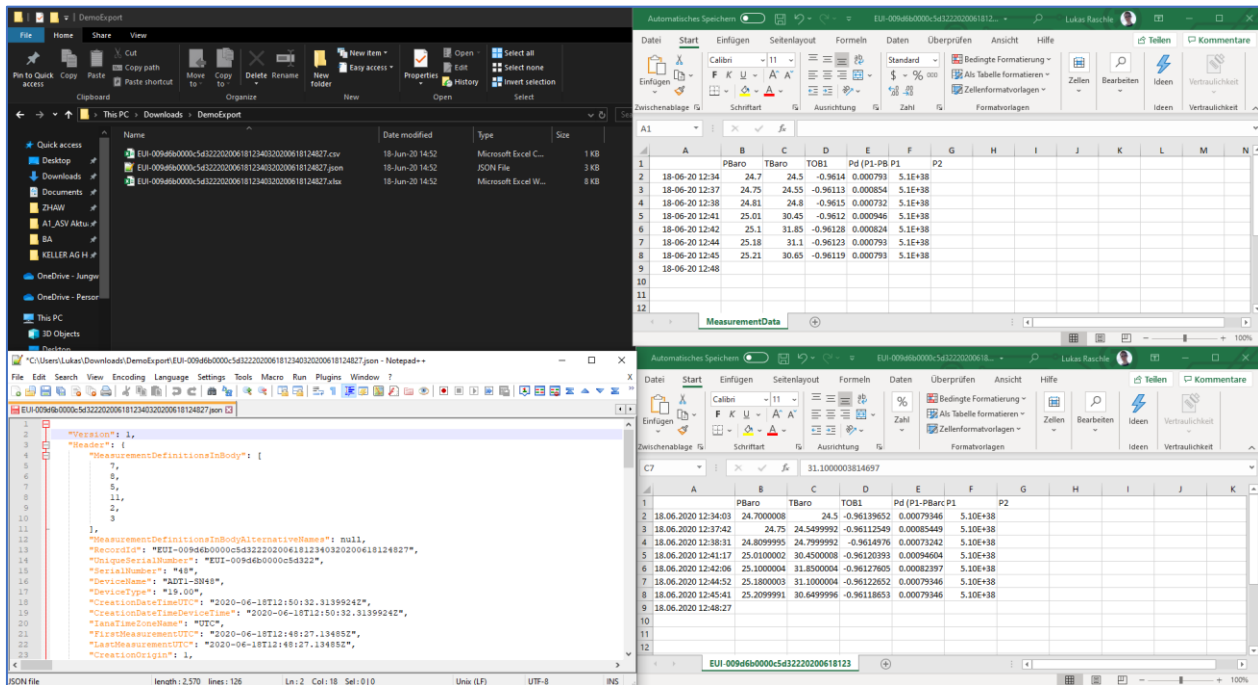


Figure 31: Screenshot of the KIWI Desktop measurement export results

## 8.3 Verification: Check Specification

The following list contains all specifications that were to be implemented within the proof of concept (as in “4.3 Specification”). Specifications that were not part of the proof of concept are left out.

The system test (see “8.2 System Test”) shows that the following specifications are fulfilled:

- GS-01: The LoRa stack unit shall receive measurements via LoRaWAN ✓
- GS-02: The LoRa stack unit shall store the received measurements. The measurements shall be persisted so that the data is still available after reboot ✓
- DS-01: KIWI Desktop requires installation and shall run on computers with the following operating systems:
  - DS-01.1: Windows ✓

- DS-03: The user shall be able to connect KIWI Desktop to the LoRa stack unit in a local network ✓
- DS-04: KIWI Desktop shall provide the following features while being connected to the LoRa stack unit:
  - DS-04.7: Transfer measurement data directly into KIWI Desktop (data is saved locally on the computer) ✓
- DS-05: KIWI Desktop shall provide following features with measurement data that was transferred to the computer as in DS-04.7
  - DS-05.1: Show measurement data in a table (✓)  
→ *Partially: It is not build into KIWI Desktop, but the measurement data can be viewed in a table after exporting it in Excel (DS-05.3.1) or CSV (DS-05.3.2) format.*
  - DS-05.2: Show measurement data in charts ✓
  - DS-05.3: Export measurement data in the following formats
    - DS-05.3.1: Excel ✓
    - DS-05.3.2: CSV ✓
    - DS-05.3.3: KOLIBRI Format ✓

The following hardware specifications are met according to [14]:

- GS-03: The LoRa stack unit shall be suitable for outdoor use with at least IP65 rating ✓
- GS-04: The LoRa stack unit offers the following hardware interfaces:
  - GS-04.1: a RJ45 interface ✓
  - GS-04.2: at least one common USB interface (USB-A, USB-C, USB-Mini, USB-Micro) ✓  
→ Mini USB
  - GS-04.3: an interface for an external alarm (e.g. light or siren) ✓  
→ Via USB and/or LAN

When operating with the devices as stated in GS-05 (10 devices sending a measurement per 30 minutes), 10 devices produce 48 measurements per day per device on 365 days per year. This results in 175'200 measurements per year (multiply 10, 48 and 365). If a single measurement would require 50 KB of storage (note that a single measurement certainly requires less than 50 KB of storage in PostgreSQL, see example in Listing 2 on page 57 and schema in Table 10 on page 56), the 175'200 measurements of one year would require 8'760'000 KB which is less than 9 GB. When using an SD card with 64 GB capacity, the storage of said measurements is possible. This shows that the following specification is fulfilled:

- GS-06: When operating with the devices as stated in GS-05, the LoRa stack unit can store all the measurements for at least one year ✓



The section “7.2.2 KIWI Server API” in chapter “7 Implementation” is a detailed documentation of the KIWI Server API which runs on the LoRa stack unit (within ChirpNest). This fulfils the following specification:

- GS-08: The APIs of the LoRa stack unit shall be documented in detail ✓

The only 3rd party software components of the LoRa stack unit / ChirpNest are ChirpStack components which are available under the MIT license [41]. This fulfils the following specification:

- GS-09: All 3rd party software that is used in the LoRa stack unit shall provide a license that allows free commercial use ✓

The manual “VII.I. Manual: Installation of a LORIX One with an ADT1 Device” in the appendix fulfils the following specification:

- GS-11: The LoRa stack unit set up shall be documented in a manual ✓

The implementation of KIWI Desktop shows the user interface to be easy to use since possible actions can be triggered by pressing labelled buttons which fulfils the following specification:

- DS-07: KIWI Desktop shall be easy to use, even for non-tech-savvy users ✓

The 3rd party software components that were used are listed in Table 14 on page 75. All of them are available under a license that allows free commercial use which fulfils the following specification:

- DS-08: All 3rd party software that is used in KIWI Desktop shall provide a license that allows free commercial use ✓

The following specification would require a simulation to prove that the LORIX One running ChirpNest is able to serve said devices. This simulation was not conducted in the scope of the bachelor thesis.

- GS-05: The LoRa stack unit shall be able to serve at least 10 devices sending a measurement per 30 minutes ?

The following specification would require the final version of ChirpNest to be tested to run for several months. Performing this test was not possible in the scope of the bachelor thesis.

- GS-07: The LoRa stack unit shall run stable for several months ?

## 9 Results and Conclusions

In this bachelor thesis, the customer need of being able to operate an IoT solution to gather and visualise measurements via LoRaWAN without the need to transfer any data over the Internet was analysed by conducting interviews and a survey with KELLER sales staff. The gained understanding of the requirements was used to create a specification and use cases were derived from it.

These specifications were prioritised, and the system was implemented in a proof of concept which covers the most important requirements. The resulting prototype consists of ChirpStack and KIWI Server on a LORIX One and the Windows program KIWI Desktop and meets the minimum requirements.

### 9.1 Proof of Concept

The result of the proof of concept consists of ChirpNest running on a LORIX One device. ChirpNest includes ChirpStack which covers the full LoRaWAN stack, allows the configuration of devices and saves the decoded measurement packets into a local PostgreSQL database. ChirpNest also includes a self-developed application “KIWI Server” which provides an API to retrieve the saved KELLER measurement packet’s contents. The Windows client software “KIWI Desktop” is also part of the proof of concept and is able to retrieve, save, visualise and export the measurement data.

The setup of the proof of concept system is explained step by step in a manual (see “VII.I. Manual: Installation of a LORIX One with an ADT1 Device” in the appendix) which empowers customers to set up and use the system on their own when it is publicly available. The API is documented in “7.2.2 KIWI Server API” which allows the integration into other systems. Also, manuals are provided for developers who want to extend KIWI Server (see “VII.II. Manual: Development of KIWI Server” in the appendix) and for updating ChirpNest with a new version of KIWI Server (see “VII.III. Manual: Create ChirpNest Yocto Image for LORIX One” in the appendix).

This proof of concept is the main result of this thesis, together with the documentation and the specification that goes further than what was implemented within the proof of concept. The applications of the proof of concept provide a good basis which can be extended (see “9.3 Further Work”).

### 9.2 General Findings

The following list contains the general findings. Findings that arose from the analysis of the customer needs are not part of this chapter (“9 Results and Conclusions”) since they are covered by and form the basis for the specification.

- The needed open-source components that can be used for commercial purpose around LoRaWAN are available and well documented with ChirpStack.
- Yocto offers the creation of an image where all the components are installed and where configuration can be included (ChirpNest).
- Yocto allows to support other devices in the future (see “9.3 Further Work” below)
- gRPC with .proto files allows the automatic generation of data access classes in C#.

## 9.3 Further Work

The prototype of the proof of concept can be built upon and the system can be expanded with additional features. Based on the analysis, the most important additional functions are the implementation of alarms and enabling the use without a local network (connect laptop with KIWI Desktop to a LORIX One directly with a USB or LAN cable). In addition, a web interface could be implemented, which would cover some functions of KIWI Desktop and could then be used without the installation of the Windows application KIWI Desktop. That web interface is already part of the specification as “KIWI Web” (see “4.3.3 KIWI Web”).

For customer use, it is recommended to add more functions to make the setup easier so that some of the manuals become obsolete and are automatically handled by KIWI Server. This includes adding and removing devices directly in KIWI Desktop (as specified in “DS-04.1”) and/or KIWI Web (as specified in “WS-02.1”) which would lead to no longer needing to use the ChirpStack web interface for configuring devices.

As mentioned above, alarming (e.g. when a measurement value surpasses a certain threshold) is another feature that would be useful for customers and which is already contained in the specification.

ChirpNest to support other devices than the LORIX One could be a future requirement of customers. Since ChirpNest is built with the Yocto project, it is possible to build ChirpNest for different hardware with little effort. This would allow customers to choose the gateway for themselves, which is especially important for other developments like the support for Grafana or other additional applications that require more performance. Based on the solution needed, each customer could decide if a low price gateway like the LORIX One fits them or if they need a high-end gateway like the Wirnet Station of Kerlink [13].

The current software is only for KELLER devices. It could be attractive for customers to be able to include third-party LoRa devices into this system. This feature would need an adjustment of the API to a more general format. Data of the different devices would need to be in a general format in the PostgreSQL and could be accessed regardless of whether they originate from a KELLER device or a third-party device.

A lot of parameters in ChirpNest are currently hard coded and could be allowed to be configured in a .toml file, e.g. the port used by KIWI Server or the credentials for the PostgreSQL database (so it could also be on another device in the same network).

KIWI Server can currently only remove measurement packet of the “device\_up” table of the ChirpStack integration. This way, not all data can be managed by KIWI Server since the “device\_up” table can be filled with packets that are not recognized as measurement packets and other tables like “device\_error” are completely ignored. KIWI Server could be extended to handle this data. Depending on the desired functionality, it could be sensible for KIWI Server to switch from the PostgreSQL integration to the MQTT integration and have an own data storage structure.

# References

## Sources

- [1] KELLER AG für Druckmesstechnik. *Company* | KELLER AG [Online]. Available: <https://keller-druck.com/en/company>. [22.5.2020].
- [2] A. Rüst, *Aufgabenstellung Bachelorarbeit BA20\_ruan\_04*. Winterthur, 2020.
- [3] LoRa Alliance®. *Home page* | LoRa Alliance® [Online]. Available: <https://loralliance.org/>. [18.6.2020].
- [4] The Things Industries., *The Things Network* [Online]. Available: <https://www.thethingsnetwork.org/>. [18.6.2020].
- [5] Lorient AG. *LORIENT - The LoRaWAN® Network Server Provider* [Online]. Available: <https://www.lorient.io/>. [18.6.2020].
- [6] Swisscom (Switzerland) Ltd. *Low Power Network (LPN) LoRaWAN®* [Online]. Available: <https://www.swisscom.ch/en/business/enterprise/offer/iot/lpn.html>. [18.6.2020].
- [7] KELLER AG für Druckmesstechnik. *ADT1-Tube* | KELLER AG [Online]. Available: <https://keller-druck.com/en/products/data-loggers/remote-transmission-units-with-data-logger/adt1-tube>. [18.6.2020].
- [8] KELLER AG für Druckmesstechnik. *ARC1-Tube* | KELLER AG [Online]. Available: <https://keller-druck.com/en/products/data-loggers/remote-transmission-units-with-data-logger/arc1-tube>. [18.6.2020].
- [9] A. O. e. al., *Value Proposition Design*, Wiley, 2014.
- [10] Strategyzer AG. (2019, July, 3). *Starting With The Customer* [Online]. Available: <https://www.strategyzer.com/blog/starting-with-the-customer>. [18.6.2020].
- [11] KELLER AG für Druckmesstechnik. *Level measuring device ADT1* | KELLER AG [Online]. Available: <https://keller-druck.com/en/company/blog/level-measuring-device-adt1>. [22.5.2020].
- [12] Multi-Tech Systems, Inc. *MultiTech Conduit®* [Online]. Available: <https://www.multitech.com/brands/multiconnect-conduit>. [22.5.2020].
- [13] Kerlink. *Wirnet Station* [Online]. Available: <http://www.kerlink.com/product/wirnet-station/>. [22.5.2020].
- [14] Wifx Sàrl. *LORIX One* [Online]. Available: <https://www.lorixone.io/de/produkte>. [22.5.2020].

- [15] O. Brocaar. *ChirpStack, open-source LoRaWAN® Network Server stack* [Online]. Available: <https://www.chirpstack.io/>. [19.6.2020].
- [16] O. Brocaar. *ChirpStack Gateway OS* [Online]. Available: <https://www.chirpstack.io/gateway-os/>. [17.6.2020].
- [17] The Things Industries. *Home | The Things Stack for LoRaWAN* [Online]. Available: <https://thethingsstack.io/>. [19.6.2020].
- [18] P. Gotthard. *lorawan-server | Compact server for private LoRaWAN networks* [Online]. Available: <https://gotthardp.github.io/lorawan-server/>. [19.6.2020].
- [19] O. Brocaar. *Sending and receiving device data - ChirpStack Application Server* [Online]. Available: <https://www.chirpstack.io/application-server/integrate/sending-receiving/>. [22.5.2020].
- [20] InfluxData Inc. *Hardware sizing guidelines | InfluxData Documentation* [Online]. Available: [https://docs.influxdata.com/influxdb/v1.8/guides/hardware\\_sizing/](https://docs.influxdata.com/influxdb/v1.8/guides/hardware_sizing/). [22.5.2020].
- [21] The PostgreSQL Global Development Group. *PostgreSQL: Documentation: Resource Consumption* [Online]. Available: <https://www.postgresql.org/docs/12/runtime-config-resource.html#RUNTIME-CONFIG-RESOURCE-MEMORY>. [22.5.2020].
- [22] ThingsBoard, Inc. *Installing ThingsBoard CE on Ubuntu Server* [Online]. Available: <https://thingsboard.io/docs/user-guide/install/ubuntu/>. [22.5.2020].
- [23] O. Brocaar. *ChirpStack Application Server source* [Online]. Available: <https://www.chirpstack.io/application-server/community/source/>. [22.5.2020].
- [24] A. Rodríguez. *GitHub, LiveCharts* [Online]. Available: <https://github.com/Live-Charts/Live-Charts>. [22.5.2020].
- [25] A. Deniel. *GitHub, Microcharts* [Online]. Available: <https://github.com/dotnet-ad/Microcharts>. [22.5.2020].
- [26] OxyPlot. *GitHub, OxyPlot* [Online]. Available: <https://github.com/oxyplot/oxyplot>. [22.5.2020].
- [27] O. Brocaar. *ChirpStack architecture* [Online]. Available: <https://www.chirpstack.io/overview/architecture/>. [6.6.2020].
- [28] O. Brocaar. *ChirpStack Application Server API* [Online]. Available: <https://www.chirpstack.io/application-server/integrate/api/>. [6.6.2020].
- [29] O. Brocaar. *PostgreSQL - ChirpStack Application Server* [Online]. Available: <https://www.chirpstack.io/application-server/integrate/sending-receiving/postgresql/>. [23.5.2020].
- [30] O. Brocaar. *Device Profile management - ChirpStack Application Server* [Online]. Available: <https://www.chirpstack.io/application-server/use/device-profiles/>. [5.6.2020].

- [31] KELLER AG für Druckmesstechnik. *GitHub, KellerAgTheThingsNetworkPayloadDecoder* [Online]. Available: <https://github.com/KELLERAGfuerDruckmesstechnik/KellerAgTheThingsNetworkPayloadDecoder>. [23.5.2020].
- [32] KELLER AG für Druckmesstechnik. *ADT1 - LoRa data communication protocol* [Online]. Available: [https://docs.kolibricloud.ch/sending-technology/ADT1%20LoRa%20data%20communication%20protocol%2002\\_2020.pdf](https://docs.kolibricloud.ch/sending-technology/ADT1%20LoRa%20data%20communication%20protocol%2002_2020.pdf). [5.6.2020].
- [33] O. Brocaar. *GitHub, chirpstack-application-server/ui/README.md* [Online]. Available: <https://github.com/brocaar/chirpstack-application-server/blob/master/ui/README.md>. [6.6.2020].
- [34] Grafana Labs. *Grafana requirements* [Online]. Available: <https://grafana.com/docs/grafana/latest/installation/requirements/>. [6.6.2020].
- [35] Syncfusion Inc. *Buy .NET Components and Products | Pricing | Syncfusion* [Online]. Available: <https://www.syncfusion.com/sales/products>. [18.6.2020].
- [36] F. B. A. P. Manuel de Leon. *GitHub, ClosedXML* [Online]. Available: <https://github.com/closedxml/closedxml/>. [18.6.2020].
- [37] O. Brocaar. *ChirpStack Gateway OS - Service monitoring* [Online]. Available: <https://www.chirpstack.io/gateway-os/use/monitoring/>. [17.6.2020].
- [38] Google LLC. *Protocol Buffers - Language Guide (proto3)* [Online]. Available: <https://developers.google.com/protocol-buffers/docs/proto3>. [18.6.2020].
- [39] The Go Authors. *Package sql - The Go programming language* [Online]. Available: <https://golang.org/pkg/database/sql/>. [23.5.2020].
- [40] O. Brocaar. *GoDoc - chirpstack-api* [Online]. Available: <https://godoc.org/github.com/brocaar/chirpstack-api/go/as/external/api>. [18.6.2020].
- [41] O. Brocaar. *ChirpStack open-source LoRaWAN® Network Server stack* [Online]. Available: <https://www.chirpstack.io/overview/>. [18.5.2020].

## Figures

Figure 1: KOLIBRI Desktop overview .....	11
Figure 2: KOLIBRI Mobile overview.....	12
Figure 3: KOLIBRI Cloud overview .....	13
Figure 4: LoRaWAN Network .....	15
Figure 5: ADT1-tube [7] .....	17
Figure 6: ARC1-tube [8].....	17
Figure 7: Scenarios of having IoT capable sensors or not and of using the Internet for data transmission or not .....	18
Figure 8: Value proposition canvas by Strategyzer [10] .....	21
Figure 9: Value proposition canvas customer profile for “Miguel López” (mine in Peru) .....	22
Figure 10: Value proposition canvas customer profile for “Walter Schmidt” (moorland in Germany) .....	23
Figure 11: Value proposition canvas customer profile for “Ivan Kowalski” (Poland, security concerns) ..	23
Figure 12: Value proposition canvas customer profile for “Michelle González” (museum in Spain) .....	24
Figure 13: Value map of the value proposition canvas for all four personas .....	25
Figure 14: System Overview.....	26
Figure 15: Use case diagram.....	33
Figure 16: Concept overview .....	53
Figure 17: Measurements sequence diagram.....	54
Figure 18: Concept illustration of the KIWI Server API.....	60
Figure 19: Mockup overview unconnected.....	61
Figure 20: Mockup overview connected .....	62
Figure 21: Mockup measurement online.....	63
Figure 22: Mockup measurement local .....	64
Figure 23: Mockup combine measurements .....	65
Figure 24: KIWI Desktop state diagram .....	65
Figure 25: System setup.....	67
Figure 26: KIWI Desktop project structure .....	74
Figure 27: Screenshot of KIWI Desktop visualising measurements .....	77
Figure 28: Screenshot of KIWI Desktop visualising measurements after LORIX One reboot .....	77
Figure 29: Screenshot of the device overview KIWI Desktop after deleting measurements .....	78
Figure 30: Screenshot of KIWI Desktop visualising locally saved measurements .....	78
Figure 31: Screenshot of the KIWI Desktop measurement export results .....	79

## Tables

Table 1: Persona “Miguel López” representing a mine in Peru .....	19
Table 2: Persona “Walter Schmidt” representing local authorities (moorland in Germany) .....	20
Table 3: Persona “Ivan Kowalski” representing a Polish fuel company with security concerns ..	20
Table 4: Persona “Michelle González” representing a museum in Spain .....	21
Table 5: LoRa stack unit specification .....	28
Table 6: KIWI Desktop specification .....	30
Table 7: KIWI Web specification .....	31
Table 8: LoRa gateway comparison [12] [13] [14] .....	48
Table 9: Steps required to setup a device on a new ChirpStack instance .....	55
Table 10: PostgreSQL table schema of “device_up” .....	56
Table 11: General fields resulting from the decoder function [32] .....	58
Table 12: Measurement packet fields resulting from the decoder function [32] .....	58
Table 13: Device information packet fields resulting from the decoder function [32] .....	59
Table 14: NuGet packages used in KIWI Desktop .....	75



# Appendix

## I. List of Abbreviations

Abbreviation	Meaning
KELLER	Keller AG für Druckmesstechnik
InES	Institute of Embedded Systems
InIT	Institute of Applied Information Technology
IoT	Internet of things
GSM	Global System for Mobile Communications
LoRa	long range radio modulation technology
LoRaWAN	Long Range Wide Area Network
FTP	File transfer protocol
TTN	The Things Network
EUI	Extended Unique Identifier
MVP	Minimum viable product
Go	Programming language "Go" ( <a href="https://golang.org">golang.org</a> )
SSH	Secure Shell
JWT	JSON Web Token, are issued by ChirpStack and “prove” authentication
ES5	ECMAScript 5, a standardized JavaScript specification from 2009
API	Application Programming Interface
REST API	Representational State Transfer
JSON	JavaScript Object Notation

## II. Thesis Objective (Aufgabenstellung)

### Aufgabenstellung Bachelorarbeit BA20\_ruan\_04

#### Autarke Cloudlösung für Drucksensoren

für Samuel Egger und Lukas Raschle

#### 1 Beschreibung

Die KELLER AG für Druckmesstechnik ist Europas führender Hersteller von isolierten Druckaufnehmern und Drucktransmittern. KELLER verkauft seit 10 Jahren IoT-Systeme basierend auf GSM-Modulen und FTP/Mail-Servern. Mit dem Aufkommen von IoT stellt KELLER den Kunden eine Cloud-Lösung ([www.kolibricloud.ch](http://www.kolibricloud.ch)) als Dienstleistung zur Verfügung.

##### Arbeitsbeschreibung

Seit zwei Jahren verkauft KELLER auch LoRaWAN-Übertragungsgeräte, welche mit der Cloud-Lösung benutzt werden. Im konservativen Wasser-"Business" (<https://keller-h2o.com/>) bestehen aber noch viele Ängste: Insofern hat KELLER viele mögliche Kunden, die die Vorteile von IoT-Geräte nutzen wollen, aber durch das öffentliche Internet abgeschreckt werden.

##### Ziel

Gesucht ist eine Lösung für die genannten Kunden mit dem Ziel, die KELLER-Sensoren verkaufen zu können. Ziel ist es nicht, eine Dienstleistung aufzubauen. Das System und die nötige HW/SW ist kein Produkt, welches verkauft wird. Darum und auch aus sicherheitstechnischen Gründen ist eine Open-Source-Lösung zu bevorzugen. Die Lösung kann und soll von allen Kunden benutzt werden und soll wo möglich auch auf existierende Open-Source-Lösungen aufbauen.

##### Umsetzung

Die KELLER wünscht eine Lösung, um Messdaten "lokal" speichern zu können mit den nötigen Schnittstellen, um an diese Daten zu gelangen. Denkbar ist die Verbindung eines LoRa-Gateways mit einem lokalen Server mit einem Datenspeicher. Teil der Arbeit ist einen Anforderungsbeschrieb zu erstellen an die nötige Hardware und Software. Insbesondere unklar ist noch, über welche Schnittstellen die Daten aus dem Server ausgelesen werden. Denkbar ist eine Client-Software, die über eine Kabelverbindung Daten auslesen und visualisieren lässt. Auch möglich ist eine Schnittstelle, die im lokalen Netzwerk zugänglich ist und somit die Möglichkeit besteht, diese auch über eine VPN-Verbindung zu benutzen. Gewünscht ist neben der Evaluation auch ein funktionierender Proof-of-Concept.

#### 2 Aufgabe

Die Projektarbeit umfasst die folgenden Teilaufgaben:

- Wahrnehmen der Projektleitung insbesondere Erstellen von Projektplan und protokollieren von Besprechungen.
- Analyse der Problemstellung zusammen mit Keller AG
- Definieren der Anforderungen und beschreiben von Use Cases
- Beschreiben und Bewerten von Lösungsvarianten
- Ausarbeiten eines Konzeptes
- Implementieren des Systems als Proof-of-Concept
- Systemverifikation und -validation

- Dokumentation der Arbeit: Erstellen eines Berichtes über die Arbeit.

### 3 Allgemeine Randbedingungen

- In der Regel findet eine wöchentliche Besprechung mit den Betreuern statt.
- Zusätzlich zu den Vorgaben in [1] soll die Dokumentation folgende Punkte enthalten
  - Dokumentation der Konzepte und Lösungen: Aus welchen Komponenten besteht das System und wie funktioniert es?
  - Verbesserungsvorschläge bzgl. Schlüsseigenschaften bzw. Dokumentation von Herausforderungen und Schwierigkeiten
  - Vollständige Informationen für den Nachbau des Systems
  - Begründungen zur Nachvollziehbarkeit von Design Entscheidungen
- Das Schreiben der Dokumentation soll parallel zur Umsetzung erfolgen. Die Planung soll Meilensteine für die Dokumentation enthalten. Drei Wochen vor dem Abgabetermin der Arbeit soll ein erster Entwurf abgegeben werden.

### 4 Allgemeine Rahmenbedingungen

- Ausgabe der Arbeit: Montag, 10.02.2020
- Abgabe der Arbeit: Freitag, 05.06.2020
- Umfang: 12 Credits. Dies entspricht einer Arbeitsbelastung von etwa 360h.
- Die Bewertung erfolgt an Hand des Rasters in [2].  
Projektverlauf, Leistung, Arbeitsverhalten  $\frac{1}{4}$ ; Qualität der Ergebnisse  $\frac{1}{3}$ ; Form und Inhalt des Berichts und der Präsentation  $\frac{1}{4}$
- Beachten Sie die Anforderungen der Schule auf dem Intranet und in den Emails des Studiengangsekretariates.

### 5 Betreuer/Ansprechpersonen

Betreuer: Prof. Andreas Rüst, ZHAW/InES  
Kurt Bleisch, ZHAW/InIT

Industriepartner: Keller AG, 8404 Winterthur  
Herr Sebastian Mojado

### 6 Literaturverzeichnis

- [1] T. Järmann, Berichtstruktur einer Projekt- oder Bachelorarbeit an der SoE, Winterthur: ZHAW SoE, 2011.
- [2] T. Järmann, Bewertungsraster zur Projekt- und Bachelorarbeit an der SoE, Winterthur: ZHAW SoE, 2011.

### III. Survey with Results

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*



*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

*[the survey with results is not included in the public version for reasons of confidentiality]*

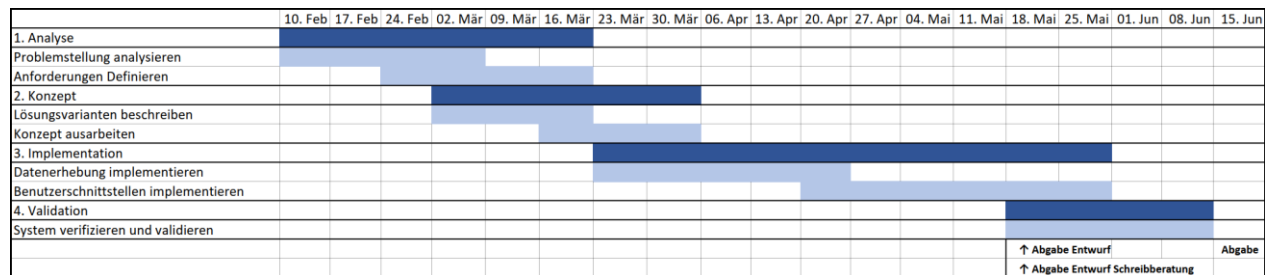


*[the survey with results is not included in the public version for reasons of confidentiality]*

## IV. Project Management

#### ***IV.I. Schedule***

The following schedule was used and continuously updated during the project:



#### ***IV.II. Risk Management***

The risks were continuously monitored using the following table:

Risiko Beschreibung	Wahrscheinlichkeit	Auswirkung	Risiko	Massnahme	eliminiert
Projektumfang zu gross	mittel	klein	<del>mittel</del>	Definieren welche Komponenten im Umfang dieses Projektes umgesetzt werden	✓
Opensource LoRa server lässt sich nicht lokal aufsetzen	sehr klein	gross	<del>hoch</del>	frühestmöglich mit dem aufsetzen eines LoRa Servers beginnen um das Risiko zu eliminieren	✓
geeignete Hardware nicht verfügbar	sehr klein	katastrophal	<del>hoch</del>	frühestmöglich nach geeigneter Hardware umsehen um benötigte um Besorgungen zu erledigen	✓
Zeitplanung zu knapp bemessen	mittel	mittel	<del>hoch</del>	Fortschritt regelmässig mit Zeitplan abgleichen um Verzögerungen früh zu erkennen und passende Massnahmen zu ergreifen	✓
Coronavirus-Massnahmen von Bund/Kanton/ZHAW erschweren Arbeit durch Bewegungseinschränkungen	wahrscheinlich	klein	<del>hoch</del>	Infrastruktur für Remote-Meetings vorbereiten	✓
Zu wenig Zeit mit Arbeit/Studium/BA	mittel	klein	<del>hoch</del>	Klar priorisieren und allenfalls Zeit für BA schaffen	✓
Verzögerung aufgrund fehlendem Know-How (v. A. in Yocto / embedded Linux)	<del>wahrscheinlich</del> unwahrscheinlich	mittel	<del>hoch</del> mittel	Scope für minimal viable product verkleinern, Technologien verwenden, die einfacheres Setup erlauben (Go, PostgreSQL), mit Herstellern in Verbindung setzen	✓

## V. Protocol: Verification at KELLER

Date: Tuesday, 9<sup>th</sup> of June 2020

Location: KELLER headquarter in Winterthur

Participants: Sebastian Mojado, Lukas Raschle, Samuel Egger  
(the presentation was watched by about 10 interested KELLER employees of the development department)

The following components were set up:

- ChirpNest on a LORIX One
- Two ADT1 devices (configured in ChirpNest)
- Windows laptop with KIWI Desktop and browser
- Local network that connects the LORIX One and the Windows laptop

Course of action:

- Presentation:  
Lukas Raschle and Samuel Egger presented the project to the interested KELLER employees. The presentation took about 20 minutes and included an overview over and explanation of the involved components, a demonstration of the configuration in the ChirpStack Application Server web interface, a demonstration of a measurement being sent by an ADT1 and a demonstration of KIWI Desktop fetching and visualizing the measurements from ChirpNest.
- Verification:  
Sebastian Mojado, Lukas Raschle and Samuel Egger performed the verification with KELLER where Sebastian Mojado represented KELLER. Lukas and Samuel provided an overview over the components and their dependencies and answered questions of Sebastian Mojado. Also, the specification was gone through as protocolled below.

Specification:

The following table shows the specifications that were to be implemented within the proof of concept. Specifications that were not part of the proof of concept are left out.

Title	met?	Comment
GS-01: The LoRa stack unit shall receive measurements via LoRaWAN	✓	–
GS-02: The LoRa stack unit shall store the received measurements. The measurements shall be persisted so that the data is still available after reboot	✓	–
GS-03: The LoRa stack unit shall be suitable for outdoor use with at least IP65 rating	✓	–

Title	met?	Comment
GS-04: The LoRa stack unit offers the following hardware interfaces:		
• GS-04.1: a RJ45 interface	✓	–
• GS-04.2: at least one common USB interface (USB-A, USB-C, USB-Mini, USB-Micro)	✓	Yes: Mini USB
• GS-04.3: an interface for an external alarm (e.g. light or siren)	✓	Yes: USB and LAN
GS-05: The LoRa stack unit shall be able to serve at least 10 devices sending a measurement per 30 minutes	✗	Not met: Simulation required where the following aspects are tested: <ul style="list-style-type: none"> <li>• CPU load is okay</li> <li>• Memory: no leak</li> <li>• SD read/write speed is sufficient</li> </ul> When such a simulation is conducted with two ADT1 devices, the specification can be considered as met from KELLER's point of view.
GS-06: When operating with the devices as stated in GS-05, the LoRa stack unit can store all the measurements for at least one year	✗	Not met: Needs to be proven by a calculation.
GS-07: The LoRa stack unit shall run stable for several months	✗	Cannot be verified today, requires a test over several months.
GS-08: The APIs of the LoRa stack unit shall be documented in detail	✗	Not met: Detailed documentation does not yet exist.
GS-09: All 3rd party software that is used in the LoRa stack unit shall provide a license that allows free commercial use	✓	Yes: ChirpStack
GS-11: The LoRa stack unit set up shall be documented in a manual	✗	Not met: Manuals do not yet exist.
DS-01: KIWI Desktop requires installation and shall run on computers with the following operating systems:		
• DS-01.1: Windows	✓	–
DS-03: The user shall be able to connect KIWI Desktop to the LoRa stack unit in a local network	✓	–
DS-04: KIWI Desktop shall provide the following features <b>while being connected</b> to the LoRa stack unit:		

Title	met?	Comment
<ul style="list-style-type: none"> <li>DS-04.7: Transfer measurement data directly into KIWI Desktop (data is saved locally on the computer)</li> </ul>	✓	–
DS-05: KIWI Desktop shall provide following features <b>with measurement data that was transferred</b> to the computer as in DS-04.7		
<ul style="list-style-type: none"> <li>DS-05.1: Show measurement data in a table</li> </ul>	(✓)	Partially: It is not built into KIWI Desktop, but the measurement data can be viewed in a table after exporting it in Excel (DS-05.3.1) or CSV (DS-05.3.2) format.
<ul style="list-style-type: none"> <li>DS-05.2: Show measurement data in charts</li> </ul>	✓	–
<ul style="list-style-type: none"> <li>DS-05.3: Export measurement data in the following formats</li> </ul>		
<ul style="list-style-type: none"> <li>DS-05.3.1: Excel</li> </ul>	✓	–
<ul style="list-style-type: none"> <li>DS-05.3.2: CSV</li> </ul>	✓	–
<ul style="list-style-type: none"> <li>DS-05.3.3: KOLIBRI Format</li> </ul>	✓	–
DS-07: KIWI Desktop shall be easy to use, even for non-tech-savvy users	✓	–
DS-08: All 3rd party software that is used in KIWI Desktop shall provide a license that allows free commercial use	✓	Yes: OxyPlot, ClosedXML

Remaining tasks for verification:

- Perform simulation for GS-05
- Conduct calculation for GS-06
- Deliver detailed documentation of the APIs of the LoRa stack unit for GS-08
- Deliver manual for GS-11

Not verifiable in the scope of the bachelor thesis:

- GS-07: Needs to be tested over several months

## VI. Used tools

The following tools were used to manage this project:

Tool	Used for	Link
Azure DevOps	Code repository Documentation (Wiki)	<a href="https://dev.azure.com/">https://dev.azure.com/</a>
OneNote	Notes and tasks	<a href="https://www.onenote.com/">https://www.onenote.com/</a>
Diagrams.net	Diagrams and illustrations	<a href="https://www.diagrams.net/">https://www.diagrams.net/</a>
OneDrive	Document share	<a href="https://onedrive.live.com/">https://onedrive.live.com/</a>
Microsoft Teams	Virtual meetings	<a href="https://teams.microsoft.com/">https://teams.microsoft.com/</a>

The following tools were used for the development:

Tool	Used for	Version
Windows 10	Almost all tasks	Version 1903 (Build 18362.900)
Visual Studio Enterprise 2019	KIWI Desktop	16.5.4
Visual Studio Code	ChirpNest, KIWI Server	1.46.0
.Net Framework	KIWI Desktop	4.7.2
GSM Setup (by KELLER)	Configuration of ADT1	4.05
Ubuntu (on WSL)	KIWI Server	20.04 LTS
Go (on WSL)	KIWI Server	1.14.4 linux/amd64
Ubuntu (in virtual machine hosted by Hyper-V)	Build of Yocto image	20.04 LTS
Docker (in virtual machine)	Build of Yocto image	19.03.11
ChirpStack Gateway OS	ChirpNest	3.2.0test1, <a href="#">commit 1dec7652</a>
balenaEtcher	Flashing linux image on SD-card	1.5.99

## VII. Manuals

### VII.I. *Manual: Installation of a LORIX One with an ADT1 Device*

Requirements:

- LORIX One with power over ethernet cable
  - Power supply
  - SD card (minimum 8 GB, 10 MB/s writing speed or more recommended)
  - ChirpNest image file for LORIX One  
(see “VII.III. Manual: Create ChirpNest Yocto Image for LORIX One” for creation of this image which ends with “.sdimg.gz”, images will also be provided to download)
- ADT1 device
  - The following parameters or the possibility to configure the ADT1 (see “Configure ADT1 Device” on page 117, note that installing a software and a Mini USB cable are required for configuration):
    - Device EUI value
    - Device address
    - Network session key
    - Application session key
- Windows computer
  - SD card reader
  - Browser
  - SSH client
  - Flashing software (e.g. balenaEtcher from [balena.io/etcher](https://balena.io/etcher))
- Local network that allows communication between the computer and the LORIX One
- Possibility to discover the IP address of the LORIX One after boot  
(e.g. through a web interface on the router)

#### **Prepare the LORIX One device**

Insert the SD card into the Windows computer and flash the ChirpNest image onto it using the third-party flashing software.

Connect the LORIX One to the local network, put the SD card into the LORIX One and power it.

#### **Configure ChirpNest via SSH**

Discover the IP address of the LORIX One e.g. through a web interface on the router.

Connect to the LORIX One using an SSH client.

Username: “admin”

Password: “admin”

In Windows 10, PowerShell can be used with the following command (replace “IP\_Address” with the actual IP address of the LORIX One):

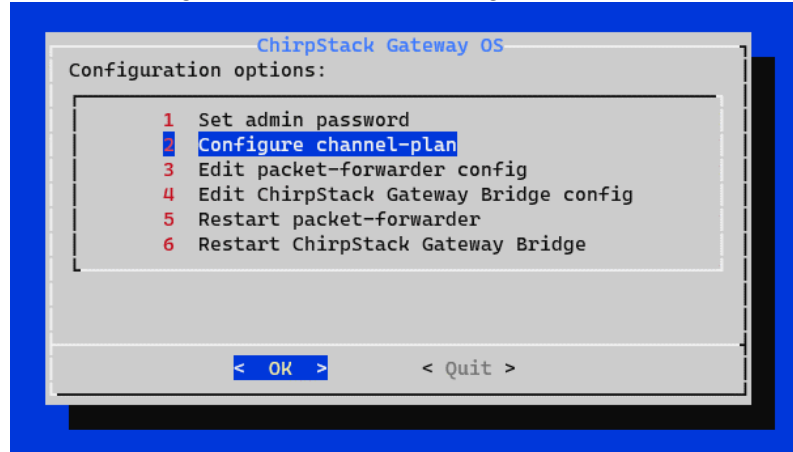
```
ssh admin@IP_Address
```

When connected to the LORIX One, type the following command:

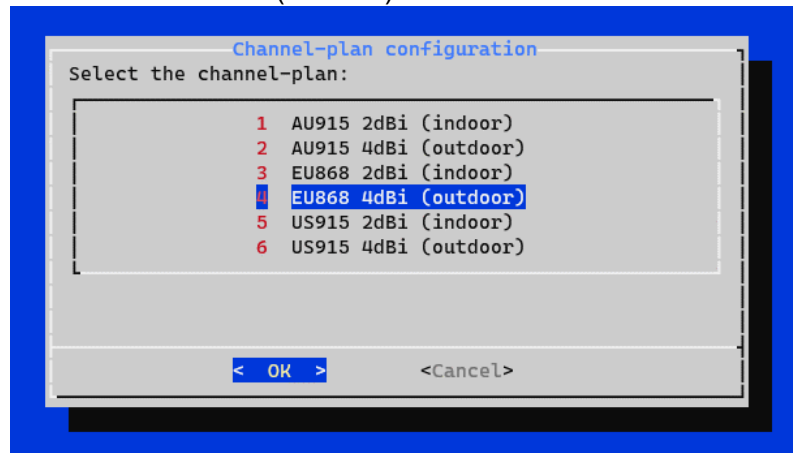
```
sudo gateway-config
```

Enter the password “admin” again when prompted.

Select “Configure channel-plan” using the arrow buttons and enter:

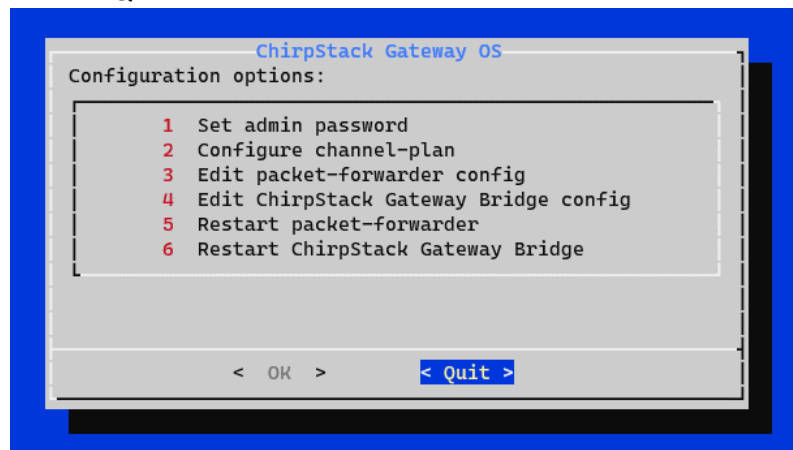


Select “EU868 4dBi (outdoor)”:



Confirm the following four messages with enter.

Select “Quit”:





Check if date and time are correct with the following command:

```
$ date
```

If date and time are not correct, set it with the following command (enter the current UTC time):

```
$ sudo date -s '2020-06-09 07:10:00'
```

Terminate the SSH connection by entering the following command:

```
exit
```

## Commands for ChirpNest

Below are some commands that can be executed on ChirpNest when connected via SSH as explained in “Configure ChirpNest via SSH” on page 111.

Consider <https://www.chirpstack.io/gateway-os/> for more information.

Command	Description
\$ sudo monit status	Show status of all services
\$ sudo monit status kiwi-server	Show status of kiwi-server
\$ sudo monit status chirpstack-application-server	Show status of chirpstack-application-server
\$ sudo monit stop all	Stop all services
\$ sudo monit start all	Start all services
\$ sudo monit stop kiwi-server	Stop kiwi-server
\$ sudo monit start kiwi-server	Start kiwi-server
\$ sudo gateway-config	Open configuration
\$ sudo cd /opt/kiwi-server/kiwi-server	Start kiwi-server manually
\$ sudo /opt/chirpstack-application-server/chirpstack-application-server	Start chirpstack-application-server manually
\$ sudo reboot && exit	Trigger reboot and quit SSH connection
\$ sudo nano /etc/chirpstack-application-server/chirpstack-application-server.toml	Change configuration file of chirpstack-application-server
\$ sudo tail -f /var/log/messages	Show logs (live update)
\$ date	Show current date and time (UTC)
\$ sudo date -s '2020-06-09 07:10:00'	Set date and time (UTC)
\$ sudo -u postgres /usr/bin/pg_ctl -D /var/lib/postgresql/data -l logfile stop	Stop PostgreSQL server
\$ sudo -u postgres /usr/bin/pg_ctl -D /var/lib/postgresql/data -l logfile start	Start PostgreSQL server
\$ sudo -u postgres psql	Connect to PostgreSQL server

<pre>\$ psql -h localhost -U chirpstack_as_events -W chirpstack_as_events</pre>	Connect to PostgreSQL server to table "chirpstack_as_events" with user "chirpstack_as_events" (password will be prompted which is "dbpassword")
<pre>\$ sudo monit stop all &amp;&amp; sudo rm -rfv /data/upperdir/ &amp;&amp; sudo reboot &amp;&amp; exit</pre>	Stop all services, remove all changes made on the file system, trigger reboot and quit

## Configure ChirpNest through the ChirpStack Web Interface

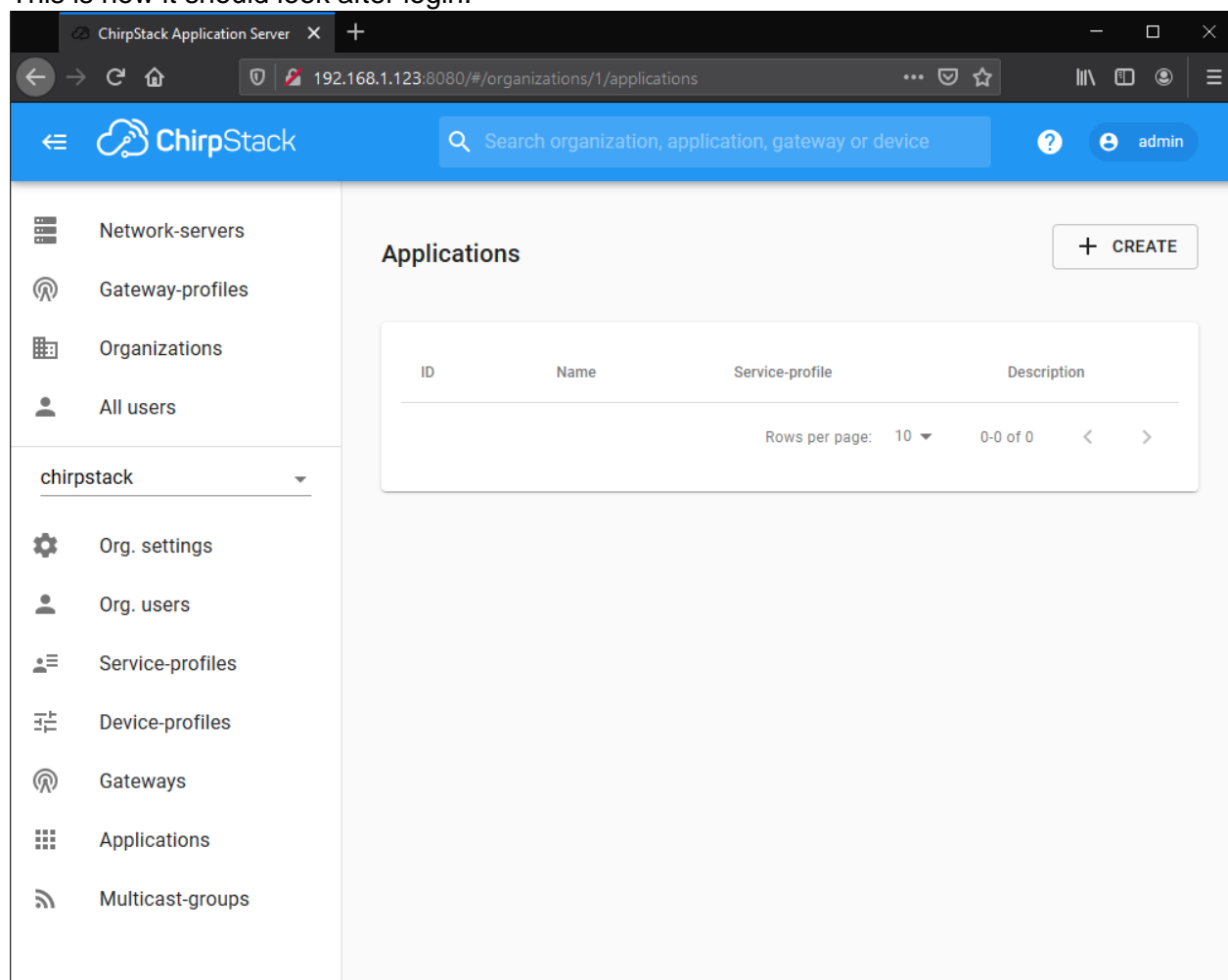
Connect to the LORIX One ChirpStack web interface with a browser on port 8080 (i.e. call [http://IP\\_Address:8080](http://IP_Address:8080) and replace "IP\_Address" with the actual IP address of the LORIX One).

Log in using the following credentials:

Username: "admin"

Password: "admin"

This is how it should look after login:



Navigate to “Network-servers”, click the “+ Add” button, enter the following data and click “Add network-server”:

- General
  - Network-server name: “ChirpNest-Network-Server”
  - Network-server server: “localhost:8000”
- Gateway discovery
  - leave all fields unchanged
- TLS certificates
  - leave all fields unchanged

Navigate to “Service-profiles”, click the “+ Create” button, enter the following data and click “Create service-profile”:

- Service-profile name: “ChirpNest-Service-Profile”
- Network-server: select “ChirpNest-Network-Server”
- remaining fields: leave unchanged

Navigate to “Device-profiles”, click the “+ Create” button, enter the following data and click “Create device-profile” (the following parameters are chosen specifically for and tested with the ADT1; using OTAA instead of ABP should also be possible):

- General
  - Device-profile name: “device\_profile\_1.0.2\_B\_abp\_ADT1”
  - Network-server: select “ChirpNest-Network-Server”
  - LoRaWAN MAC version: select “1.0.2”
  - LoRaWAN Regional Parameters revision: select “B”
  - remaining fields: leave unchanged
- Join (OTAA / ABP)
  - leave all fields unchanged
- Class-B
  - leave all fields unchanged
- Class-C
  - leave all fields unchanged
- Codec
  - Payload codec: select “Custom JavaScript codec functions”
  - Decode function:  
paste the decode function, see “Decode function for ADT1” on page 119  
make sure the function signature is `function Decoder(port, bytes)`
  - Encode function:  
leave unchanged

Navigate to “Applications”, click the “+ Create” button, enter the following data and click “Create application”:

- Application name: “ChirpNest-Application”
- Application description:  
“This is the single application configured on this ChirpNest environment.”
- Service-profile: select “ChirpNest-Service-Profile”

Navigate to “Applications”, select the single existing application “ChirpNest-Application” (click on the name) which navigates to the “Devices” tab of that application, then click the “+ Create” button, enter the following data and click “Create device”:

- General
  - Device name: enter any name consisting of letters, numbers and hyphens (e.g. “ADT1-No-92” where 92 is the serial number)
  - Device description: enter any description (e.g. “ADT1 with serial number 92”)
  - Device EUI: enter the device EUI of your device (e.g. “00 9D 6B 00 00 C5 D2 4F”)
  - Device-profile: select “device\_profile\_1.0.2\_B\_abp\_ADT1”
  - Disable frame-counter validation: check
- Variables
  - leave unchanged
- Tags
  - leave unchanged

Navigate to “Applications”, select the single existing application “ChirpNest-Application” (click on the name) which navigates to the “Devices” tab of that application, select the single existing device just created before (click on the name), switch to the “Activation” tab, enter the following data and click “(Re)activate device”:

- Device address:  
enter the device address of your device (e.g. “26 01 16 83”) or click the round arrow to generate a new device address which must then be configured on the ADT1
- Network session key (LoRaWAN 1.0):  
enter the network session key of your device (e.g. “E6 19 A5 8F 1F 1F 84 F2 C3 0B 2E EA 1E 1F B4 7D”) or click the round arrow to generate a new network session key which must then be configured on the ADT1
- Application session key (LoRaWAN 1.0):  
enter the application session key of your device (e.g. “F5 32 68 0E C0 4C C4 95 7D AA FC 5E 34 CC 6E 35”) or click the round arrow to generate a new application session key which must then be configured on the ADT1
- Uplink frame-counter: leave “0”
- Downlink frame-counter: leave “0”

Now the ADT1 is configured and activated. You can verify by opening the “Device data” tab on your newly created device configuration and then sending a measurement packet (see “Configure ADT1 Device” below). Note that the “Device data” tab must be opened before the measurement is sent.

Last step: Send the information packet from the ADT1 device so the details of the device are stored in ChirpNest (see “Configure ADT1 Device” below). After that, the configuration of ChirpNest is complete.

## Connect KIWI Desktop with ChirpNest

When ChirpNest is set up, KIWI Desktop can simply be connected to ChirpNest by entering the IP address of the LORIX One.

## Configure ADT1 Device

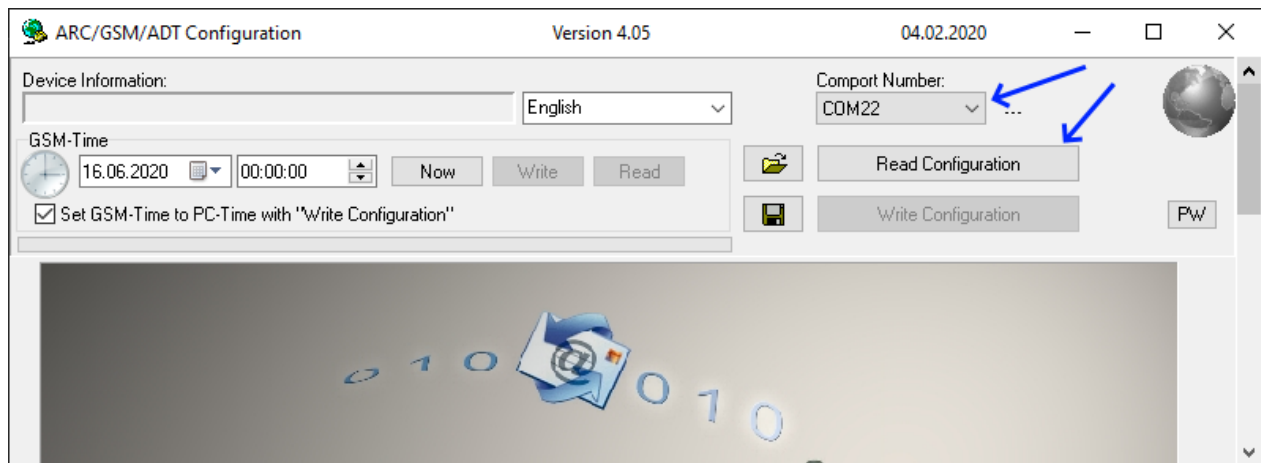
This section explains how the required parameters “device EUI”, “device address”, “network session key” and “application session key” can be read out and/or configured.

The software “GSM setup for remote transmission units” needs to be installed on the Windows computer. It can be downloaded from here:

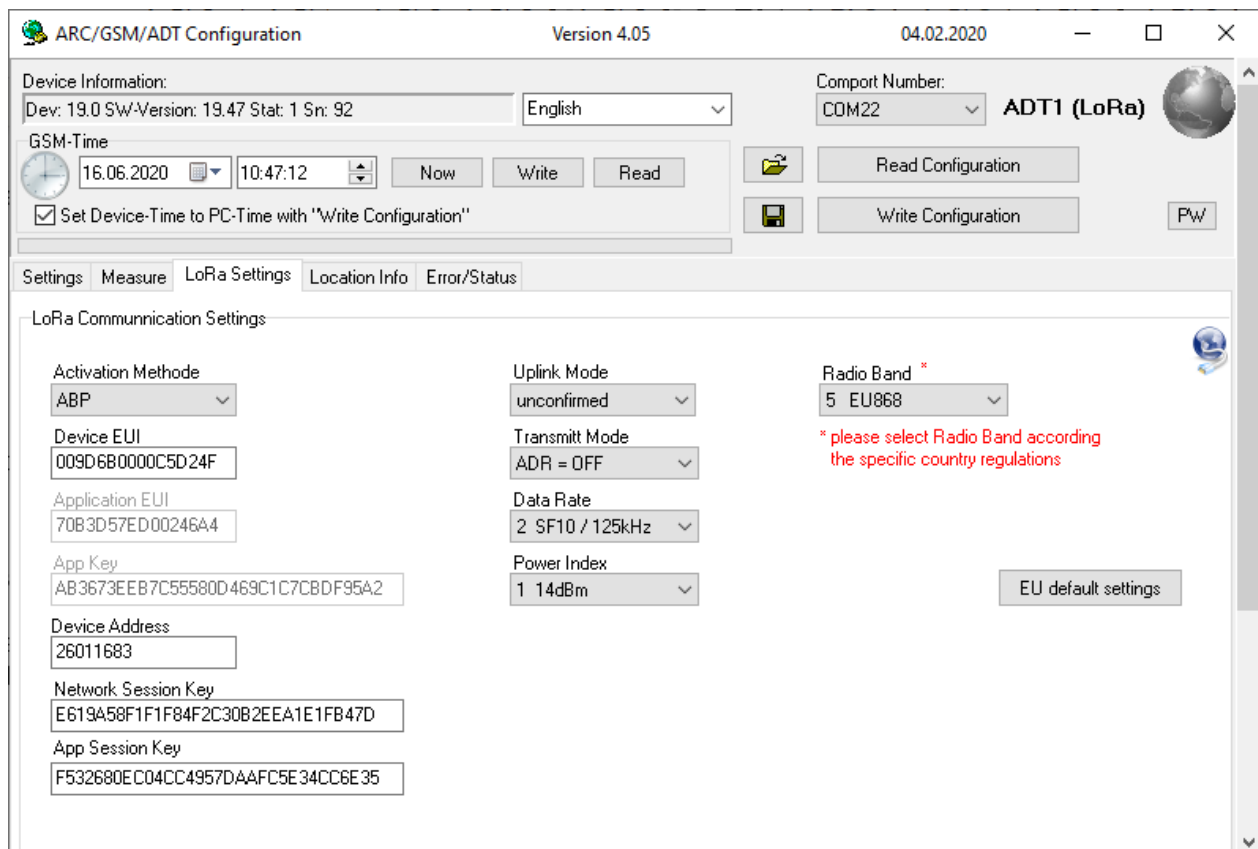
<https://keller-druck.com/en/products/software/desktop-applications/gsm-setup-for-remote-transmission-units>

Connect the ADT1 device to the Windows computer using a Micro USB cable.

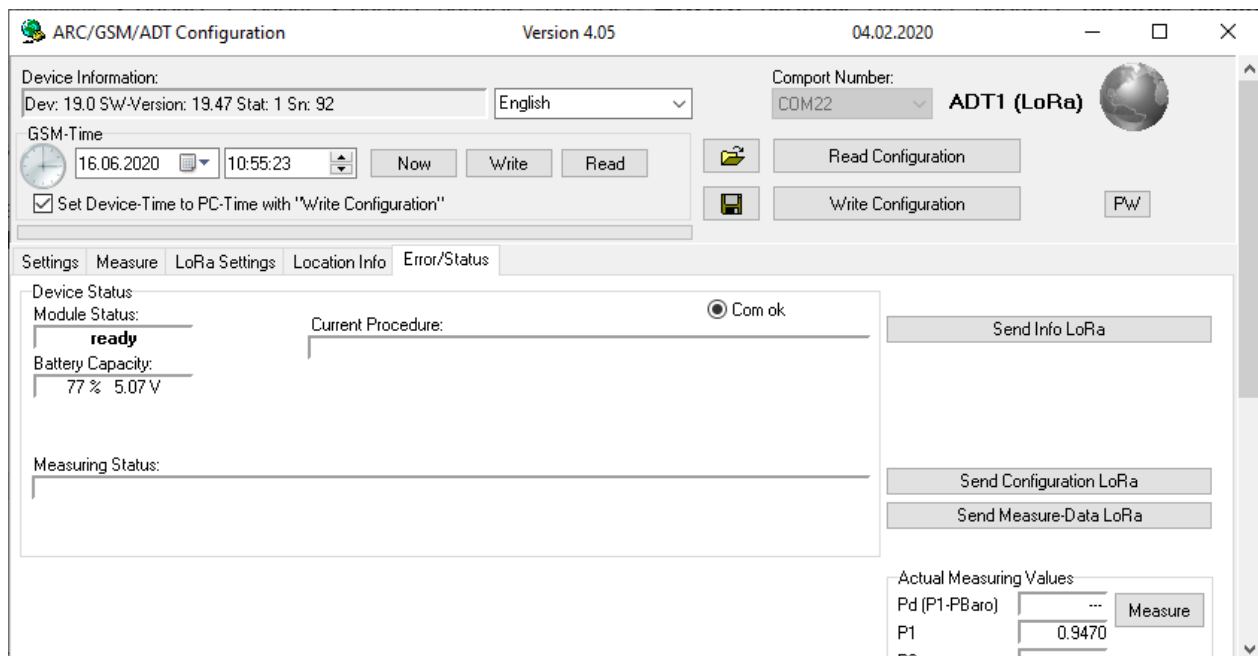
Select the corresponding COM port and press the “Read Configuration” button (see screenshot).



To read out or configure said parameters, switch to the “LoRa Settings” tab. The parameters are then displayed (see screenshot). To configure the parameters, select “ABP” as “Activation Methode” and enter the parameters. Press the “Write Configuration” button after entering the parameters.



To send the information packet from the ADT1 device, switch to the "Error/Status" tab and press the "Send Info LoRa" button (see screenshot).



To manually trigger sending a measurement packet, the button "Send Measure-Data LoRa" can be pressed.

## Decode function for ADT1

The decode function is provided by KELLER in a GitHub repository.

Follow this URL:

<https://github.com/KELLERAGfuerDruckmesstechnik/KellerAgTheThingsNetworkPayloadDecoder>

The actual function is in the file “PayloadDecoderFunction.js”.

Important: ChirpStack required another function signature than The Things Network. For this reason, function `Decoder(bytes, port)` has to be changed to `function Decoder(port, bytes)`.

The decode function can also be copied from here:

```

Entry point of decoding
function decode(bytes) {
    var functionCode = bytes[0];
    var result = {};
    var funcName;
    port.port = functionCode;
    result.bytes = function(bytes) { return pad(bytes.toString(16).toUpperCase(), 2); }.bind("");
};

if (functionCode == 12) {
    // Device sends information package, has to be decoded differently
    result.channelName = information(bytes, result);
} else {
    // Device sends measurement package
    fillUpChannelMeasurements(bytes, result);
}

return result;
}

// Decode the device information package and append it to the result object
function decodeDeviceInfoInformation(bytes, result) {
    result.batteryVoltage = bytes.slice(1, 4).join("");
    result.batteryLevel = bytes.slice(5, 8).join("");
    result.humidity_percent = bytes[9];
    result.temperature = bytes[10];
    result.channel_group_name = bytes.slice(11, 24) + " - " + pad(bytes[11], 2);
    result.temperature_unit = bytes.slice(24, 25) + " - " + pad(bytes[25], 2);
    result.serial_number = bytes.slice(26, 40).join("");
    result.device_model_name = bytes.slice(41, 50).join("");
    result.device_model_name = bytes.slice(51, 60).join("");
}

// Decode the device measurement package and append it to the result object
function decodeMeasurement(bytes, result) {
    result.t = bytes[0];
    result.temperature = bytes.slice(1, 4).join("");
    result.channel = bytes[5].toString(bytes.slice(2, 4).join(""));
    result.channelName = result.channel.match(/(\d+)$/);
    var channelNumber = result.channel.slice(1).replace(".", "");
    result.timestamp = result.channel.slice(1).replace(".", "");
    // Link through all additional package content, every group of 4 bytes
    // contains information for another channel (Channel 1, Channel 2, ...)
    for (var minIndex = 0; result.channelCount > 1; minIndex += 4) {
        var byteIndex = bytes.slice(minIndex, minIndex + 4);
        result.deviceInfo(channelName=result.channel[0], byteOffset=minIndex);
        result.channelName = channelName + result.byteOffset;
    }
}

// Convert an array of bytes into a float: bytesToInt([ 0x3f, 0x2f, 0x6a, 0x1a ]) == 0.8026754500819267
function floatFromBytes(bytes) {
    // JavaScript's Number type can only safely represent numbers up to 2^53
    // So we convert the bytes to a float using the IEEE 754 standard
    var bits = bytes[0] << 24 | bytes[1] << 16 | bytes[2] << 8 | bytes[3];
    var sign = (bits >> 31) << 1;
    var exp = (bits >> 23) << 8;
    var mantissa = (bits >> 12) << 20;
    var float = sign * (1 + mantissa / 16777216) * 2^(exp - 127);
    return float;
}

// Convert an array of bytes into a binary string: bytesToBinaryString([ 0x25, 0x40, 0x6f, 0x6A ]) == "00100101010011010100111110101010"
function binaryFromBytes(bytes) {
    var binary = "";
    for (var i = 0; i < bytes.length; i = i + 1) {
        var bits = bytes[i].toString(2);
        while (bits.length < 8) {
            bits = "0" + bits;
        }
        binary = binary + bits;
    }
    return binary;
}

// Convert an array of bytes into an integer: bytesToInt([ 0x25, 0x40, 0x6f, 0x6A ]) == 625828080
var binaryString = bytesToBinaryString(bytes);
var int = parseInt(binaryString, 2);
return int;
}

// Convert an array of bytes into a UTC date. The bytes array must represent
// the date of record since "2008-02-01T00:00:00Z" bytesToInt([ 0x25, 0x40, 0x6f, 0x6A ]) == 2010-10-11 07:54:50
var unix = new Date("2008-02-01T00:00:00Z").getTime();
var time = bytesToBinaryString(bytes);
var date = new Date(unix + time);
var year = date.getFullYear();
var month = date.getMonth() + 1;
var day = date.getDate();
var hour = date.getHours();
var minute = date.getMinutes();
var second = date.getSeconds();
return year + "-" + month + "-" + day + " " + hour + ":" + minute + ":" + second;
}

// Prepends a number of "0" to a parameter, pad(7, 2) == "07"
function pad(num, size) {
    var s = num + "";
    while (s.length < size) s = "0" + s;
    return s;
}

var WP = {
    1: "P1-P2",
    2: "P2",
    3: "P2",
    4: "P2",
    5: "P2",
    6: "P2",
    7: "P2",
    8: "P2",
    9: "P2",
    10: "P2",
    11: "P2",
    12: "P2",
    13: "P2",
    14: "P2",
    15: "P2",
    16: "P2",
    17: "P2",
    18: "P2",
    19: "P2",
    20: "P2",
    21: "P2",
    22: "P2",
    23: "P2",
    24: "P2",
    25: "P2",
    26: "P2",
    27: "P2",
    28: "P2",
    29: "P2",
    30: "P2",
    31: "P2",
    32: "P2",
    33: "P2",
    34: "P2",
    35: "P2",
    36: "P2",
    37: "P2",
    38: "P2",
    39: "P2",
    40: "P2",
    41: "P2",
    42: "P2",
    43: "P2",
    44: "P2",
    45: "P2",
    46: "P2",
    47: "P2",
    48: "P2",
    49: "P2",
    50: "P2",
    51: "P2",
    52: "P2",
    53: "P2",
    54: "P2",
    55: "P2",
    56: "P2",
    57: "P2",
    58: "P2",
    59: "P2",
    60: "P2",
    61: "P2",
    62: "P2",
    63: "P2",
    64: "P2",
    65: "P2",
    66: "P2",
    67: "P2",
    68: "P2",
    69: "P2",
    70: "P2",
    71: "P2",
    72: "P2",
    73: "P2",
    74: "P2",
    75: "P2",
    76: "P2",
    77: "P2",
    78: "P2",
    79: "P2",
    80: "P2",
    81: "P2",
    82: "P2",
    83: "P2",
    84: "P2",
    85: "P2",
    86: "P2",
    87: "P2",
    88: "P2",
    89: "P2",
    90: "P2",
    91: "P2",
    92: "P2",
    93: "P2",
    94: "P2",
    95: "P2",
    96: "P2",
    97: "P2",
    98: "P2",
    99: "P2",
    100: "P2",
    101: "P2",
    102: "P2",
    103: "P2",
    104: "P2",
    105: "P2",
    106: "P2",
    107: "P2",
    108: "P2",
    109: "P2",
    110: "P2",
    111: "P2",
    112: "P2",
    113: "P2",
    114: "P2",
    115: "P2",
    116: "P2",
    117: "P2",
    118: "P2",
    119: "P2",
    120: "P2",
    121: "P2",
    122: "P2",
    123: "P2",
    124: "P2",
    125: "P2",
    126: "P2",
    127: "P2",
    128: "P2",
    129: "P2",
    130: "P2",
    131: "P2",
    132: "P2",
    133: "P2",
    134: "P2",
    135: "P2",
    136: "P2",
    137: "P2",
    138: "P2",
    139: "P2",
    140: "P2",
    141: "P2",
    142: "P2",
    143: "P2",
    144: "P2",
    145: "P2",
    146: "P2",
    147: "P2",
    148: "P2",
    149: "P2",
    150: "P2",
    151: "P2",
    152: "P2",
    153: "P2",
    154: "P2",
    155: "P2",
    156: "P2",
    157: "P2",
    158: "P2",
    159: "P2",
    160: "P2",
    161: "P2",
    162: "P2",
    163: "P2",
    164: "P2",
    165: "P2",
    166: "P2",
    167: "P2",
    168: "P2",
    169: "P2",
    170: "P2",
    171: "P2",
    172: "P2",
    173: "P2",
    174: "P2",
    175: "P2",
    176: "P2",
    177: "P2",
    178: "P2",
    179: "P2",
    180: "P2",
    181: "P2",
    182: "P2",
    183: "P2",
    184: "P2",
    185: "P2",
    186: "P2",
    187: "P2",
    188: "P2",
    189: "P2",
    190: "P2",
    191: "P2",
    192: "P2",
    193: "P2",
    194: "P2",
    195: "P2",
    196: "P2",
    197: "P2",
    198: "P2",
    199: "P2",
    200: "P2",
    201: "P2",
    202: "P2",
    203: "P2",
    204: "P2",
    205: "P2",
    206: "P2",
    207: "P2",
    208: "P2",
    209: "P2",
    210: "P2",
    211: "P2",
    212: "P2",
    213: "P2",
    214: "P2",
    215: "P2",
    216: "P2",
    217: "P2",
    218: "P2",
    219: "P2",
    220: "P2",
    221: "P2",
    222: "P2",
    223: "P2",
    224: "P2",
    225: "P2",
    226: "P2",
    227: "P2",
    228: "P2",
    229: "P2",
    230: "P2",
    231: "P2",
    232: "P2",
    233: "P2",
    234: "P2",
    235: "P2",
    236: "P2",
    237: "P2",
    238: "P2",
    239: "P2",
    240: "P2",
    241: "P2",
    242: "P2",
    243: "P2",
    244: "P2",
    245: "P2",
    246: "P2",
    247: "P2",
    248: "P2",
    249: "P2",
    250: "P2",
    251: "P2",
    252: "P2",
    253: "P2",
    254: "P2",
    255: "P2",
    256: "P2",
    257: "P2",
    258: "P2",

```

## VII.II. Manual: Development of KIWI Server

The manuals in the development section describe the setup we used to develop said components.

### Install Go on WSL

When using Windows, activate the Windows subsystem for Linux (WSL) ([link](#)) and install the Linux distribution “Ubuntu 20.04 LTS” ([link](#)) and set it up (don’t forget to run `sudo apt update` && `sudo apt upgrade`).

Download the file “[go1.14.4.linux-amd64.tar.gz](https://golang.org/dl/go1.14.4.linux-amd64.tar.gz)” from <https://golang.org/dl/>.

Extract the file to “/usr/local” with the following command:

```
$ sudo tar -C /usr/local -xzf /mnt/c/Users/my_user/path_to_file/go1.14.4.linux-amd64.tar.gz
```

Add the following rows to the file “~/.profile” (e.g. use the command `sudo nano ~/.profile`):

```
# add GOPATH and GOBIN and add GO to the PATH environment variable
export GOPATH=$HOME/go
export GOBIN=$GOPATH/bin
export PATH=$PATH:$GOBIN
export PATH=$PATH:/usr/local/go/bin
```

Leave the command line with the command `exit` and open it again (so a logon is performed).

Just as a note: The official install guide for Go can be found at <https://golang.org/doc/install>.

### Clone the Repository and get started

Clone the repository “kiwi-server” to the local file system (in WSL accessible at /mnt/c/...) using your favourite git tool (note that the git tool should not replace line feeds “LF” with carriage return and line feeds “CRLF”) or the command line as follows:

```
$ git clone URL_TO_GIT_REPOSITORY
```

In case the code is not yet put online and is provided in a ZIP file, copy the content of “kiwi-server” into a folder on the local file system.

Now try to build kiwi-server with the following command (use `sudo apt install make` if make is not yet installed):

```
$ make build
```

Now install the development requirements with the following command:

```
$ make dev-requirements
```

Now create a snapshot with the following command:

```
$ make snapshot
```

The snapshot which runs on the LORIX One (Linux, ARM 7) can be transmitted to the LORIX One using the following command (replace IP\_ADDRESS with the IP address of the LORIX One):

```
$ make copy-to-lorix host=IP_ADDRESS
```



After transmitting it to the LORIX One, it can be started on the LORIX One when connected via SSH (see “Configure ChirpNest via SSH” on page 111) with the following command:

```
$ ~/kiwi-server
```

### Edit the Source Code

To edit the source code we recommend installing Visual Studio Code (<https://code.visualstudio.com/>). Under Windows: use the extension “Remote - WSL” (<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl>) and work remote on the WSL.

Also install the “Go” extension (on WSL when using Windows) (<https://marketplace.visualstudio.com/items?itemName=golang.Go>).

### Generate API Code from .proto Files

When the definition of the API is changed in “measurement.proto” and/or “device.proto”, the server files (“.pb.go” and “.pb.gw.go”) can be generated automatically by performing the following steps.

Install the following packages:

```
$ sudo apt install protobuf-compiler libprotobuf-dev
```

Change the directory as follows:

```
$ cd apidefinition/go/external
```

Run the following commands:

```
$ make requirements
```

```
$ make api-external
```

Now all “.pb.go” and “.pb.gw.go” were generated.

### Unit Tests

To run the unit tests, a correctly configured

```
$ make test
```

If the error message `exec: "gcc": executable file not found in $PATH` appears, run the following command:

```
$ sudo apt install build-essential
```

`make test` should now run the tests, but they fail because a correctly configured PostgreSQL database is required.

To install PostgreSQL locally (recommended), run the following command:

```
$ sudo apt install postgresql postgresql-contrib
```

Then, start the PostgreSQL instance with the following command (might be necessary again after rebooting the system):

```
$ sudo service postgresql start
```

Follow the instruction in the code when searching for “unit test howto: create required database”. When this is done, the unit tests should pass when triggered with `make test`.

### **Add new Version to ChirpNest**

How to create a ChirpNest Image is explained in “VII.III Manual: Create ChirpNest Yocto Image for LORIX One”. This section explains how the source code of the “chirpstack-gateway-os” repository can be updated so a new version of KIWI Server is installed.

First, create a snapshot using `make snapshot` (as explained above). Then, the file “kiwi-server\_v0.0.0-next\_Linux\_armv7.tar.gz” from the “dist” folder needs to be uploaded somewhere where it can be directly accessed through an URL.

Now edit the following file in the “chirpstack-gateway-os”:

```
layers/chirpstack/meta-chirpstack/recipes-chirpstack/kiwi-server/kiwi-server_BETA.bb
```

Replace the URL so it points to the newly uploaded file (search for “.tar.gz”). Also replace the SHA-256 checksum (search for “sha256sum”). The SHA-256 checksum can be determined by the following command (replace `PATH_TO_FILE` with the path to the file ending with “.tar.gz”):

```
$ sha256sum PATH_TO_FILE
```

That's it, now the image can be created as explained in “VII.III Manual: Create ChirpNest Yocto Image for LORIX One”.

### ***VII.III. Manual: Create ChirpNest Yocto Image for LORIX One***

Recommended operating system: Ubuntu 20.04 LTS

In our setup, creating the Image with Docker on Windows did not work and produced an error. For that reason, we built the images in a virtual machine with Ubuntu 20.04 LTS within Hyper-V.

#### **Install Docker**

Check if docker is already installed:

```
$ sudo docker --version
```

Check if docker is running properly:

```
$ sudo docker run hello-world
```

If Docker is installed and running properly, the steps of this section “Install Docker” can be skipped.

To install Docker, the following steps must be performed (source: <https://docs.docker.com/engine/install/ubuntu/>):

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

$ sudo apt-key fingerprint 0EBFCD88
```

The previous command should produce the following output (check if the fingerprint matches):

```
pub  rsa4096 2017-02-22 [SCEA]
     9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid          [ unknown] Docker Release (CE deb) docker@docker.com
sub  rsa4096 2017-02-22 [S]

$ sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

$ sudo docker run hello-world
```

## Install Docker Compose

Check if Docker Compose is already installed:

```
$ docker-compose version
```

If Docker Compose is installed, the steps of this section (“Install Docker Compose”) can be skipped.

To install Docker Compose, the following steps must be performed  
(source: <https://docs.docker.com/compose/install/>):

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose

$ docker-compose version
```

After the setup, a reboot must be performed.

## Create ChirpNest Image

Navigate to the folder where the repository should be created and clone it:

```
$ git clone URL_OF_REPOSITORY_HERE
```

In case the code is not yet put online and is provided in a ZIP file, copy the content of “chirpstack-gateway-os” into a folder on the local file system.

Run the following commands:

```
$ make submodules
```

```
$ make permissions
```

```
$ sudo docker-compose run --rm busybox
```

*(indented commands are to be run within the docker machine)*

```
$ chown 999:999 /build
```

```
$ exit
```

```
$ sudo docker-compose run --rm yocto bash
```

```
$ source oe-init-build-env /build/ /chirpstack-gateway-os/bitbake/
```

The configurations should already be correct. However, if a configuration needs to be edited, now is the moment to do that.

```
$ nano conf/local.conf
```

```
$ nano conf/bblayers.conf
```

Run the following command to start creating the image:

```
$ bitbake chirpstack-gateway-os-full
```

Continue:

```
$ /chirpstack-gateway-os/scripts/chirpstack-prepare-deploy
```

```
$ exit
```

Now, the file ending with `.sdimg.gz` is the image file. It's located here:

```
$ ls -l ./deploy/wifx/lorix-one-512-sd/3.2.0test1/
```