University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design, Implementation, and Evaluation of an Object Tracking Motion Detection System

Jan Meier
Zürich, Switzerland
Student ID: 08-922-858

Supervisor: Sina Rafati, Corinna Schmitt
Date of Submission: May 31, 2017

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

CSG
Department of Informatics
Communication Systems Group

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

This master's thesis documents the design, implementation, and evaluation of Pharos, an Internet of Things (IoT)-based object tracking system (OTS). The OTS also provides a Geofencing-based alarm mechanism, to detect unauthorized displacements of the tracked objects. Pharos consists of hardware devices, the so-called tags, and a web application. The challenges of this project were to create a reliable, secure, and user-friendly system. The tag's two major components are a positioning and a wireless communication module. Since there are multiple feasible solutions for both modules, a detailed comparison of positioning and wireless communication technologies is provided in this master's thesis. These comparisons are then used to justify the LoRa and Global Positioning System (GPS)-based design of the tags. The web app consists of a front-end and a back-end component. The back-end is a Django-based service which implements the data management and logic for the OTS. The Docker virtualization technology is used for the hosting of the back-end. The major design concerns for the back-end were its security and reliability. The front-end of Pharos provides the interface to control the system. The challenges for the front-end were to develop a user-friendly interface which promotes efficiency and a safe usage of the system. The front-end implementation uses the technologies Hypertext Markup Language 5 (HTML5), Cascading Style Sheets 3 (CSS3) and JavaScript. Further, the Vue.js front-end framework was used to make the web page reactive. A Representational State Transfer (REST) Application Programming Interface (API) is used to connect the front-end to the back-end.

Acknowledgments

First of all, I would like to express my sincere gratitude to my supervisors Dr. Corinna Schmitt and Sina Rafati for their support, expertise and time they dedicated to me. I thank Prof. Dr. Burkhard Stiller, the head of the Communication Systems Group at the University of Zurich, for making this thesis possible.

Further, I would like to thank Henry Raymond, who read the whole thesis, and provided lots of valuable feedback concerning different aspects.

Last but not least, I would like to thank Yvonne Moser-Meier and Andreas Meier – my parent – for giving me the opportunity to study and for all their support I received during all these years.

Contents

Chapter 1

Introduction

Every year, there are thousands of unsolved vehicle thefts and burglaries in Switzerland. The Swiss police registered 44900 vehicle thefts in 2015 [1]. Moreover, only 3.5% of these crimes were solved. The number of bicycles among the stolen vehicles is 38322 and their case the success rate is even lower with 2.3%. The statistics on burglary present a similar picture. The Swiss criminal figures report 42416 burglaries in 2015 and a case success rate of only 14.2%. Both, the absolute numbers as well as the high percentages of unsolved cases make it apparent that there is a need for new theft protection measures.

In recent years academia and the press have written intensively about the concept of the Internet of Things (IoT). Also, the industry came up with a wide variety of new products embracing this new trend. The IoT paradigm foresees that the digital world and the objects in the physical world (e.g., lamps, bikes, dishwashers) will merge together. In other words, everyday objects will be connected to the Internet [2]. This development has been made possible by the growing availability and the increasing computational power of Micro Controller Units (MCUs), as well as the steadily improving coverage of wireless communication technologies.

Alarm systems are a good way to protect valuable goods such as artworks, bikes or other vehicles. For example, car alarm systems start sounding as soon as someone heavily touches the car without permission. Home security systems on the other hand, protect houses and the valuable things they contain. These systems monitor the building and automatically inform the police when an unauthorized person approaches or enters the house. However, these systems are not applicable to many objects because as they require too much installation. A lightweight IoT-based object tracking system (OTS) with a theft protection functionality, is a more practical solution for these type of objects. Such an OTS has two major functionalities: First, a continuous tracking of an object's position. Second, informing the user about unauthorized displacements of the object. There are already solutions on the market embracing this approach. TrackR [3], which uses Bluetooth for positioning, and FindMe [4], a tracking solution using Global Positioning System (GPS) and LoRa, are only two of them. So far, there is no predominant combination of communication and positioning technology.

The design and implementation of an IoT-based OTS introduces a variety of challenges. First, the design of such a system has to be done based on an in-depth understanding of the use case to create user acceptance for the product. For instance, the physical device attached to an object has to work without failure in any environment, require little maintenance and has to be affordable as well. Also, the device must be practical, which puts some close restrictions on its size. Therefore, the selection of technology and components used has to be well planned, which necessitates a profound comparison of the available options. The same holds true for the user interface on which the user controls the object tracking system. This has to be easily accessible, easy to understand but yet powerful enough to allow a fine-grained control of the object tracking.

Further challenges of an OTS are privacy and security issues. They are important for every software system but become even more crucial when dealing with ubiquitous IoT devices and positional data. The difference between ubiquitous devices and traditional computers or smartphones is that people do not use them with the same level of awareness [5]. These IoT devices are integrated into a person's everyday life, so the user may forget that they are connected to the Internet. While the owner might be unaware of this, these devices still report environmental or positional data to the cloud. Moreover, if this data is accessed by the wrong person, this must be considered a major violation of the owner's privacy. Especially in the case of positional data which allow inference of a person's most intimate information such as religion or health issues. The security issues of an OTS are therefore important for two reasons: Firstly, an OTS without proper security measures cannot guarantee the user's privacy. Secondly, an insecure OTS cannot reliably track and protect objects, because attackers can easily disable it.

1.1 Contribution

The overall goal of this master's thesis will be to create the concept for an OTS with theft protection functionality, named Pharos, as well as, the implementation of a prototype. The concept for the system is created based on a detailed analysis of the problem. This analysis includes a stakeholder analysis and usage scenarios, as well as an analysis of existing OTS. This analysis of the problem space will be used to derive a list of requirements.

The solution design includes a detailed specification for the OTS's three components, which are the front-end, the back-end, and the IoT device. This design is the blueprint for a prototype implementation. For completeness, the presented design is going to be very extensive. However, some of the identified requirements will not be implemented completely, because their implementation would exceed the scope of this master's thesis (e.g., secure tag registration, gapless application of the presented privacy principles). The developed prototype will be evaluated to show that the implemented solution solves the previously identified challenges.

1.2 Use Case

To provide the reader with a better understanding of Pharos' components and functionality, a schematic use case is illustrated in Figure 1.1. This use case will be elaborated throughout this master's thesis.

Alice is the owner of a bicycle, which she wants to protect from theft. Therefore, she attaches an IoT device to it. This IoT device, also called *tag*, has two functionalities: The first functionality is determining the tag's position and thereby also the position of the bike. The second functionality is to communicate with the back-end and transmit the determined position. This communication has to happen over a wireless channel as the tag will track mobile objects. The front-end of the Pharos OTS is shown on the right-hand side of the figure. Alice uses this front-end to control and configure the tag as well as the back-end. Evil Eve, illustrated on the bottom of the figure, has all sorts of bad intentions. Most of all, she wants to steal Alice's bike. Further, Eve is interested in stealing Alice's private data. Hence, Pharos' goal is to empower Alice to effectively protect her bike while not allowing Eve to do any harm or circumventing the OTS's theft protection functionality.

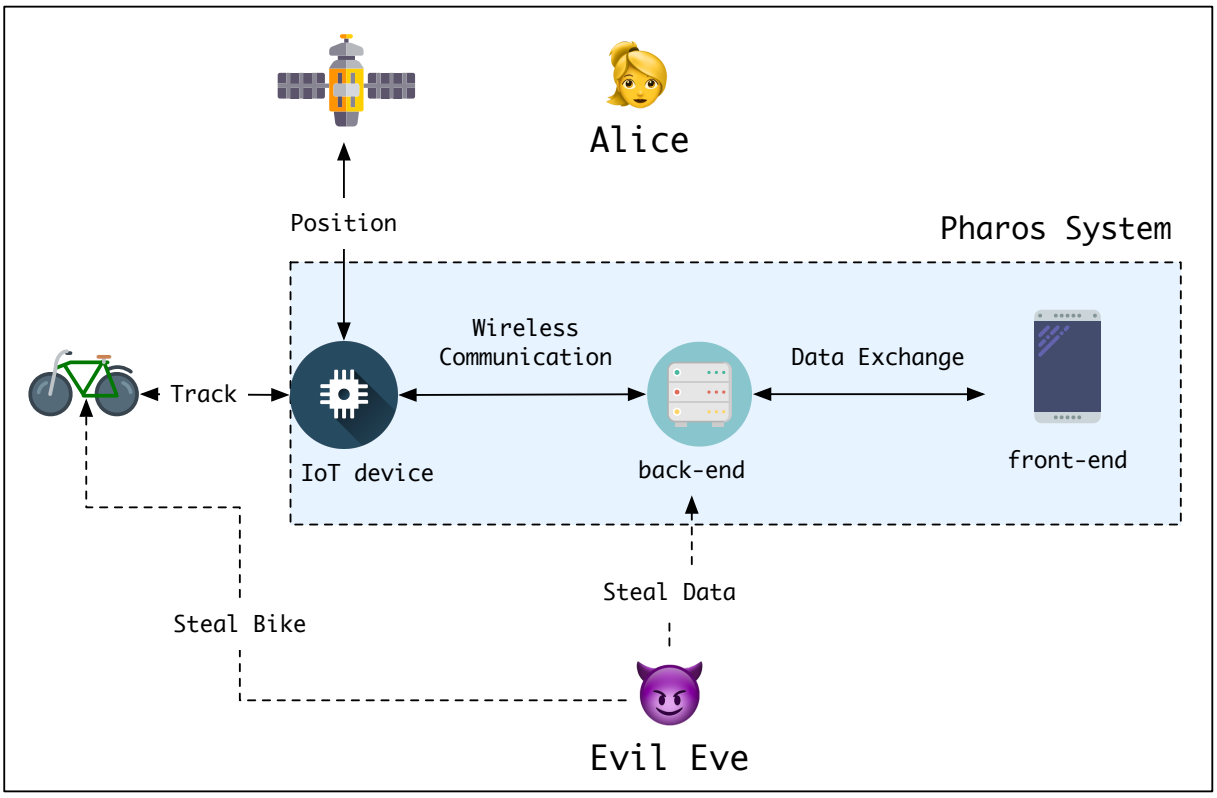


Figure 1.1: A schematic use case of the Pharos' components and functionality.

1.3 Outline

This master's thesis is structured as follows: Chapter 2 provides the theoretical background for the further chapters. Then, Chapter 3 presents an in-depth analysis of the OTS's use case. This analysis is performed with the help of a stakeholder analysis, a user scenario, and the presentation of existing OTSs. Subsequently, in Chapter 4, the key points of the previous analysis will be summarized and transformed into a list of requirements. This list of requirements will then be the basis for the design of the system presented in Chapter 5 and description of the prototype's implementation in Chapter 6. Chapter 7 will deal with the evaluation of the prototype. The master's thesis is concluded in Chapter 8.

Chapter 2

Background

The design and implementation of an energy-efficient, user-friendly, and secure OTS requires knowledge from different areas of computer science. The tag requires positioning and communication technology which is energy-efficient while still allowing for a high quality of service. In the system's data storage and logic component, privacy and security are the major issues. These three topics are of a complex nature and therefore require rigorous scrutiny.

2.1 Technologies

An **MCU** is an integrated circuit with one or several input-output channels. In recent years they have become very popular through the rise of the Arduino platform [6], which simplified the programming workflow and increased the availability. MCUs allow developers to create small electronic devices which are more flexible concerning configuration and customization compared to hard-wired embedded systems. The Arduino product family ranges from tiny, extremely low-power products, to medium-sized ones, with considerable performance.

2.2 Web Technologies

Hyper Text Transfer Protocol (HTTP) is a stateless communication protocol for the web. It was designed to transport Hypertext Markup Language (HTML) documents from a server to a browser, by using a Transport Control Protocol (TCP) connection. HTTP sends data in plain text, and the involved server is not authenticated. These are major threats to the integrity and confidentiality (see Section 2.7.1) of the communication. Hyper Text Transfer Protocol Secure (HTTPS) has been developed to tackle these issues. The confidentiality of the communication is assured by a Secure Sockets Layer (SSL)/Transport Layer Security (TLS) secured connection. A certificate, issued by a trusted certificate authority (CA), guarantees the authenticity of the communication partners [7, 8].

A web application, or in short a **web app**, is a software architecture for an application which runs in a browser. Web apps are built according to the client-server architecture. The server, called the *back-end*, stores the data and runs complex and computationally expensive operations. The client part, called the *front-end*, is implemented as a web page. The front-end handles the user interaction which is then transmitted to the back-end with *HTTP requests*. Traditional web apps send the entire web page content as a pre-rendered HTML file. Hence, a simple change of the displayed content often requires complete retransmission of the file. Newer web apps use a pattern called asynchronous JavaScript and XML (AJAX) for partially updating the front-end web page. This pattern transmits the required data from the back-end to the front-end, where the front-end code re-renders the HTML page to display the newly received data.

Web apps are seldom developed from scratch, but rather based on an existing **web framework**. Web frameworks handle many tasks such as data persistence, HTTP request handling, user authentication, session management and security. Data persistence is required to store data in a database for the long term. Relational databases such as Postgres [9] or MariaDB, are commonly used for that task. The web frameworks often use an Object Relational Model (ORM) to abstract away the database access. ORMs achieve that by creating an object representation for each database entry. Hence, the programmer does not have to deal with database queries and the extraction of values from the query results.

Many different web frameworks are available and used in practice [10]. The Python-based Django [11] framework is a prominent one. It provides all of the previously mentioned features of a web framework. A Django project structures its code around apps. An app is a logical unit or module which encapsulates a set of features to provide a specific functionality. A Django app is stored in a folder, containing all the app-related files.

Virtualization abstracts away a system's underlying hardware-specific implementation. Applications are then developed against a virtual system interface and thus run on any machine supporting the same virtualization method. **Containerization** takes this concept one step further. It involves building images from applications or components of an application which are self-sufficient. Self-sufficient in this context means they carry all the code, the Operating System (OS), the libraries, and configurations they require to run. The developer then creates instances of these images – the instance of an image is called a *container* – and runs them on the virtualized hardware. In general, the containers do not share disk space, memory or computational resources. The communication among the components takes place based on network protocols or explicitly shared disk sections. The advantages of this method are that the images can be developed and run independently of each other. Further, if the code in the images is stateless, horizontal scaling is easily achieved by just adding further containers for the specific image. The newly launched containers then take over the workload the old containers could not handle. A very popular containerization platform is Docker [12].

Application Programming Interface (API) is a set of clearly defined methods, which two application components use to communicate with each other. A **Representational State Transfer (REST)** interface is such an API, commonly used in distributed systems for server-client communication. The essential characteristic of a REST API is that no state is stored. Hence no session is required. The absence of a state simplifies the client-

server coordination as well as the hosting of the API. The server-client communication in a REST API is based on HTTP requests. There are different types of HTTP requests available for different purposes. An overview among them is presented in Table 2.1.

Table 2.1: Available HTTP Methods for REST Services [13, 14]

HTTP	**Description**
`POST`	Create a new data set.
`GET`	Get a list of data sets or detailed information about a data set.
`PUT`	Update or replace data set
`PATCH`	Update or modify a data set.
`DELETE`	Delete a data set.

Given the data types *Car*. A REST API will create an *endpoint* this data type. Each endpoint has its distinct Uniform Resource Locator (URL) such as "`https://www.example.com/cars`". A `GET` request to an endpoint returns a serialized version of an object. Which serialization format is used is left up to the developer but commonly the JavaScript Object Notation (JSON) [15] serialization is used. These serialized objects can then be modified and sent back to the server with a `POST` HTTP request and the changes are adopted on the server side [13, 14].

Nginx is a free-to-use web server published under the Berkeley Software Distribution (BSD) license [16]. Nginx is most often used as a forward proxy which provides a uniform interface for different services behind a single URL. Furthermore, Nginx is capable of handling certificate and encryption tasks around HTTPS and the serving of files.

2.3 Internet of Things

The Internet was engineered in the late 1980's [2]. It was designed as a mean to connect humans, through their computers, to other computers or large mainframe machines. The concept of IoT extends this traditional approach by connecting everyday objects (e.g., lamps, dishwashers) to the World Wide Web. While the basic idea behind IoT is not new, it recently gained a lot of attention from research, industry and the general public.

The term IoT goes back to the late 1990's [17] but the concept of giving everyday objects a digital identifier or connecting them to the internet goes back to the 1980's. Back then, radio-frequency identification (RFID) chips were used for automatic object identification and to track the objects within a warehouse. Later in the 1990's intelligent wireless sensor networks (WSNs) appeared in domains such as health care and industrial facilities.

In the 2000's the concept of IoT as interconnected things, ubiquitous computing, and cyber-physical systems appeared [5].

The term Internet of Things is used widely in computer science and engineering, but no formal and common definition is in place [18, 19]. Those descriptions have the general concept of IoT – connecting devices using the Internet – in common. In this master's thesis the following definition for IoT is applied [5]:

> *" [...]a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'Things' have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network[...] "*

One of the reasons for IoT's increased popularity is the recent advances in some of the key technologies related to it. For example, mobile communication technologies such as cellular and wireless local area network (WLAN) have become widely deployed [2]. Further, some very low power communication technologies such as ZigBee and Long Range Wide Area Network (LoRaWAN) appeared and made the development of tiny, long-living IoT devices possible. Also, the increasing number of user-friendly platforms for integrated computer systems such as Raspberry Pi and Arduino made it possible for commercial developers, as well as private persons, to easily create IoT devices [20].

IoT allows to integrate new aspects of economic and private life into web apps and enable software solutions in domains where it previously was not possible [17]. For example in a hospital, monitoring the patients and assets with IoT devices can make workflows more efficient and enable a higher quality of service. For private persons new applications such as context-aware systems to locate lost or stolen objects are possible. However, this diversity of use cases and domains comes at a price: IoT systems have a highly heterogeneous technology landscape, and a transparent integration of all pieces requires additional effort [2].

New technologies, such as IoT require new conceptual foundations. In the case of IoT systems, there are different architectural frameworks. Many of them follow a Service-oriented architecture (SOA) approach with a varying number of layers. A framework which uses four layers distinguishes between the Sensing Layer, the Network Layer, the Service Layer, and theInterface Layer.

The **Sensing Layer** consists of the physical devices of an IoT system which are attached to the Internet. The Sensing Layer forms a system of interconnected tags or sensors which make it easy to track things or to measure environmental data. There also can be devices in the Sensing Layer which interact with the physical world through actuators.

The **Network Layer** is responsible for connecting the devices of the Sensing Layer to the internet. The connecting of the two layers often involves different communication technologies, mostly wireless ones, to connect different classes of devices. The major challenge in this layer is to find the right communication technology which is both, power saving

and allows a stable connectivity. These are often conflicting goals, which require a well-balanced trade-off. Further challenges are service discovery, data and signal processing, as well as security and privacy issues.

The **Service Layer** contains and orchestrates various services, and integrates them into a functioning system. Some of the Service Layers responsibilities are to deal with the business logic and to store data persistently. A modular structure in this layer is crucial to ensure reusability, and thereby lower costs and a high extendability.

The **Interface Layer** provides a unified interface to interact with the system, often through a web API or a web page. These interfaces allow users and third-party systems to interact with the IoT system. A fundamental functionality of the Interface Layer is to provide a user interface for the end-users. This user interface gives the user the ability to interact with the back-end logic and the IoT devices in the system.

The design challenges of an IoT architectural framework are to integrate various types of devices and systems (such as different internet-enabled objects and a manifold of web services) into a coherent and well-connected system. The design goals of a web framework are to promote extensibility, scalability, modularity, and interoperability among heterogeneous devices. Further, the architecture has to take into account that the devices are highly mobile and often need real-time interaction which makes them different from many traditional software systems [5].

2.4 Positioning Technology

There are many different technologies to determine an object's position, but none of them is suitable for every use case. The most widespread technology is the GPS, which most newer mobile phones use. GPS determines under optimal conditions an object's location with an accuracy of several meters [21]. In urban areas and indoor settings, the precision of the positioning decreases to tens of meters. In underground locations, such as parking lots or cellars, GPS does not work at all. This section presents the most commonly used positioning system. Besides listing their advantages and drawbacks, aspects such as costs and methods used for measuring the position are elaborated[1]. Table 2.2 shows a comparison of the discussed positioning techniques.

Cellular-based positioning systems use the signal of mobile phone networks to determine the position of a device. There are two approaches, both assuming that the location of the cellular network antennas is known. The first approach merely reports to which antenna a device is connected to, which is called a *Cell-of-Origin*. This method is trivial and reliable, but has a low accuracy, since these towers have a range up to 35 km [21]. The second method uses the connection to several cellular antennas and determines the device's position based on the signal strength and *trilateration*. However, the accuracy of this approach lies between 50 and 200 m [22]. The accuracy of both methods strongly depends on the number of reachable antennas. The coverage of this approach is close to gapless in urban areas with many cellular antennas. In rural areas with few to no

[1]For a more detailed description look into the referenced literature for each concept.

antennas, the coverage, as well as the accuracy, is low. Further, the accuracy in indoor settings is worse than outdoors as the walls of the building block the antenna's signal. Connecting to a cellular requires a lot of energy, which makes this method of positioning not energy-efficient. On the plus side, costs of this methods are extremely low. Cellular receivers are cheap, and the network antennas are already deployed.

The **Global Navigation Satellite System (GNSS)** positioning technologies are the most prominent group of positioning systems. The previously mentioned GPS belongs to this class. These systems use positioning techniques that rely on satellites, which orbit around the earth. These systems then use the satellites as reference points for *triangulation*. The GNSS approaches have interesting properties. Most importantly, they have a global coverage and do not rely on local infrastructure. If the GNSS module has good connections to the satellites, the accuracy of these systems is in the range of several meters. However, in urban areas tall buildings often block the signal of the satellites or cause reflection. These effects decrease the accuracy of these systems. In places where the satellites are unreachable (e.g., cellars, parking lots), GNSSs are unusable. GNSSs are rather power-efficient, and the positioning takes a fraction of a second, once the chip knows the position of at least three satellites [21]. Although searching for the satellite can take up to minutes and has a high power consumption.

There are positioning methods which use the communication technologies **Bluetooth** and **WiFi**. These methods rely on *fingerprinting*, which creates an extensive list that maps positions in the area of interest to the signal strength of multiple Bluetooth beacons or WiFi base stations. The current position then is determined by looking up the current signal strengths in the list and choosing the best fit. The accuracy of these methods can be from a few to several meters [22], depending on the density of the collected data. The fundamental drawback of these methods is the need for collecting the data. Also, a good coverage requires many base stations which are costly, requires a time-consuming set up, and must be maintained from regularly. The need for a dense network of base stations limits the availability of these methods significant. Regarding energy-efficiency, Bluetooth and WiFi-based approaches are interesting, because receiving signals does not require much energy. Particularly interesting are approaches which use Bluetooth Low Energy (BLE), a Bluetooth version designed for low-energy consumption.

The very short range **RFID technology** is also used for positioning. These systems rely on many receivers, installed all around the area of interest, which register the proximity of a tag and forward that to a central system. These systems have a Cell-of-Origin accuracy. Due to the need for external base stations, this technology is only suited for indoor settings. It is also quite expensive due to the same reason. The big advantage of this method are the cheap RFID tags which allow tracking a large number objects, while still being affordable. On one side, the tracked objects do not require any power source at all, as RFID tags are powered by signal they received. On the other side, the external hardware does.

Table 2.2: A overview among positioning technologies.

Technology	Accuracy	Infrastructure	Cost	Domain
WiFi	meters	Medium: Several base stations must be installed.	Cheap	Indoor
Bluetooth	meters	Medium: Bluetooth beacons must be distributed.	Cheap	Indoor
RFID	Cell-of-Origin	High: RFID receivers must be installed and set up.	Expensive	Indoor
Cellular	50 - 200 m	Low: Cellular infrastructure is already available.	Cheap	Outdoor (Indoor)
GPS	1 - 5 m	Low: GPS infrastructure is already available.	Cheap	Outdoor (Indoor)

2.5 Wireless Communication Technology

The tags of an OTS require some communication technology. Since the tracked objects can be relocated freely, this technology is bound to be wireless. This Section presents the most common wireless communication technologies on the market. Table 2.3 shows a summary of the presented approaches.

The **IEEE 802.11** standard, also known as **WiFi**, is a very common wireless communication technology. Originally developed for use cases in domestic settings, WiFi has a range of approximately 100 m. The maximum data rate depends on the version of the IEEE 802.11 standard. Currently, throughputs up to 400 Mbps are possible. The IEEE 802.11 standard consumes a considerable amount of energy, as it was intended for devices that can be charged every one to two days, such as smartphones, tablet computers or laptops [23, 24].

When introduced, the **Bluetooth** technology was intended as a replacement for the infrared of remote controls [23]. Bluetooth is highly power-efficient in operation which makes it an attractive option for IoT devices. The introduction of **BLE**, which further reduced the technology's power consumption, made the approach even more promising. With a short range of up to 10 m BLE is most suitable for domestic settings or wearables. The transmission rate of BLE is 1 Mbps. A device with BLE can run $1 - 2$ years on a single charge [24].

The **cellular network** was created for mobile phones. In urban areas, it has a stable connectivity – indoors as well as outdoors. In rural areas, there are regions with connectivity problems. The current cellular standard G4 is not intended for energy-constrained devices and therefore not power-efficient. A further disadvantage of the mobile network is that the telecommunication providers charge for its usage [23].

The in the year 2013 released **LoRa** technology, which belongs to the family of Low Power Wide Area Networks (LPWANs), was designed for highly resources-constrained devices. The small data rate of 0.3 – 50 kbps [23] restricts its usage to use cases where little data must be transmitted. On the other hand, the technology's minimal energy consumption allows running a battery powered device up to several years [23]. The exceptional high range of up to 15km [25], is a further advantage of the LoRa technology. On top of the LoRa physical layer, the open source LoRaWAN stack is being developed. This standard is maintained by the LoRa Alliance, a public consortium of various industry partners. This open standard allows developers to create private or community-driven LoRaWANs. *The Things Network* is such a community-driven WAN provider and has access points all over the world. It can be used for free but with limited air-time.

SigFox is another LPWAN technology, similar to LoRa. It appeared in 2009 and had rapidly increased in popularity since then. This standard allows a range of up to 50 km in rural areas. The protocol and technology are entirely proprietary. Hence, little information is available about SigFox's technical details [25]. SigFox achieves a data rate of 10 – 1000 bps, and the low power consumption also allows devices to run up to several years [25].

ZigBee is a further wireless communication technology, which is designated for domestic sensor networks. It has a range of 100 m and bandwidth of 250 kbps [23]. ZigBee uses a mesh topology which increases the range in areas with many devices. The ZigBee technology is very power-efficient and allows devices to run for months or years [23].

Table 2.3: Comparison of various wireless communication technologies.[25, 23, 24]

Technology	Range	Transmission Rate	Expected Lifetime
WiFi	<100 m	<600 Mpbs	hours
Bluetooth LE	<10 m	1 Mbps	1 – 2 years
Cellular	35 km	35 kbps – 10 Mbps	days
LoRa	2 – 15 km	0.3 – 50 kbps	years
ZigBee	10 – 100 m	20 – 250 kbps	months to years
SigFox	3 – 50k m	10 – 1000 bps	years

2.6 Privacy

Privacy in software systems is a critical issue. Even more so if the system deals with sensitive data, which is the case for an OTS. First, such a system contains a list of a person's potentially expensive belongings. Second, it can store positional data of a person, produced for instance by a tracked bicycle. Not only could this data tell a potential thief where valuable goods are located, but it would also allow him or her to create a movement profile of the owner. This section explains how a system can be engineered to alleviate these privacy issues. The gained knowledge will later be used to design (see Chapter 5) and implement (see Chapter 6) the Pharos OTS with the user's privacy in mind.

The exact meaning of the word **privacy** depends on the context in which it is used. A general definition, according to [26], is: *"The right to select what personal information about me is known to what people."* In the context of an OTS, **information privacy** is most relevant. Information privacy states that the information gathered about a person, and especially sensitive information such as health data or positional data, is under no circumstances publicly accessible. Maintaining information privacy is a non-trivial challenge for information technology (IT) systems as they often store and process data in a distributed manner (e.g., cloud computing, client-server architecture). Due to the importance of privacy, it is also regulated by legislation. These laws there must be met by software vendors [26].

There has been extensive discussion within academia and the general public about the issues of privacy, whereby the positions range from "as little privacy as possible" to "as much privacy as necessary." However, when it comes to sensitive data, most authors and legislation consider privacy as unarguably mandatory [26]. For instance, if health data of everyone would be publicly available, long cured diseases could lower a person's chances of finding a job or health insurance.

The recent success of IoT further increased the importance of information privacy. Traditional computer systems (e.g., desktop computers, laptops) allow a clear separation of when someone is using them and when not. Hence, people know when data might be collected and when not. Ubiquitous IoT devices tear down this separation as users often will not be aware that they are present [20]. Also, these devices will penetrate aspects of life, such as collecting health data or tracking personal habits, which traditional computing do not. A further issue is the resource-constrained nature of IoT devices. These devices often have little disk space which forces them to outsource the collected data to a cloud storage. The external storage of data comprises further threats to the privacy of the stored data.

Software systems are often complex constructs and assuring the privacy of its data is a non-trivial task. *Privacy by Design* has been invented to tackle this problem and is defined as [27]:

> *" Privacy by design is the philosophy of protecting privacy throughout the process of technological development, that is from the conception of a new technology up to its realization. "*

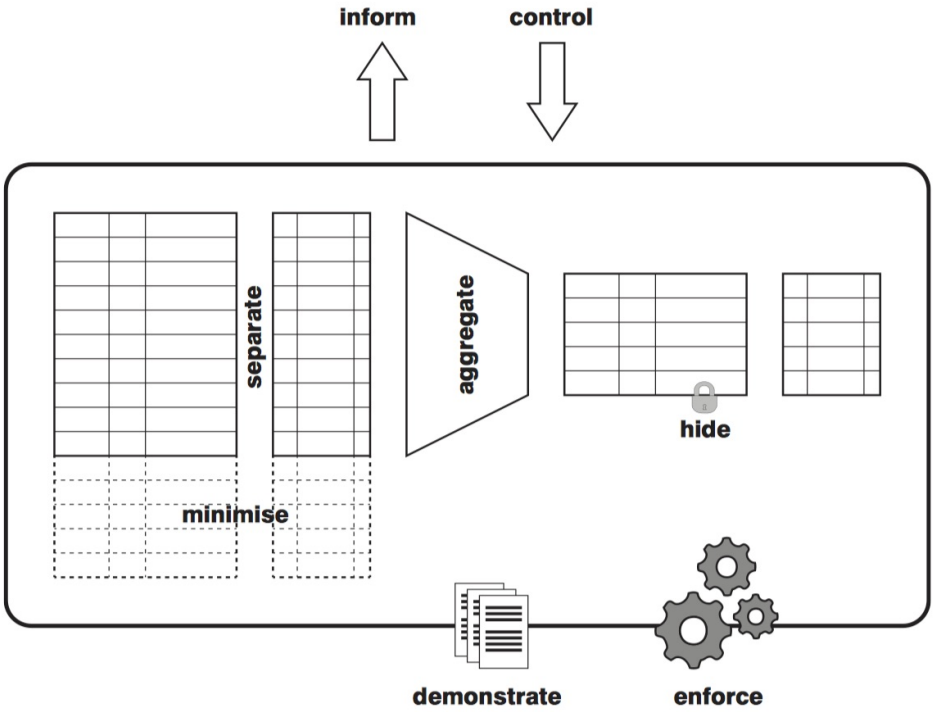


Figure 2.1: Visualization of the privacy by design strategies [27]

The privacy by design approach is based on the assumption that efficient privacy measures cannot be added to software after its implementation. These measures are bound to be an integral part of the system and have to be considered right from the start of the project. There are several different privacy by design frameworks which have a lot in common but differ in the details [27, 28, 29]. This master's thesis follows the approach of Hoepman and Jaap-Henk [27], which presents eight *design strategies*. These design strategies were developed to enforce the seven privacy principles proposed by Cavoukian [29]. Figure 2.1 provides an overview of these eight strategies, which will be elaborated consecutively.

Minimize the amount of collected personal data. Data which does not exist can not be a privacy issue. People also should have the power to decide whether a certain piece of information is gathered or not.

Hide any data, which is gathered or processed from plain view. If the data cannot be seen without the effort of breaking cryptographic procedures, few people or machines will read it, and the chance of abuse decreases. Methods to implement hiding are encryption and mixing data sets so they cannot be mapped to an individual user.

Separate the processing of personal information. This reduces the possibilities of creating complete user profiles. Further, distributed data should be processed locally so that the data exchange is minimized.

Aggregate the data as much as possible to remove unnecessary details. The amount of the user's personal data is thereby reduced, and will no longer pose a privacy threat.

Inform the users adequately about the data, which is collected or processed about them. If the user subject is well informed, he or she can make better-founded decisions. Further, the subject should know why certain data is collected and how it is used. Only with this transparency, can users decide whether or not to share the data.

Control over the data processing should be granted to the user. This control includes the right to view, update, and even delete the data. Further, the user must be made aware of the control options so that he or she can exert them.

Enforce the privacy policies, which in turn have to be compatible with in-place legislation. Further, future changes in legislation must be anticipated, and frequent checks have to be performed to assure that the system still complies with the law.

Demonstrate how the system complies with the privacy policy and any applicable legal requirements. This goes one step further than the enforce strategy by making the compliance provable. The demonstration strategy can be achieved through privacy management system which logs and audits the data access and processing.

2.7 Security

The intention of Pharos is to protect valuable goods. As soon as there is money involved, there will be malicious forces which try to take advantage of it. In the case of Pharos, this means deactivating the tags without permission or cause malfunctions in the back-end to prevent the system from reporting the theft of an object. This section deals with the fundamentals of IT security and the countermeasures a system can implement to assure its security. As for the privacy issues, the presented knowledge will be used to analyze possible use cases for security threats and defining suitable countermeasures in the Pharos' software architecture.

Eckert [30] defines the term *IT Security* as:

> *" The purpose of IT security is to protect an IT-System against damage caused by* ***unauthorized access****,* ***manipulation of data*** *or* ***unavailability****."*

The importance of IT security becomes apparent when looking at recent newspaper articles. The Forbes magazine recently reported the theft of the data of 1.5 billion Yahoo! user accounts [31]. This data included user names, email addresses, poorly protected passwords and plain text security questions. According to the IT security definition by Eckert, this would be an example for unauthorized access. Another issue, which recently fills the headlines, is ransomware, where attackers encrypt a computer's data and then blackmail its owners [32].

This section will first discuss the goals of software security. Further, the Open Web Application Security Project (OWASP) Top 10 list is presented, which lists the 10 most common security threats alongside effective countermeasures.

2.7.1 Goals of Software Security

The goals of software security are manifold, and various definitions of them exist. This master's thesis will follow the goals proposed by [30], which are *authenticity*, *integrity*, *confidentiality*, *availability* and *non-repudiation*.

Software that assures **authenticity** checks that objects and subjects are, what they are expected to be. For example, the system has to assure that an administrator is a real administrator and not just a regular user who pretends to be an administrator. This "pretending someone/something" is made impossible by effective authenticity measures assuring a user's identity. Otherwise, any user and attacker could modify the system and its data as they will. Authentication schemes, such as password/username combinations, are the most common mean to assure authenticity. The Authenticity of transmitted data also has to be assured. Data sent from A to B must not be replaced during transmission. Further, A has to check that B is the true B and not just pretending to be B. Otherwise, false data could enter the system and cause harm.

The **data integrity** goal states that stored and transferred data is not modified without proper permissions. A banking software system, for example, has to assure that a transaction request sent from A to B is not modified during transmission. Otherwise, someone could change the destination of A's transaction, and the software sends the money to the wrong person. Further, if an unauthorized person can manipulate the data in the bank's database, an attack can unrightfully transfer money to his or her account.

The **confidentiality** goal states that a system prevents unauthorized persons from accessing a system's data. This goal also applies to stored and transferred data. In the case of stored data, this means that there is an authorization scheme in place, which regulates who can access which data. Further, the stored data should be hidden from plain view by appropriate encryption schemes. The goal also applies to transmitted data. There, the system has to assure that no attacker can tap the communication channel. The concept of confidentiality is strongly related to the privacy issues discussed in Section 2.6. A system, which does not provide confidentiality, can not provide privacy.

The **availability security** goal states that a system has to be available if authenticated users want to access it. A theft protection system which does not trigger an alarm when an object is stolen is worthless. Also, if the positional data recording suddenly stops and the collected data is lost, the users will no longer trust the system.

A software system, which fulfills the **non-repudiation** goal, assures that actions which took place cannot be denied afterward. Hence, no one can claim or deny to have done something, unless it is true. For instance, this goal prevents that a user denies having placed a bank transaction when he or she has. Another example is, that a doctor cannot delete a prescription from a patient's digital health record, which caused severe complications, without leaving traces and escape possible consequences.

2.7.2 OWASP Top 10

The *OWASP Top 10* is a list of web apps' ten most common security issues. The list is updated from time to time and is created from security reports provided by numerous software companies [33]. The list helps to avoid security risks by explaining the vulnerability and suggesting appropriate countermeasures. This thesis will use these insights for the design of the software (see Chapter 5), as well as for the implementation (see Chapter 6). The individual threats are elaborated below:

Injection: The threat of injection describes the issues which arise when an attacker abuses a system input to insert malicious code into the system. The most common injection attack is the SQL injection. If a user can insert a parameter into the system which then is directly inserted into a SQL query, this parameter can be abused to modify the SQL query. Thereby, the attacker can alter the database in unintended ways and violate the system's data integrity and authenticity goals. Checking every input value for harmful code is the most efficient countermeasure against injection attacks. A suitable API can simplify the checking process.

Broken Authentication and Session Management: Authentication systems require strong authentication mechanisms. Moreover, user credentials must be protected and stored in a secure manner. Further, the granted access – in other words, the session – must not be stolen by adversaries. Login credentials transmitted in plain text are an example of a weak authentication system. An attacker can eavesdrop on the communication and steal them and take over a users' account. A central authentication mechanism, a sophisticated session management, and encrypted communication channels (e.g., HTTPS) are reliable approaches to these issues.

Cross-Site Scripting: A secure web page only runs the scripts that the system transmits to the browser. Cross-Site Scripting (XSS) attacks try to inject code into a form, which the web page later delivers to the other users. This code, which can contain malicious code, then is executed in the browser of other users. Therefore, the system must scan every user input for script content. If such content is detected, the system will escape the code or discard the input.

Insecure Direct Object References: Plain text request parameters, which directly refer to objects, are vulnerable to manipulation. The attacker could change a requests content and thereby fetch objects that he or she might not be eligible to access. There are two methods to prevent these attacks. First, the requests should use indirect object references. A proxy service then translates the indirect references into direct references. The attacker then cannot get access to a specific object, since the reference translation is intransparent to him or her. A second method checks the access rights for every request and rejects invalid requests.

Security Misconfiguration: Poor configurations introduce security issues to otherwise secure software. Issues like outdated software, deactivated features, default passwords or active debug modes are easy targets for adversaries. This kind of security threats is best prevented by keeping the components up-to-date and having a software architecture which isolates the components and makes them well testable. Periodic checks for bad configurations or outdated components are further countermeasures.

Sensitive Data Exposure: Sensitive user data, either stored in the database or transmitted, is a potential security threat when processed with insufficient encryption. Thereby, the confidentiality of the system is violated. The same holds for locally stored data (e.g., cookies, browser cache). Attackers can get hold of this information and misuse it. Examples for insufficient encryption are poorly applied algorithms, outdated algorithms or missing security headers when dealing with HTTP requests. Also, unencrypted backups of former encrypted data make the initial encryption useless. The identification of sensitive data items is the first and most important countermeasure. Minimizing the amount of collected data is the second one. Nonexistent data is neither stored nor transmitted, and therefore not exposed at all. A third countermeasure is a periodical review of the used cryptographic algorithms. This review aims to identify outdated principles or to spot implementations with known vulnerabilities.

Missing Function Level Access Control: An application should prevent users from invoking actions or accessing data without the proper permissions. Further, the system must make sure that the mentioned actions are not visible to these users. A system failing to do so suffers from missing function level access control. This is different from a general access control, where unauthorized users can not use the system at all.
An access right management with restrictive default values is a effective countermeasure against this vulnerability. The access right management assures that users only access data items and functionalities if they have the proper privileges. Restrictive default values avoid that privileges are granted by accident. Also, the system must check the access rights multiple times in case of a multi-step procedure. Otherwise, the procedure might be attacked between the steps.

Cross-Site Request Forgery: A authenticated user can send any request to a server. An attacker can make use of that and send a request on behalf of the user. For example, a user has logged into his banking account in the browser window A. In a browser window B the user is browsing a different web page. Now, code from that browser window B sends a request to the banking system. Since the user is logged into his bank account in window A, the banking software considers the user logged in and accepts the request. Cross-Site Request Forgery (CSRF) tokens help against this vulnerability. The server randomly generates these tokens and sends them to the clients. From then on the server expects every request from the client to contain this token, to assure that the request comes from the right process. In the previous example, the banking system no longer accepts the request from window B, as this request does not contain the token.

Using Components with Known Vulnerabilities: Every software has failures. This also holds for libraries and third-party software components. Using such a component will introduce the failure to the developed software and thereby introduce a vulnerability. Therefore, the developer has to assure that the used components are up-to-date and do not suffer from known vulnerabilities. A list of all the components a software uses is a suitable tool to tackle this problem. This list then must be consulted on a regular basis, and every component should be checked for updates and newly found vulnerabilities.

Unvalidated Redirects and Forwards: If the web page redirects the user, an attacker might abuse this functionality and guide the user to malicious web pages. Alternatively, if the web page internally relies on redirects, attackers might circumvent authorization by abusing these redirects.
Entirely avoiding redirects is the simplest countermeasure. If this is not feasible, redirects should avoid using URLs to pass the parameters for the redirect. Further, the server must always check if a user has the privileges to visit a specific link.

2.7.3 Two-Factor Authentication

Authentication works usually based on username/password scheme. These schemes are vulnerable to many sorts of attacks, where an attacker steals a user's username and password combination [34]. Unencrypted transmission of login parameters or badly protected user databases are among the most common sources of these attacks, but there are others. **Multi-factor authentication** reduces that risk by adding a second security component. Most systems are a so-called "have-something" approach, where the user has to prove that she or he possesses something. A common approach uses smartphones. There, after the initial username/password verification succeeded, a message with a secret code is sent to the device. The user then must enter this code into the login form, and thereby proves his or her possession of the phone. An attacker can now no longer successfully authenticate himself or herself.

Chapter 3

Problem Analysis

This chapter analyzes the problem space of Pharos. First, related products and work from industry as well as academia are listed. Second, a stakeholder analysis is conducted which analyzes future stakeholders and their importance for the project. Third, two possible use cases are presented to derive a list of required features and user expectations.

3.1 Tracking Solutions

There is a large number of different OTS systems available. These systems range from small, easy to use Bluetooth trackers, to sophisticated tracking solutions, which require an extensive setup. OTS are seldom discussed in academia, and only very little work on this topic can be found. This section first highlights different approaches of commercial OTS before taking a look at the academic literature on OTS.

3.1.1 Bluetooth Tracker

Bluetooth Trackers are small devices which are used to locate and track personal belongings such as keys or handbags [35]. The Bluetooth technology is used for locating the objects and communicating the evaluated position. The low energy consumption of Bluetooth tracker, especially of such trackers using BLE, allow a compact design and long lifetime. A selection of such trackers is presented in Figure 3.1.

Bluetooth-based trackers allow users to locate things within the range of the Bluetooth technology and are therefore best suited for indoor settings. The Bluetooth trackers are mostly used to remind the users if a tracked object moves out of range. Thereby they prevent the user from accidentally leaving their bag or wallet in a restaurant or at home when going outdoors. All of the presented products shown in Figure 3.1 are built around this functionality.

(a) Lapa finder (b) Pebbelbee (c) Tile (d) TrackR

Figure 3.1: A selection of Bluetooth-based tracking devices. (cf. references Table 3.1).

Due to the limited range of Bluetooth, these trackers always require an assistant device such as a smartphone. To overcome this limitation the TrackR [3] tag and the Lapa Bluetooth finder [36] implement a crowdsourcing approach. All phones with the corresponding app installed constantly scan the environment for tags, and report their location to the cloud.

The big advantage of Bluetooth-trackers is their power-efficiency and the resulting long lifetime. Further, they have the advantage that no base stations have to be installed, and no time-consuming setup is required. Their drawbacks are the low accuracy, dependence on other users having the app installed and short communication range. Further, a test by the heise magazine has shown [35] that most of the tags work unreliably. In the test, the tags' positions were not reported properly, even though there were devices around with the app installed.

Table 3.1: A selection of finder products

Name	Link
Nut Find 3	`https://www.nutfind.com/` (accessed: 23.5.2017)
Lapa	`https://findlapa.com/` (accessed: 23.5.2017)
Pebbelbee	`https://pebblebee.com/` (accessed: 23.5.2017)
musegear finder	`http://www.musegear-finder.net/` (accessed: 23.5.2017)
Tile	`https://www.thetileapp.com/` (accessed: 23.5.2017)
TrackR	`https://www.thetrackr.com/` (accessed: 23.5.2017)
Chipolo	`https://chipolo.net/` (accessed: 23.5.2017)
keeper	`http://www.thekeeperapp.com` (accessed: 23.5.2017)

Table 3.2: A GPS based tracking solutions

Name	Link
iota Tracker	`http://iotatracker.com/` (accessed: 23.5.2017)
fineMe	`http://www.findme-tracker.ch/` (accessed: 23.5.2017)
itraq	`https://www.itraq.com/` (accessed: 23.5.2017)
whistel	`http://www.pettracker.com/` (accessed: 23.5.2017)
Tractive	`https://tractive.com` (accessed: 23.5.2017)

3.1.2 GPS Tracker

Besides the Bluetooth-based trackers, there is a second big group: the GPS-based trackers. These trackers are mostly intended to keep an eye on older adults suffering form dementia and children. Because the GPS technology is not restricted by the requirement of a base station they can track objects almost everywhere. These trackers are often combined with a long-range communication technology, such as the cellular network, therefore work almost everywhere. Selected trackers are introduced below and more can be found in Table 3.2 including links for further information.

The drawback of the described devices is that the cellular technology is power-intensive and therefore the devices have a short battery lifetime. Some of the GPS trackers compensate for that by integrating bigger battery modules which lead to bulky hardware. Therefore, some devices use alternative communication technologies (e.g., Bluetooth, LoRa) to overcome the mentioned issues.

FindMe [4] is a Swiss product, developed by the Mobiliar insurance. It is a credit-card-sized device, which communicates through the LoRa technology. The producers advertise that the tracker runs for two years without recharging. Unfortunately, the product's homepage provides little information, and the product is not yet available.

Itraq [37] is a GPS-based tracker which uses a cellular connection. Depending on the device settings the battery works up to three months without charge. The first version of the credit-card-shaped device can be ordered for USD 129 The annual fee for the data plan is USD 59. Version 3 of the device can be pre-ordered and offers a variety of new and improved features. The device is designed to track objects and people. Through the app, the customer can parametrize the device to fit the specific use case. Some of the app's features include setting report interval, sharing the tag with other people, live tracking, activating and deactivating, geofencing and switching between GPS location or cellular location.

Ping[38] is another GPS and cellular-based tracking device. It has been funded through IndieGoGo [1] . The tag runs three months on a single charge. The device supports further positioning technologies (e.g., Bluetooth). The device's size is only 34×34 mm, what is smaller than most competitors.

3.1.3 Business Solutions

The last category of commercial OTSs are the business solutions. These solutions often combine several different technologies into a single product. The business domains, for which these OTS are intended for, are diverse and range from the heath sector, over security companies to the mining industry. These OTS mostly have a distinct tracking solution for indoor settings and another for outdoor settings. GPS is predominantly used one in the outdoor settings. The technologies used indoors are more diverse. Most commonly used are BLE, RFID, and WiFi. However, there are also approaches which use GPS signal repeaters. Table 3.3 presents a selection of commercial OTS.

Lighthouse.io is one of these business solutions and is designed for the health care, cleaning, and mining domains. Its indoor positioning technique is based on Bluetooth fingerprinting. Besides plain object tracking, lighthouse.io offers extended features such as live asset tracking, productivity measurements, alert management, incident reporting and exception tracking.

Table 3.3: Business tracking solutions

Name	Link
insoft	`https://infsoft.com` (accessed: 23.5.2017)
lighthouse.io	`http://www.findme-tracker.ch/` (accessed: 23.5.2017)
nextome	`https://www.nextome.net` (accessed: 23.5.2017)
GeoMoby	`https://www.geomoby.com/` (accessed: 23.5.2017)
BEACONinside	`https://www.beaconinside.com/` (accessed: 23.5.2017)
Inside GPS	`http://www.insitegps.com/` (accessed: 23.5.2017)

[1] `https://indiegogo.com` (accessed: 12.4.2017)

3.1.4 Research

There is little academic literature about OTS. Most documents focus on the technical aspects of object tracking and do not present complete systems.

The paper "Anti-Loss Key Tag Using Bluetooth Smart" presents a Bluetooth-based loss protection system [39], though the paper mainly focusses on the implementation's technical aspects. The proposed system uses a Bluetooth device, which relies on the communication technology of a close-by smartphone. The range of this tracking solution is therefore limited. The paper neither discusses security considerations, privacy aspects nor thoughts regarding the solution's usability.

Ramani et al. [40] present the concept for a GPS-based theft-protection system for cars, which includes a tracking functionality. The paper provides extensive information regarding the chip design and the used technologies. The software and user interface of the project are discussed in-depth.

A mobile-phone-based solution for car tracking and theft-protection is presented by [41]. It is based on an Android app which uses GPS for positioning. This paper briefly discusses the system's architecture as well as the user interface. The resulting tracker is costly and is comparably large since it is based on a fully functional smartphone.

BikeTrack [42] is a Bluetooth-based solution for tracking stolen bicycles. The presented approach uses Bluetooth positioning in combination with a smartphone app. The collected positional data is stored on a centralized server, and the user interface uses Google Maps for visualization. Security and privacy issues are not addressed by the paper.

3.2 Stakeholder Analysis

The *stakeholders* of a software project are persons and organizations which have some level of interest in the project [43]. A *stakeholder analysis* systematically lists these stakeholders along with their interests in the project. This analysis consists of three steps. The first step is to identify the involved stakeholders. The second is to analyze the stakeholder's vision and expectations concerning the software product. The third step then determines the individual stakeholder's importance for a successful software project. The results of the firsts two steps are presented in this section. The result of the third steps is deferred to Chapter 4. Personas are used as an elicitation technique. The concept of personas is described by [44] as *"the envisionment and elaboration of a hypothetical target user with his or his personality, needs, and preferences"*. Their role during software development is characterized by [44] in the following way:

> *"Personas are often developed during the problem analysis or requirements specification phase of a project, as a means for understanding, expressing, and working with the goals and implied requirements of different target users."*

Having real people for a stakeholder analysis would be preferable but due to the limited scope of this master's thesis personas were the only practicable option.

The **private customer** wants to use the theft protection system mostly in a domestic setting and for few items, such as a bicycle or a car, therefore also outdoors. The private customer might live in densely populated areas or rural areas, with sparse population. While traveling, the private customer might take his or her belongings to distant places. In general, the private customer only wants to track his or her most valuable possessions. The private customer favors a light-weight system which is easy to understand and set up. The entire system, including tags and possible base stations, should be installable within a few hours. Further, the set up process should be manageable without a profound technical understanding and thus without the help of a technician. The system should require as little maintenance as possible.
The private customer is considered to be highly price-sensitive. Hence, he or she are not willing to buy an expensive system. The user interface of the product and the tag handling has to be easy to understand and allow an effective workflow for the most common activities. Also, the user interface has run fast and long loading times are not acceptable. The private customer demands a reliable system which will capture all possible thefts and allow a gapless tracking of his or her objects.
Private users are sensitive to security and privacy issues. Privacy is inevitable to protect personal data such as information about the personal belongings or the positional data which could be used to create movement profiles. Also, the list of valuable goods, should not be accessible to potential burglars, as it might lead them to favor the theft of specific items. The security of the system is considered mandatory by the user because otherwise neither the data privacy nor the reliability of the system will be granted.

Business customers share a lot of the private customer's expectation, but there are also significant differences. Business customers will more likely use the theft protection system in crowded areas and indoor settings.An exhibition hall is a good example for a business customer. Such a hall has many, expensive exhibits. Therefore, the management wants a highly reliable solution, which is suited to track many objects at once.
Business customers are price-sensitive but also have the financial means to make an investment. A lightweight system is favored by the business customer as well. However, the exhibition hall already has technical installations, and therefore a more complex tracking technology setup is feasible. Maintenance costs are also more acceptable to business customers.
Regarding privacy and security business customers are even more sensitive than private persons. The tracked objects might be highly expensive, making security issues which affect the dependability of the OTS unacceptable. The larger amount of tracked objects requires an efficient way to use the interface. Further, the tool's interface should allow the integration into existing workflows of the organization.
In the business context, the OTS will be used by many different employees, and they will change regularly. Therefore, the system should be easy to learn and use, to keep the training costs for the new staff low. Since some of the employees might work for contractors, the business customer wants to share the tag with them, but only with restricted access rights.

Delegates are persons who use the OTS on behalf of someone else. For example, if someone goes on vacation and a neighbor takes care of someone's belongings. These delegates should be able to use the system but only to a limited extent. For example, the delegate can disable or accept alarms, but cannot see the positional data of a tag, which has been recorded in the past. Delegates will work with the OTS system on an infrequent basis. Therefore, they do not have much experience and will appreciate a self-explanatory interface, with intuitive work flows.

The **police** will get involved when a tracked object is stolen. In such a case, the police want to know where it is. Also, previous positions are of interest to the, as the allow to create a movement profile of the thief. The police want the OTS to report thefts as soon as possible, as this increases the chance of catching the thief. In the best case, the OTS automatically sends a notification as soon as the alarm is confirmed. Once an object is stolen, the police is interested in the most recent position of the stolen object. Therefore, the OTS should have a high update rate of the tracked object's position.

System administrators are interested in a system which is easy to maintain. Therefore, the system should be scalable, not error prone, protected against attacks, and easy to deploy. If the workload of the system suddenly increases, the system administrator should be capable of increasing the available resources as fast as possible. Errors should be avoided by all means because they cause downtime which is not tolerable. Thus the system should be well tested. Further, the system should be well documented such that the administrator can fix bugs efficiently.

The **Developer** wants an easily extendable and maintainable system. A system is extendable if it has a proper architecture created with extensibility in mind. Such a system is also easier to understand when it has a modular design and a clear separation of responsibilities. A well maintainable system also has to offer thoughtfully designed APIs and other interfaces.

3.3 User Scenario

The previous section presented a stakeholder analysis for an OTS. This section now presents a specific use case for such a system. The method utilized for this use case is borrowed from "Scenarios-Based design" by John M. Caroll [45].

"Scenario-Based Design" is a user-centered design method, which aims to "exploit the complexity and fluidity of design by trying to learn more about the structure and dynamics of the problem domain" [45]. The method achieves this goal by writing scenarios which describe how the future system will be used by an end user. This method is based on the assumption that software systems not only provide functionality but also are embedded into existing environments and routines.

A scenario describes the way a system will be used as a story, written in natural language. This non-technical approach has several advantages over more technical ones, such as creating a plain list of features. First, the scenario-based approach allows specifying a system's functionality within a few words. The level of detail can be adjusted on demand

but is usually kept low. The fact the scenario describes a system in easy-to-understand language and as a non-technical document, allows the software architects discuss on the system's functionality. The scenario thereby almost works like an early prototype of the system.

The scenario presented for the use case of an OTS will pick up the story presented in the introduction of this thesis (see Chapter 1).

3.3.1 Alice's Bike

Someone recently stole Alice's bike. Now Alice has bought a new one and wants to protect it with an object tracking system. Alice goes to the store and buys the product which appears to be the most trustworthy while still being affordable.

As Alice arrives home, she opens the package and is surprised that there is no thick manual booklet. The booklet refers her to a web page which guides her through the entire set up process. At first, she is asked to create an account which is surprisingly easy. Within a few steps, she has entered her credentials and provided additional information.

Next, the web page requests Alice to register the tag. Alice goes to her bike and attaches the small tag bellow the saddle. Next, she presses the "Add Tag" button on the web page and registers the tag. She is surprised how easy it was to connect the tag to the system. She only had to press a button on the device and enter the serial number printed on the manual.

Once Alice has registered the tag, she wants to activate the alarm so that no one can steal her bike anymore without her noticing. At first, she enters the current position of the bike and how far the bike can be moved before triggering an alarm. She is pleased that she can define a region since she has to relocate her bike from time to time within her apartment. If she had to deactivate the alarm every time, this would be a nuisance. Next, Alice enters how much time she has to deactivate a false alarm. To finally activate the alarm, she presses the corresponding toggle. After a while, the system informs her that the tag has a weak positioning signal. The weak signal reduces the accuracy of the tracking and delays the alarm triggering. An additional indoor tracking solution could solve this problem. Since she is not willing to spend additional money, she ignores that message.

The following week Alice wants to visit a friend by bike. Thus, she first has to deactivate the theft-protection on her bike. Unfortunately, she forgot her login credentials. To her delight, there is an easy way to reset her password. Once logged in, she can deactivate the theft-protection within two clicks. Further, she is notified that the tag still is fully charged and runs correctly.

Some weeks later Alice goes on vacation. Since Alice will not have access to the Internet all the time, she decides to give access to her neighbour to the theft-protection system. Her neighbor is now capable of dealing with her bike's alarm. In the case of a theft, she also gets access to the positional data of the bike. However, she cannot see where Alice

has been in the past. The neighbor only has access to the positional data recorded during the period of her responsibility.

After a few days, the neighbor is looking after Alice's apartment and pets. Thoughtless the neighbor moves away Alice's bike out of the already set region, and immediately an alarm pops up on her smartphone. The neighbor logs into the web page right away and disables the alarm.

As Alice had no access to the Internet all day long, the system her informs her in the evening that there has been an alarm. She logs into her account, and a notification tells her what happened. She is relieved that it was a false alarm and nothing happened.

Chapter 4

Findings and Requirements

Chapter 3 analyzed the use case of an OTS. This Chapter first summarizes the findings from the previous chapter's analysis and then deduces a list of requirements from that summary.

4.1 Findings Concerning Existing Solutions

Section 3.1 presented commercial OTS and some academic work about this topic. Most of the presented products are either Bluetooth-based trackers or trackers which use a combination of GPS and cellular. The following issues with the existing solutions have been identified: 1) The Bluetooth trackers have a low accuracy. Further, they require an additional device (e.g. smartphone, base station) for the transmission of the position. 2) GPS-based trackers mostly use the cellular network for the communication. This technology is energy-consuming and the lifetime of the trackers is therefore rather short. Further, the usage of the cellular network causes monthly fees. 3) The GPS/LoRa trackers mostly overcome the issues of short lifetime and monthly fees. However, most of them are in an early project state and thus not yet available. 4) The scientific literature about trackers presents approaches for the tracking hardware. However, most literature does not mention considerations regarding the OTS's back-end software and infrastructure or discuss the issues of security, privacy, and usability.

4.2 Findings Concerning Stakeholder's Expectations

Section 3.2 described the stakeholders which are relevant for the development of an OTS. Table 4.1 summarizes the findings of the stakeholder analysis and lists the stakeholders expectations. The table further describes the stakeholder's role in the project and how important they are to develop a successful object tracking solution.

Table 4.1: Stakeholder Analysis

Name	Area of Interest	Expectations	Role
Private Customer	Application	• Use OTS in urban, rural, and remote areas • Manages small amount of tags • OTS is light-weight and "non-technical" • OTS requires little maintenance • OTS is affordable • OTS has an intuitive and effective user interface (UI) • The interface has a low latency • OTS has high availability • OTS handles privacy issues carefully • OTS handles security issues carefully • Tags can be shared with other users	Uses the OTS on a regular basis and is therefore **highly relevant** for the product.
Business Customer	Application	• Track objects indoors • Track many objects at once • OTS is lightweight and requires few set up • OTS requires little maintenance • OTS is affordable • OTS has an intuitive and effective UI • OTS has low latency • OTS has maximal availability • OTS handles privacy issues carefully • OTS handles security issues carefully • OTS allows tag sharing with sophisticated access right management	The business customers work with the system on a daily basis what makes them **highly relevant** for the product.
Delegate	Application	• OTS's UI is self-explanatory and the workflows are easy to learn. • OTS's UI remains comprehensible despite restricted access rights	The delegates use the system very infrequently and have therefore a **low relevance** for the product.

System Administrator	Maintenance	• OTS has well-documented code and a comprehensive user manual • OTS the software's reliability is assured through extensive testing • The OTS scales well in terms of users and tracked objects	The system administrator works which the system on a daily basis and is therefore of **high importance** for the project.
Developer	Development	• OTS has well documented code • OTS has a easily extendable and therefore modular system architecture • OTS has code which can be well maintained	The software developer works with the system from on a daily basis to time and is therefore of **high importance** for the project.
Police	Data	• OTS reports thefts as soon as possible • OTS reports the objects positions with low latency • OTS provides the previous positions of a tracked object	The police works only very sporadicly with the product and is therefore of **low relevance**.

4.3 Findings Concerning Selected User Scenario

Section 3.3 presented two scenarios how personas will use the Pharos object tracking system. This section now will list the key features and expectations mentioned there.

Table 4.2: Key features from the scenario presented in Section 3.3

Feature	Description
Affordable price	Alice is a price sensitive customer and wants a low-cost product.
Professional appearance	Alice wants a product which looks professional.
Self-explanatory UI	Alice appreciates the self-explanatory UI which guides her through the essential functionalities.
Simple registration process	Alice has a fast success experience by creating a account in short time.

Simple setup of user details	Alice can easily enter the information for her account.
Simple tag set up	Alice has a fast success experienced because she can register her first tag fast.
Small tag	Alice appreciates the small tags which she can easily attach under the bike's saddle.
Location based alarm conditions	Alice can set the alarm conditions based on the position of her bike.
Reset of password	Alice can reset the forgotten account password.
Fast tag management	Alice can enable/disable the tag fast.
Information about signal quality	The system informs Alice about the tag's signal quality.
Transparent support for multiple technologies	Alice can extend the OTS with more positioning and communication technology to improve the signal quality indoors.
Display of power level	The OTS informs Alice about the power level of her tags.
Tag sharing	Alice can share tags with other users.
Alarm handling	Alice can cancel false alarms in an efficient manner.
Activity transparency	The OTS informs Alice about activities (e.g., alarms, configuration changes) related to her tags.

4.4 Resulting Requirements For Pharos

The previous three sections summarized what the stakeholders expect from an OTS and listed the key insights from the user scenarios. This section now provides a detailed list of requirements for the prototype developed for this master's thesis. The list is structured around the four layers of an SOA for IoT as presented in 2.3. The requirements are further classified into one of three categories. The *Features* category are functionalities which the system offers to a user or another component of the system. This category includes statements about what the system does but not elaborate how. The *Quality Attributes* category complements the features by defining how or under which circumstances the

functionalities must be provided [43]. The *Usability* requirements build the third class of requirements. The usability requirements describe how a user would like to perform a certain action. The usability requirements were introduced for two reasons. First, the user satisfaction was identified as a key issue in Section 3.2 to the success of a commercial object tracking solution. Second, it avoids that the technical requirements of the quality assurances class are mixed with human-centric one. The Service Layer does not have usability characteristics, because end-users will not directly access it.

Table 4.3: Requirements for the Service Layer

Domain	**ID**	**Name**	**Description**
Features	**F5**	Data Storage	The Service Layer offers a data storage to store the positional data persistently.
	F6	Multiple sensor types	Multiple types of sensors can be connected to the Service Layer
	F7	Front-end API	The Service Layer offers an API to the Network Layer and to the Interface Layer.
	F8	Privilege management	The access to the stored data can be fine controlled on a per-user basis.
	F9	Alarm handling	Users can specify conditions when an alarm is triggered in case an objects movement is reported.
Quality Assurances	**Q9**	Availability and Reliability	The service layer has a high availability and delivers its service reliably.
	Q10	Security	Data and connections are protected against security issues.
	Q11	Privacy	Data privacy is assured.
	Q12	Extendability	The functionality of the Service Layer can be easily extended.
	Q13	Maintainability	The application can be easily maintained.

Table 4.4: Requirements for the Sensing and Network Layer

Domain	**ID**	**Name**	**Description**
Features	**F1**	Positioning	The tag can determine its position.
	F2	Communication	The tag can transmit data to a server.
	F3	Rechargeable	The tag can be recharged.
	F4	Configurable	The tag can be configured remotely.
Quality Assurances	**Q1**	Affordable	The tag is affordable for private and business customers.
	Q2	Reliability	The tag reliably transmits its location to the Service Layer.
	Q3	Security	The tag is not vulnerable to security issues.
	Q4	Privacy	Unauthorized persons can get as little information out of the tag as possible.
	Q5	Long Lifetime	The tag should run as long as possible without recharging.
	Q6	High coverage	The tag's communication and positioning technology has a high coverage in domestic, urban and rural settings.
	Q7	Accuracy	The tag reports its position with an accuracy of 1-2 m.
	Q8	Flexibility	The tag can run with different positioning and communication technologies.
Usability	**U1**	Small size	The tag is small and therefore attachable to all sorts of objects.
	U2	Aesthetics	The tag and potential additional hardware are aesthetically pleasing to the users.
	U3	Easy set up	The tag and potential additional hardware are easy to set up – if possible without external help.

Table 4.5: Requirements for the Interface Layer

Domain	**ID**	**Name**	**Description**
Features	**F10**	Account Management	The Interface Layer allows to create and configure user accounts and reset login passwords.
	F11	Display Positional Data	The user can look at the location of the tags.
	F12	Configure Alarm Settings	The conditions for a tag's alarm triggering can be configured.
	F13	Receive and Handle Alarms	Alarms can be received and confirmed/deactivated.
	F14	Tag Management	Tags can be registered, configured and shared with other users.
Quality Assurances	**Q14**	Availability and Reliability	The interface has a high availability and works reliably.
	Q15	Responsiveness	The Graphical User Interface (GUI) adapts to different display sizes.
	Q16	Security	The GUI is protected against possible security issues.
	Q17	Privacy	The GUI treats the users' data with care to avoid privacy issues.
	Q18	Extendability	The GUI can be extended to meet requirements identified in the future.
	Q19	Maintainability	The GUI is easy to maintain.
Usability	**U4**	Appealing Interface	The GUI is perceived by the users as appealing and modern.
	U5	Easy Usage	The user finds it easy to learn the system's workflows.
	U6	Fast Account Creation	A user account is created within minutes.
	U7	Fast Tag Registration	A tag is registered within minutes
	U8	Efficiency	The workflows are as short as possible.

Chapter 5

Design

Section 4.4 elaborated the requirements of an OTS. This chapter explains the design of Pharos and how all the requirements will be met. The end users and the developers are considered the most important stakeholders (see Section 3.2). Therefore, the focus of the design lies in a high usability and a sound software architecture. The sound architecture includes a clean modularization of the individual components and the use of modern development standards. This modularization aims to maximize the maintainability and extendability of the system, both of which are also requirements. These two aspects are especially important since the proposed system is considered an evolutionary prototype, meaning that it will be extended upon in the future.

The architecture follows the principle of a SOA with four layers, as shown by the high-level architecture diagram in Figure 5.3. This chapter first discusses the individual layers and then explains the communication among them.

5.1 Sensing Layer

The Sensing Layer of an IoT software system contains the IoT devices. These are the tags in the case of an OTS. According to the requirements listed in Table 4.5, these tags measure their position. Their second core functionality is to send the measured position to the Service Layer. A platform which integrates and coordinates these two functionalities is the third mandatory component of the tag. This section explains which technologies will be used for these components.

The tag's **platform** integrates the positioning technology and the communication technology to a single system and further runs the tag's firmware. A rechargeable tag (F3) is the first requirement. There are two options to meet this requirement: The first option consists of adding a hardwired and rechargeable battery, such as a lithium-ion battery. The second option uses a replaceable power source such as batteries or a USB power bank. The developed prototype will use the second option because of its simplicity. The configurability of tags (F4) is the second requirement. The configurability will be addressed by

the tag's firmware, which can listen to incoming messages and adapt its behavior accordingly. Flexibility (Q8) regarding of the used technologies is a further requirement. The configurability and flexibility requirements favor a programmable MCU (see Section 2.1) over a custom made Printed Circuit Board (PCB).

The choice of an MCU is strongly restricted by the requirement of an affordable (Q1) product. Therefore, an existing platform with existing hardware has to be used since custom devices are too expensive in small numbers. Also, the need for reliability (Q2) favors existing platforms, as custom builds would require exhaustive testing to reach a similar level of reliability. Security (Q3) and privacy(Q4) issues are mostly the responsibilities of the tag's software. However, the many protocols and algorithms used for these two topics need to use a minimum of computational power or dedicated hardware. The long lifetime requirement (Q5) requires power-efficient hardware, since a too large battery would make the device bulky (U2, U1).

Due to the stated reasons, the candidate platforms are freely available MCUs which are well programmable, power-efficient but still have sufficient computational power. The mentioned requirements make the *Arduino UNO* [46] platform an excellent choice. Even though it is neither the most power-efficient nor the smallest MCU, it is well suited for a first prototype since it provides a reasonable amount of computational power and is highly customizable. Moreover, for an improvement of the tag, a smaller and more efficient sibling of the Arduino MCU can be used. A custom built PCB with a highly optimized selection of components could be an option for an end-product but is beyond the scope of this work.

Positioning (F1) is the second major functionality of the tag. There is a wide variety of candidate technologies for that purpose (see Section 2.4). The requirements from Table 4.5 demand a technology which is accurate (Q7), reliable (Q2), has a high coverage (Q6) and is affordable (Q1). Further, the technology should be power-efficient (Q5), easy to set up (U3), and not too large (U1).

The ease of set up categorically rules out all Bluetooth and WiFi-based approaches. Both of them rely on fingerprinting which requires an extensive training phase, to create the position signal strength table. RFID-based systems are also not an option as they require stationary hardware, which contradicts the high coverage or easy requirements. Also, the stationary hardware would imply initial costs. The accuracy criteria rules out cellular-based approaches (see Table 2.2). The remaining technologies are the GNSS-based approaches. Among these technologies, GPS is favored due to the high availability of the components.

The third key feature of the tag is **communication**. According to the requirements in Table 4.5 the communication technology should be secure (Q3), consider privacy (Q4), be energy-efficient (Q5), affordable (Q1) as well as have a high coverage (Q6).

The high coverage rules out ZigBee and Bluetooth. While being valid candidates for indoors, they are not suitable for outdoors as a sufficient coverage would require many receivers. WiFi consumes too much power and therefore is also not an option. The remaining candidates with a sufficient coverage are SigFox, LoRa and cellular. As cellular's newest version 4G is not very power-efficient, it is also not regarded as an option. Further,

cellular involves a monthly fee for the infrastructure access which makes it more expensive than the other two options.

The tag prototype of Pharos will use LoRa. The choice in favor of LoRa and against SigFox was made based on the better availability of the hardware components at the time of development.

To summarize, the tag prototype of the Pharos system will be implemented as an Arduino-based device which uses GPS for positioning and LoRa for the communication with the Service Layer. Both, GPS and LoRa have a limited quality of service in indoor settings. Therefore, the tag prototype is considered as an outdoor only prototype. Also, the Arduino-based platform will have a limited usability due to its size and insufficient power-efficiency. Therefore, the tag it is considered a proof of concept prototype which will require an in-depth improvement or redesign in the future.

5.2 Network Layer

Section 5.1 elaborated the design of the Sensing Layer and the tag. Once the positional data is collected, it has to be transmitted to the Service Layer. As previously stated, the LoRa technology will be used for communication. This section explains how the messages sent from the tag will be processed in the Network Layer.

The signals of a LoRa communication chip are received by LoRa base stations which are called LoRa gateways. Gateways can be purchased and installed to create private LoRaWANs. A LoRaWANsis a network, usually with a star-of-stars topology, which connects end-devices to a central server [47]. However, creating a personal network infrastructure will always have limited coverage. Hence, using an existing one is the better option.

There are two major LoRaWAN providers in Switzerland. One is the telecommunication company Swisscom which runs the Low Power Network (LPN) [48]. LPN mostly has a good coverage and offers different pricing plans for different use cases. The other provider is the community-driven The Things Network (TTN) [49], which can be used free of charge. At the time of this writing, the coverage of TTN in Switzerland is quite good, but not as good as the one of LPN. If it is insufficient, users can add new gateways to the network on their own. Even though TTN does not charge for usage, they limit the amount of air time per LoRa transmitter to several seconds a day.

For the use case of this work the TTN is the better option. It is free to use (Q1) and therefore well suited for prototyping. Also, the option of adding the personal gateways is interesting as this solves possible connectivity issues (Q6) during testing.

Using TTN has the restrictions of a traffic cap and comparably lower connectivity which have to be considered during implementation. If the connectivity becomes a severe issue, there are two options. As already mentioned, TTN allows to set up additional gateways. If there is no gateway at hand or the area to cover is too large, the Swisscom solution can alternatively be used. This option would require some changes in the Network Layer but not in the Sensing Layer.

5.3 Service Layer

Once the positional data from the tags in the Sensing Layer is transmitted via the Network Layer, the data will arrive in the Service Layer. The Service Layer receives the data and has to store it in a persistent database (F5). According to the requirements in Table 4.5, the Service Layer has been able to receive data from different types of tags (F6). Even though the design of the Sensing Layer specifies only one type of tag, this capability has to be part of the Service Layer. A front-end API (F7) makes the stored data accessible to the Interface Layer. This API will use a fine-grained privilege management system (F8) to assure security and privacy. The alarm handling (F9) will be a further functionality of the Service Layer. This section will explain how the Service Layer is designed for this functionality while taking into account the quality requirements in section 4.5.

To **integrate the various sensor types** (F6) a uniform REST API (see Section 2.2) will be provided using HTTPS. To integrate devices not capable of connecting to this API a *forward proxy service* has to be implemented, which takes care of the compatibility issues. The main security (Q10) and privacy (Q11) threats for this API are eavesdropping and invalid data sources. The eavesdropping problem already is solved by using the HTTPS protocol. The authenticity of the data source has to be assured by an authentication scheme. For the sake of simplicity, the integration of sensors is considered unidirectional. Unidirectional means in this context that no data can be sent to the sensors. However, the proposed design allows for adding this functionality at a later stage (Q12), as the proposed proxy services could handle any compatibility issues.

The persistent database (F5) will be a relational database, which is well suited for storing structured data such as user information, tag information or sensor data. Further, the database will require an add-on for the efficient treatment of positional data. Such an add-on will allow for more efficient comparison of areas and points, which will be needed for the alarm handling. The option of a time series database was discarded as only a small amount of positional data per sensor is expected. The hosting of the database is assumed to be trustworthy. Therefore, the data is considered safe regarding security (Q10) and privacy (Q11).

It is a bold assumption that the hosting of a system is trustworthy without limitations. A full discussion of security threats originating from the hosting services is beyond the scope of this thesis. For the sake of completeness, here are some points to consider: To assure the privacy of this system the principles of "Privacy by Design" (see Section 2.6) should be met. For meeting the "Hide" principle, the data in the database should be encrypted. Otherwise, a curious administrator or a versed attacker can get hold of the positional data of all the tags.

To summarize, a REST API (see Section 2.2) will **connect the Service Layer to the Interface Layer** (F7). The API will be secured by an HTTPS connection and an authentication scheme. On top of the authentication, there will be a fine-grained access control, which will be elaborated upon later. To reduce the complexity of the prototype, the implementation will not make all the data stored in the database available. However, the architecture will assure this missing functionality can be added later on as well as the possibility of an extended data model (Q12).

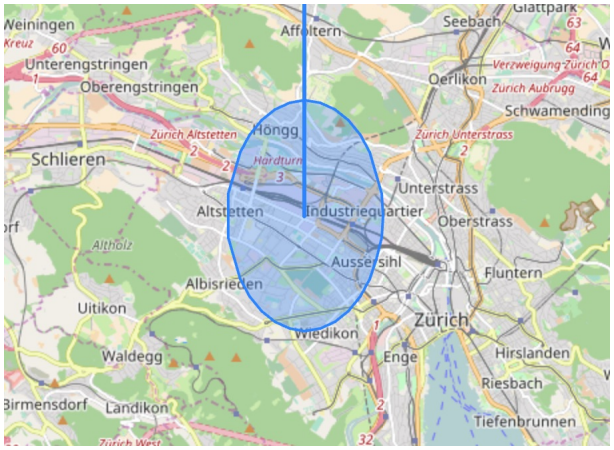


Figure 5.1: Circular Geofence view in the city of Zurich

A well-designed **privilege management** will take care of some of the Service Layer's security (Q10) and privacy (Q11) issues. It ensures that only users can access and manipulate data which are explicitly allowed to do so. The privilege management will be based on an authentication scheme and will allow per data entry privileges. The authentication will be a two-factor authentication scheme (see Section 2.7.3). The first factor will be a username a password combination. The second factor is an Short Message Service (SMS)-based security code. First of all, the authentication ensures that only the right people can access the REST API. The privilege management will have three privilege levels: *None*, *Read* and *Read/Write*, which will be checked at every request to the API. Items with a *None* privilege are neither modifiable nor visible to the user. The *Read* privilege gives the user the permission to access the data but not to modify it. The *Read/Write* permission gives a user all permissions for an item, which involves accessing, modifying as well as deleting the item.

The requirements presented in Section 4.4 imply that a tag can be shared. The sharing in this prototype will be translated to *Read* access. However, this does not cover use cases such as *View on Alarm*, which has been mentioned in Chapter 3. Therefore, the privilege management must be realized in an extendable manner, mapping quality attribute (Q12).

A further requirement for object tracking is the alarm functionality (F9). This alarm functionality includes the triggering of alarms under certain conditions, the processing of alarms as well as involving the user into this process. The alarm triggering will be based on Geofencing. A Geofence is a virtual area, bordered by a virtual fence which is defined by a set of corner points [50]. Figure 5.1 shows an example of such a Geofence. As soon as an object is reported outside its Geofence, an alarm will be triggered. The procedure of the alarm handling is shown in Figure 5.2.

At first, the user defines the area of the Geofence. For simplicity reasons, the area is restricted to circular shapes. Once the area is set, and the tag is activated, violations of the Geofence will trigger an alarm. Violations change the alarm state to *pending* and the user, as well as the shared users, will be informed. Now, the informed users can disable the alarm within a certain time frame to prevent false positives. The tag changes back to a no alert state and is ready to be triggered again, once the tag is reported inside

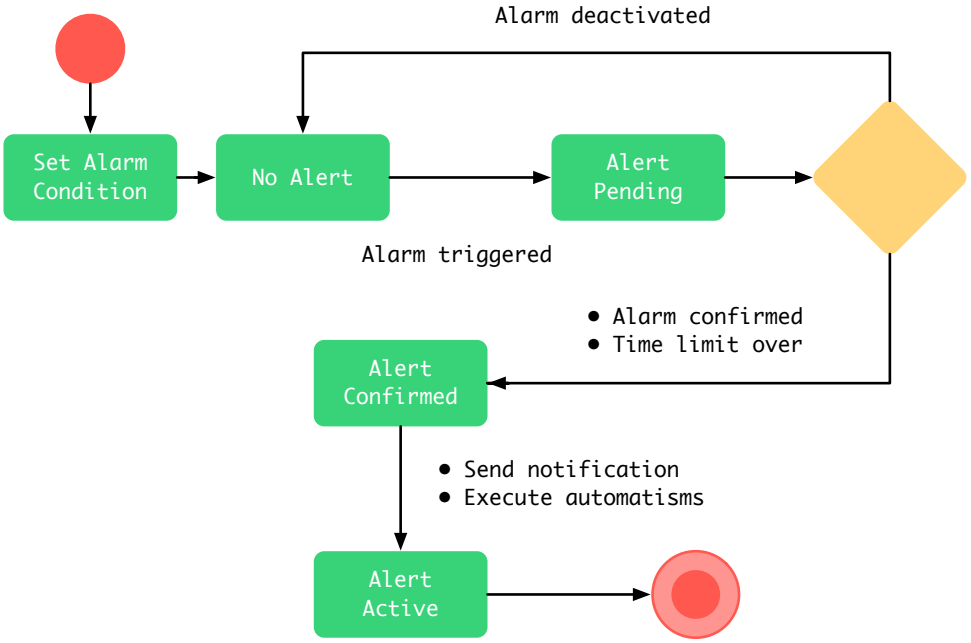


Figure 5.2: Configuration and handling of alarms

the Geofence again. If a user confirms the alarm or the time expires the alarm state changes to *confirmed*. Further notifications are sent to the users and shared users, and also possible automations are triggered. Then, the alarm's state changes to *active*. Tags in this state reveal more information to shared users than when in the non-alarm state. The extendability (Q12) of the alarm handling must be assured by a clear implementation of the trigger condition as well as the subsequent alarm handling.

The entire tracking application will be implemented as a web app (see Section 2.2). Of which the back-end is located in the Service Layer. The back-end implements the REST APIs and deals with the alarm handling. Its architecture and implementation will follow a modular architecture to assure extendability (Q12) and maintainability (Q13).

The rest of the Service Layer will be concerned with DevOps. It will assure a proper hosting of the web app as well as provide the third party services, such as the data storage. The DevOps has to assure that the Service Layer runs reliably and has a high availability (Q9). It also has to remain easily maintainable (Q13) and extendable (Q12).

These quality attributes will be met by using the advantages of virtualization and containerization (see Section 2.2). The containerization of an app can increase its availability by having a backup container running the same service. If the first container fails, the second can take over. The same holds for the scalability. If the amount of sensor data or the requests from the front-end are too high to be handled by the back-end, a second instance can be launched to deal with it. As mentioned in Section 2.2 containerization requires applications to be stateless. Since REST APIs are stateless by definition, this is given by design. Maintainability also increases through containerization, as the images are self-sufficient. Any updates required by the images will only affect the image and leaves all the others untouched. The fact that changes only have local effect also increases the extendability of the system.

Summarizing, the Service Layer will contain the back-end of a web app, which is accessed through two REST APIs: One for the connection to the Network Layer and one for the access from the Interface Layer. The connections for both APIs will be secured by enforcing HTTPS to ensure security and privacy. Further, there will be authentication and authorization mechanisms to make the app secure. The web app also implements the logic for the alarm handling. The Service Layer will further offer a persistent data storage as a relational database. This database uses an extension for geospatial data, for the stored positional data. Besides these two interfaces the back-end stores all the data which is necessary for the proposed OTS. In addition to that, an alarm handling mechanism is implemented which allows using Geofencing for objects and which also handles the possible violations. The hosting of the Service Layer will be based on the principles of containerization.

5.4 Interface Layer

So far, the system design provides the functionality to collect data, transmit the data to the back-end and to access the data through a well-defined API. The Interface Layer now makes the system accessible to the users. On the one hand, it presents the user the positional data and the tag configurations. On the other hand, it allows the user to interact with the back-end which involves the registration of tags and the configuration of tag alarm settings. This section first presents how the data is displayed to the user. Then, the interaction schemes are elaborated. Last, the general design decisions for the entire Interface Layer will be explained.

The **Data Presentation** distinguishes between positional data and auxiliary data. Geographical data (F11) is different since humans cannot intuitively make sense of it. For most humans, coordinates are an abstract concept which requires some effort to read. For that purpose, the positional data will be displayed on a map. Each tag has two interesting properties regarding its positional data. The first one is its current position. The second is the history of past locations. The GUI will separate these aspects in different presentation modes. This separation aims for a more efficient (U8) user interaction and a more appealing (U4) and well-structured interface. An overview map will display the current position of all tags. A detail view of every tag then presents the history of past positions.

The **Data Manipulation** involves dealing with tags (F14), creating and managing user accounts (F10) and dealing with alarms (F13, F12). The Interface Layer has to be designed to support an efficient handling of these tasks (U6, U7, U8). Further, the different actions have to be presented in an attractive and easy-to-learn manner (U5). These issues mostly depend on the way they are implemented and are thus not further addressed at this point.

Following the OWASP Top 10 list (see Section 2.7.2), the security-relevant issues in the Interface Layer (Q16) are *Broken Authentication and Session Management*, *Cross-site Scripting*, *Cross-Site Request Forgery* and *Unvalidated Redirects and Forwards*. They have to be addressed by the implementation. Checking for privileges and permissions is not part of the Interface Layer since these issues are already handled in the Service

Layer. However, users must get an intuitive understanding – through a well designed GUI – which actions are available and which are not.

As mentioned before, the presented system design will as far as possible follow the "Privacy by Design" principles presented in Section 2.6. For enforcing the "Inform" and "Control" principle, the front-end must present the data collected about the user and the tags and provide the option to change or delete this data. The "Inform" principle becomes especially important when dealing with shared tags. It has to be assured that a user can easily see with whom a tag is shared or what someone is doing with a tag.

As the presented OTS is developed as a web app, the GUI will be implemented as a web page. To stick to the terminology of the web app architecture, the GUI will be called the *front-end*. The implementation as web page has the advantage that it is hardware and OS-independent. The page will further implement a responsive design (Q15), so it can be used on both mobile and stationary devices. Hence, the page's content will automatically adapt to the screen size. The front-end's availability and reliability (Q14) strongly depends on the same factors as the back-end. If the back-end is unavailable, the front-end will not be available either. Further, the availability and reliability of the front-end strongly depend on the delivery of the data required by the page. These quality goals are best met by a redundant hosting.

The Interface Layer's maintainability and extendability (Q19, Q18) will be achieved by a clean and well-structured project setup.

In short: The Interface Layer will be implemented as a web page, the so-called front-end. The communication with the back-end is handled entirely over the REST API. The data received will be presented in a well-structured manner to achieve a high usability and learnability of the system. The positional data further will be presented as a map because geographical coordinates are hard to make sense of for most humans. Also, data manipulation will be possible. However, the front-end does not check for a user's permissions. This task is delegated entirely to the back-end. The front-end implementation will make sure that neither Cross-site Scripting nor Request Forgery is possible. Also, Unvalidated Redirects and Forwards are prevented by the front-end's implementation.

5.5 Summary

This chapter explained the individual layers of the Pharos OTS. This section now summarizes the design decisions and assembles them to a consistent system architecture which is depicted in Figure 5.3.

The Sensing Layer consists of Arduino-based tags, which use GPS for positioning. This tag is going to be small and power-efficient to guarantee a high usability. A LoRa communication module connects the tag to the Network Layer.

The free to use LoRaWAN provided by TTN forms the Network Layer, which connects the Sensing Layer to the Service Layer. The latter one will include two components: The back-end, which is responsible for storing the data, dealing with the authentication and

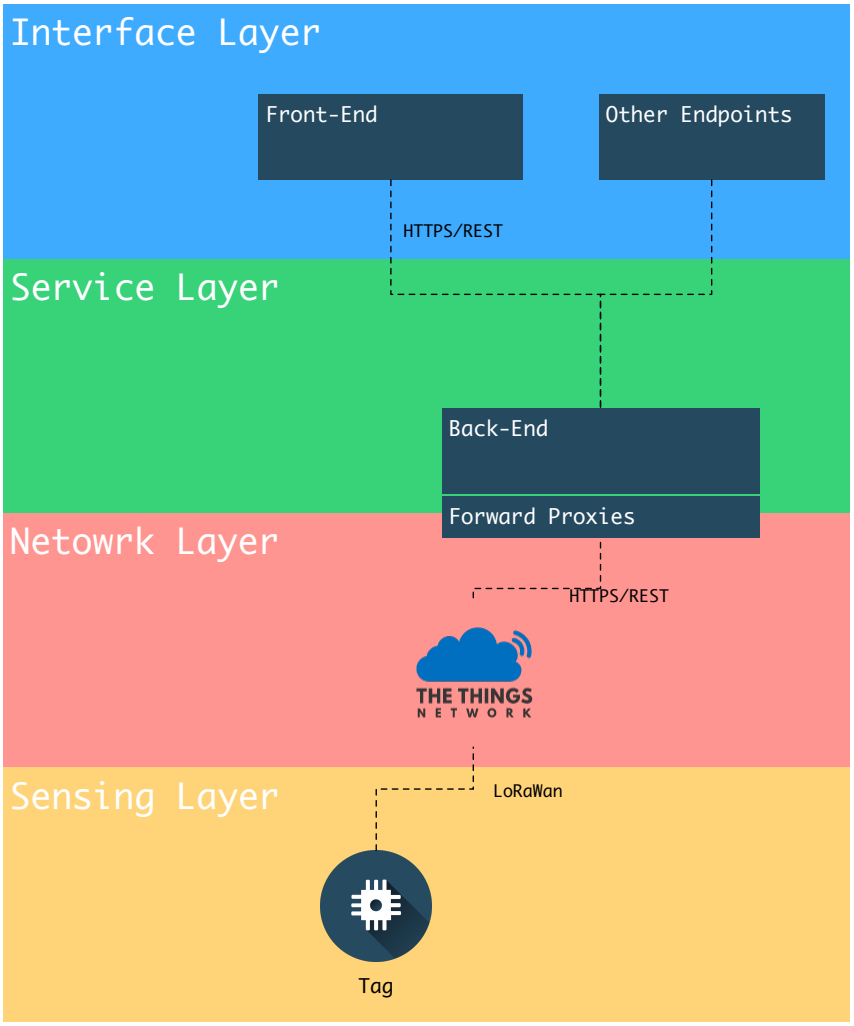


Figure 5.3: System architecture of the Pharos OTS

authorization as well as with the alarms. Two HTTPS-secured REST APIs connect the back-end to the Interface Layer as well as to the Network Layer. A forward proxy service is the second component of the Service Layer. This proxy handles the conversion between the communication protocols of the Network Layer and the REST API of the back-end.

The GUI – the Interface Layer's main component – is implemented as a web page. This page displays the data received from the back-end through the HTTPS/REST API and gives the user the ability to manipulate this data as well. The next chapter will elaborate how the prototype version of the Pharos OTS implements the design decisions presented in this chapter.

Chapter 6

Implementation

The previous chapter presented Pharo's system design which meets all the requirements listed in Section 4.4. This chapter describes the implementation details for each of the four layers: Sensing Layer, Network Layer, Service Layer, and Interface Layer

6.1 Sensing Layer

According to the design from Section 5.1, the tags are the only components in the Sensing Layer. For this prototype system, only one tag was built. This tag is based on the Arduino platform and integrates a GPS and a LoRa communication module. The focus of this work lies on the software side; the tag implementation is therefore very simple. Also, there will be no in-depth reasoning regarding the usage of specific technologies or chip versions. Figure 6.1 shows a schematic of the Sensing Layer's implementation. This section does not cover the tag's software sides specification, as it was provided before.

The tag's *hardware* is shown in Figure 6.2. The MCU(1) used for the tag is a Freaduino UNO Rev1.8 [51] board. This board is a less expensive clone of the Arduino UNO series, which is fully compatible with the original. A Dragino LoRaWAN shield [52] (2) is attached on top of the Freaduino. The shield, used for the LoRa communication, is based on Semtech's RFM95W LoRa transceiver. The shield's antenna is seen on the right-hand side of the LoRaWAN shield. The GPS receiver GY-NEO6MV2 [53](3) can be seen right to of the antenna, attached to a breadboard. The module uses a U-Blox NEO-6M GNSS receiver chip. This chip is manufactured by U-Blox, a leading GNSS module manufacturer located in Switzerland. The USB cable (4) on the left-hand side powers the tag when plugged into a power bank or stationary USB power source.

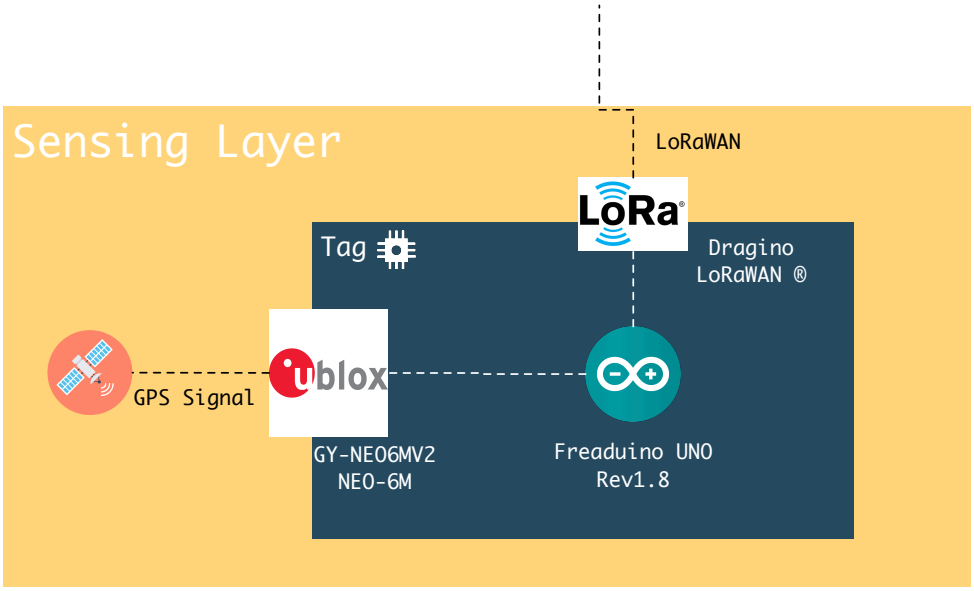


Figure 6.1: Implementation of the sensing layer

(1) Freaduino UNO [51]
(2) Dragino LoRa Shield [52]
(3) GY-NEO6MV2 GPS Module [53]
(4) USB cable for power

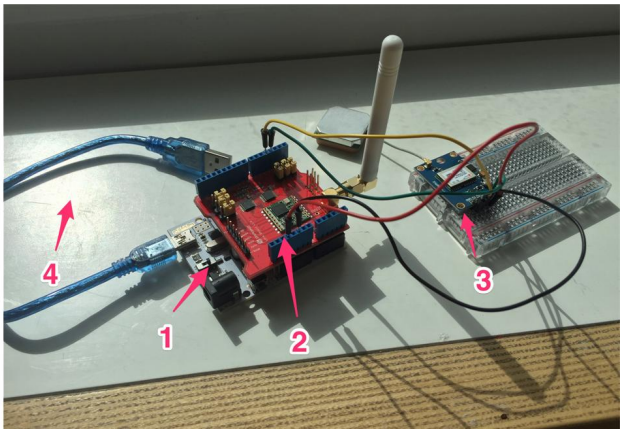


Figure 6.2: Prototype of a tag.

6.2 Network Layer

This section describes the implementation of the Network Layer, which follows the design decisions in Section 5.2. The Network Layer uses TTN [49] and its free-to-use LoRaWAN. The setup required to use the TTN LoRaWAN requires the following steps: service creation, service set up, and tag registration. Figure 6.3 shows a diagram of the Network Layer's implementation.

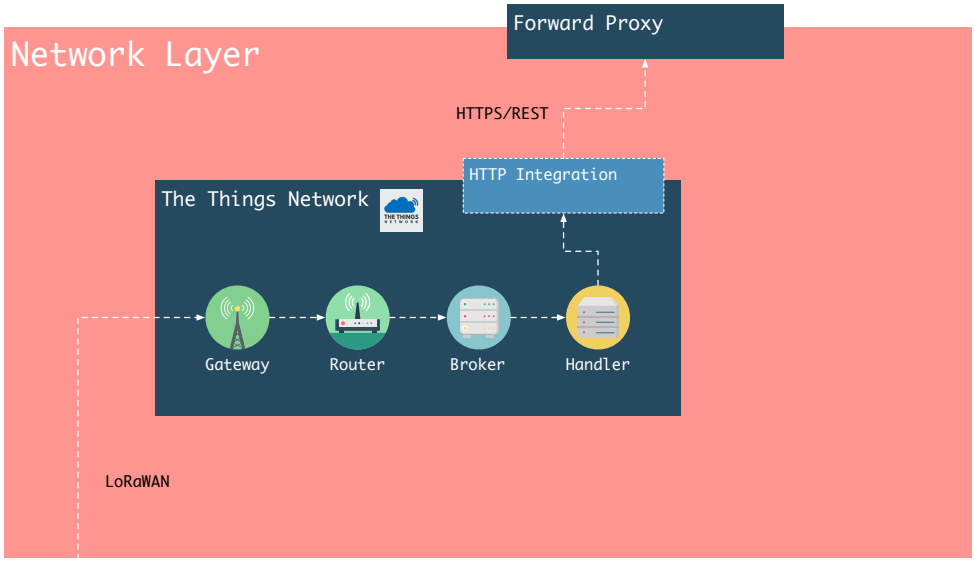


Figure 6.3: Implementation of the Network Layer

The creation of a TTN app is the first step towards using TTN's LoRaWAN. A TTN-app is a logical entity to which LoRa devices can be assigned. The app further defines what happens with incoming data. Figure 6.4 shows a screenshot of the TTN website which depicts the two available handler classes for incoming data. Once an app is set up, devices can be registered. The registration depends on the used registration scheme. More information about the different options can be found on the TTN website [49].

Once a tag is registered the tag transmits data to the TTN app, which has several options for processing the incoming data. This project uses *HTTP Integration* to forward the data as an HTTPS message to the Service Layer. Figure 6.5 shows the corresponding configuration panel. The endpoint URL, the used HTTP method, as well as the authorization header can be set. According to the specification of Section 5.2, an HTTPS connection and a username/password authentication scheme are used.

6.3 Service Layer

So far the implementation of the Sensing and Network Layer has been discussed. This section now explains processing of the incoming data, the back-ends internal logic and the implementation of the two REST API's The implementation of the back-end will be elaborated first before presenting the surrounding services in the DevOps part.

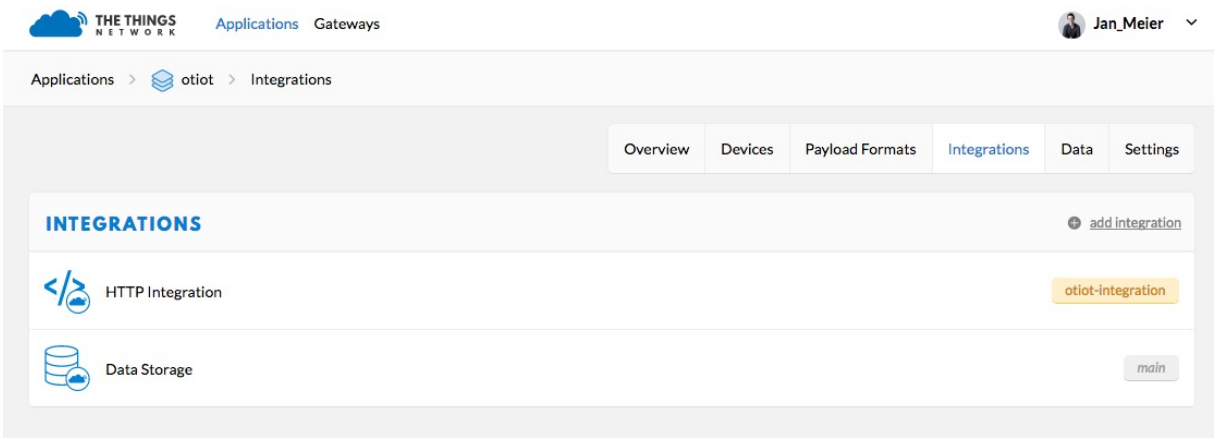


Figure 6.4: List of the two types of integrations provided by TTN.

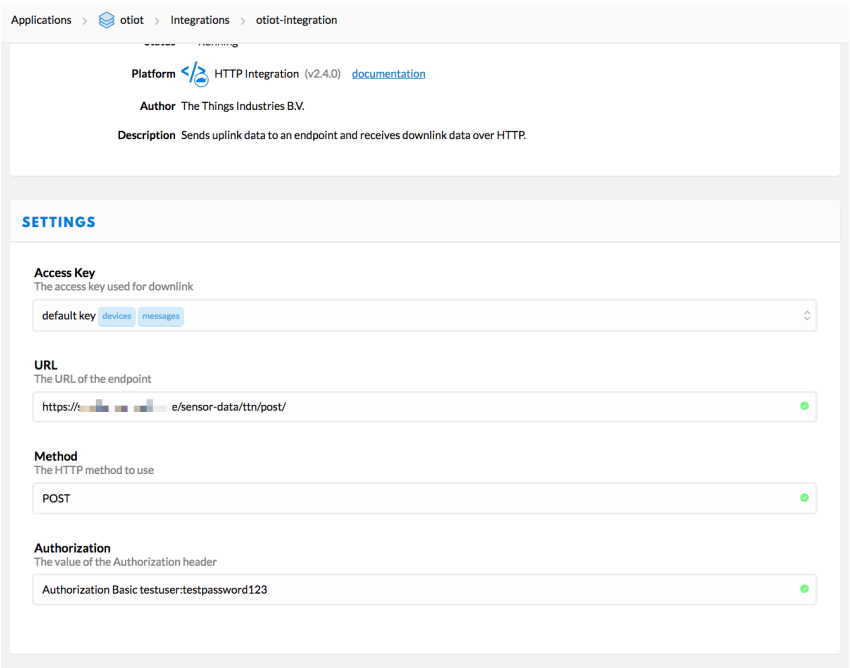


Figure 6.5: Configuration of a TTN HTTP Integration.

6.3.1 Back-End

The back-end of the Pharos OTS is implemented with the Python web framework Django (see Section 2.2), along with a Postgres [9] database. Postgres [9] is an open-source relational database, which Django has integrated bindings for. Further, the PostGIS [54] extension for Postgres adds features related to geospatial data. The GeoDjango [55] library makes these features accessible to Django. The Celery [56] distributed task queue is used to add parallelism to Django. The RedisDB provides the key-value storage which Celery requires to store its task queue [57]. Figure 6.6 shows an overview of the mentioned technologies.

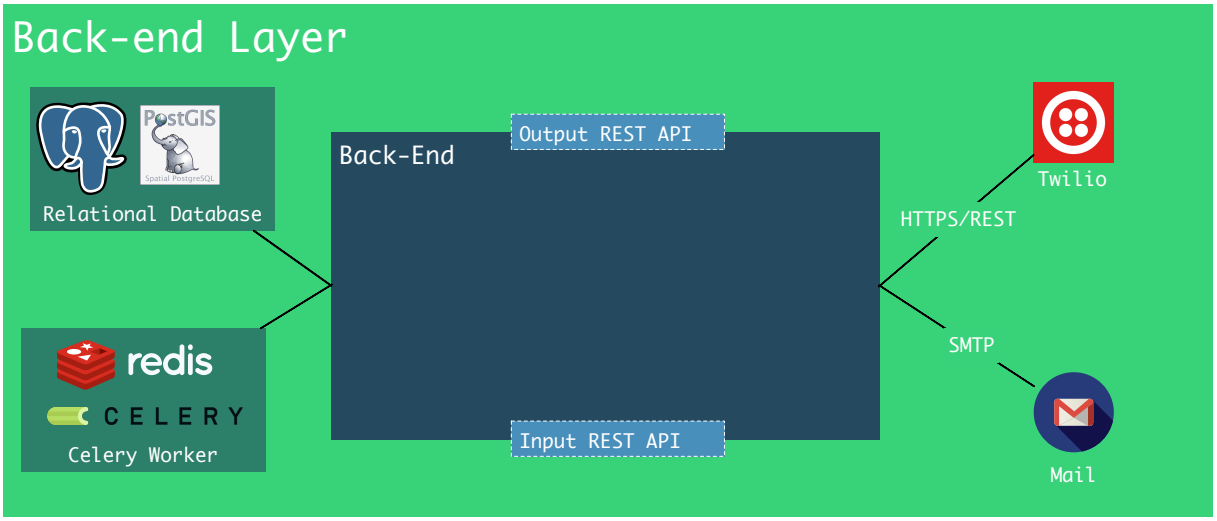


Figure 6.6: Implementation of the back-end

Figure 6.8 shows a module diagram of the Django web app and Figure 6.7 outlines the back-end's the project structure. In the following sections, the purpose and functionality of the Django apps (see Section 2.2) listed in Figure 6.7 are elaborated.

The **alarm** app deals with the alarm configuration and handling. The configuration of the alarms is handled by the `AlarmConfig` and `AlarmConfigArea` classes (Figure 6.8). The first class defines the time for alarm deactivation and the second the circular Geofence (see Section 5.3). Figure 5.1 shows a visualization of a circular geofence, which is displayed as an oval, due to the projection of Earth's curvature. Due to this separate class other shapes could be added in a modular manner at a later stage. The alarm app further implements the alarm handling shown in Figure 5.2. Every time new positional data from a tag is received, the app checks whether the corresponding tag's alarm conditions are met. If this is the case, the app notifies the users by email or SMS. Which method is used depends on the users' settings. The messages contain two links, one for confirming the alert and one for canceling it. Using links, instead of a separate web page, accelerates the process, which is useful for false as well as justified alarms. One-use codes, integrated into the link URL, protect the links against guessing attacks.

The back-end's registration process and the authentication/login functionality is implemented in the **login_registration** app. These features were realized with the help of two third-party Django libraries. The first library is *Django Registration Redux* [58] which

```
~/C/0/m/web ››› ls -l
total 64
-rw-r--r--   1 [redacted]  staff   213 May 25 17:44 Dockerfile
-rw-r--r--   1 [redacted]  staff   486 May 19 21:16 DockerfileBase
-rw-r--r--   1 [redacted]  staff  1133 Mar 21 15:44 Makefile
-rw-r--r--   1 [redacted]  staff  3104 Mar 21 15:44 README.md
drwxr-xr-x  19 [redacted]  staff   646 May 19 21:16 alarm
drwxr-xr-x   8 [redacted]  staff   272 Mar 31 14:18 dashboard
drwxr-xr-x   8 [redacted]  staff   272 Mar 29 13:08 home
drwxr-xr-x   8 [redacted]  staff   272 Mar 29 13:08 login_registration
-rwxr-xr-x   1 [redacted]  staff   543 May 19 21:16 manage.py
drwxr-xr-x  13 [redacted]  staff   442 May 19 21:16 master_thesis_backend
drwxr-xr-x   3 [redacted]  staff   102 May 15 11:57 public
-rw-r--r--   1 [redacted]  staff   610 May  4 21:17 requirements.txt
-rwxr-xr-x   1 [redacted]  staff   381 May 19 21:16 run_celery.sh
-rwxr-xr-x   1 [redacted]  staff   210 May 25 17:44 run_web.sh
drwxr-xr-x  13 [redacted]  staff   442 May 25 17:44 sensor_data
drwxr-xr-x   2 [redacted]  staff    68 May 19 19:20 static
drwxr-xr-x  14 [redacted]  staff   476 May 17 11:09 tags
drwxr-xr-x   7 [redacted]  staff   238 May 28 16:44 templates
drwxr-xr-x  13 [redacted]  staff   442 May 19 21:16 user
drwxr-xr-x   8 [redacted]  staff   272 May  2 13:57 utils
```

Figure 6.7: Code structure of the back-end app

provides the user account handling[1]. This includes tasks like the creation of user accounts and resetting of passwords. The library further introduces security features such as confirmation emails before an account is activated or the enforcement of strong passwords. The second library is *Django Two-Factor Authentication* [59] providing the two-factor authentication process (Section 2.7.3) based on a cod sent by SMS. The SMS broker *Twilio* [60] is used to deliver the messages.

The **master_thesis_backend** app defines the Django project setup. It contains the project's main configuration files as well as the root URL configuration.

The **sensor_data** app stores the incoming sensor data. The sensor data is not directly linked to the tags, which is visible in the module diagram, on which these two objects have no connection. This separation allows storing data on unregistered tags.

The tag management is implemented in the **tag** app. A tag is always registered to a user, has a name, and is either activated or not. Only activated tags trigger alarms. The app further contains a model for shared tags. A shared tag is a one-to-many mapping between several users and a single tag. Hence, a tag can be shared with multiple users.

The **user** app stores user-specific information for instance avatar images and how a user wants to be notified about alarms. As Figure 6.8 shows, the user data refers to the actual Django user model by a one-to-one relation. This implementation detail avoids a direct

[1]Django comes with built in user management. However it is intended for the Django back-end and is not well suited for non administrator users.

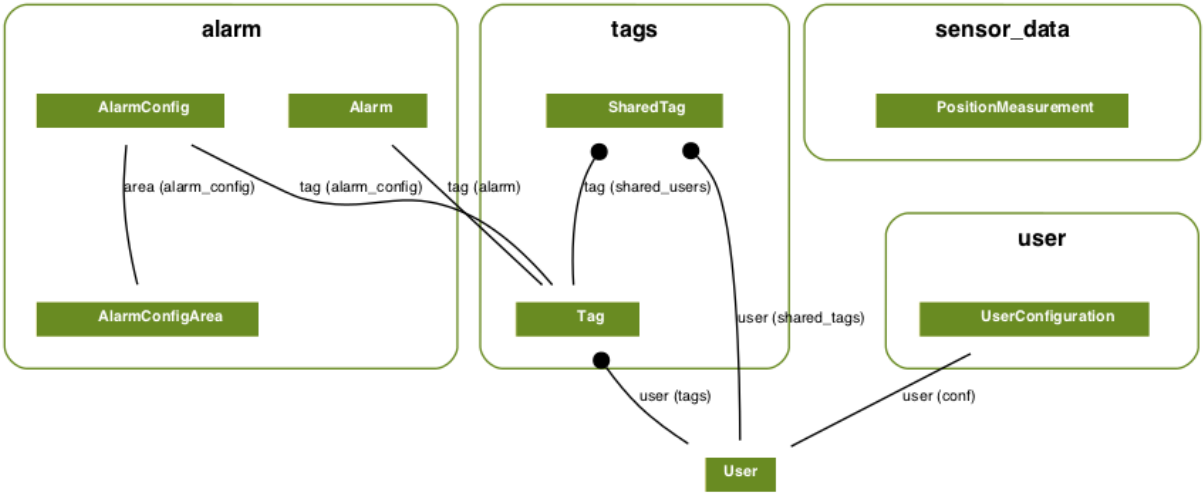


Figure 6.8: Class diagram of the Django app

extension of the Django's original user model. To extend that model would comprise the risk of breaking third party libraries that rely on that model.

The **utils** app contains various utility methods used by the other apps. It also implements an asynchronous image upload, which is used for the upload of the avatar images.

6.3.2 Output API

The back-end's REST APIs are realized with the *Django REST Framework*. This framework simplifies the creation of endpoints for the various models mentioned in Section 6.3.1. The creation of endpoints has two components: First, a serializer creates JSON serialized versions of the objects. Listing 6.1 shows Python code which implements such a serializer. The serializer defines which of the model's fields are included in the serialized version as well as which fields are read-only. The read-only fields can not be changed by the REST API. Second, views define the actual endpoints. They specify which query set they use and which serializer. Further, the view limits the query set by applying filters to it and defines security policies through permission classes. Listing 6.2 shows the code required for the tag model endpoint.

The back-end app provides end points for the following models: AlarmConfig (including the AlarmConfigArea), Tags, SharedTag, PositionMeasurement, UserConfiguration (including information from the original user model)

6.3.3 Forward Proxy and Input REST API

The design in Section 5.3 mentions forward proxy service to connect the back-end app to the communication layer. Since the TTN service, which is used in the communication layer, delivers the data already as HTTP requests, the forward proxy is not required for

```
    class TagSerializer(serializers.HyperlinkedModelSerializer):
        lookup_fieled = 'pk'
        current_position = PositionMeasurementSerializer(read_only=True)
        alarm_config = AlarmConfigSerializer(read_only=True)
        alarm = AlarmSerializer(read_only=True)

        class Meta:
            model = Tag
            fields = (
                'pk', 'name', 'charge_status', 'get_status', 'last_update',
                'active', 'avatar', 'uid', 'current_position', 'color',
                'alarm_config', 'alarm', 'url', 'user', 'shared_users'
            )
            read_only_fields = (
                'pk', 'charge_status', 'get_status', 'last_update', 'avatar',
                'current_position', 'color', 'alarm_config', 'url', 'alarm',
                'shared_users'
            )

```

Listing 6.1: Definition of a model serializer

```
    class TagViewSet(viewsets.ModelViewSet):
    queryset = Tag.objects.all()
    serializer_class = TagSerializer
    filter_backends = (filters.DjangoObjectPermissionsFilter, )
    permission_classes = (ExtendedObjectPermissions, )

```

Listing 6.2: Definition of a REST API view

the prototype implementation. Instead, the TTN service connects directly to the Input REST API.

The implementation of the input REST API also uses the Django REST Framework (see Section 6.3.2), but only provides one endpoint. This endpoint is limited to POST which expects a JSON-formatted message like the one shown in Listing 6.3.

6.3.4 Security and Privacy

The Django web framework addresses most security issues of the OWASP Top 10 list (see Section 2.7.2). The Django ORM prevents *Injection* of SQL code by checking the queries' parameters for SQL code. *Authentication Management and Session Management* is handled by Django. As mentioned in Section 6.3.1, the original authentication mechanism has been extended to a two-factor authentication which further protects against password guessing or overwriting. *Cross-site scripting* is also prevented by Django, by escaping potentially dangerous characters by default.

```
{
    "sensor_id":
  "tag_id",
  "measurements": [
      {
          "time_stamp": "2017-03-29T14:56:25.968986418Z",
      "position": {
          "latitude": 47.3866,
        "longitude": 8.505300
        }
    }
  ]
}
```

Listing 6.3: Example JSON for the input API. The message contains the tag's ID and a list of measurements. The measurements contain a time stamp as well as a position, which is specified as longitude and latitude values.

Insecure Direct Object References affects REST API but is well handled. The endpoints check for every request if the requesting user is allowed to access the demanded objects. API for unauthorized users is not possible at all. For example, a tag model instance can only be accessed by the user who created it. The privilege management knows three privilege management levels: *None, Read, Read/Write*. The enforcement of the access rights is done by limiting the available methods of a REST endpoint. A user who has Read access to a tag can only use the `GET` method but not the `PUT` method.

The *Security Misconfiguration* threat is tackled by the hosting of the web app, which will be explained later in Section 6.3.5. *Sensitive data exposure* is addressed by the usage of HTTPS encrypted communication and the fined-grained access control. The *Missing Function Level Access Control* is also solved by the access controlled.

Django has a protection against *Cross-Site Request Forgery* (CSRF) by default. Every processed `POST` request must contain a valid CSRF token. This token is generated before every access of the front-end. During the implementation, a focus was placed on checking that there were no *Components with Known Vulnerabilities* used. However, to assure that there will be no such components in the future, all components must regularly be checked. *Unvalidated Redirects and Forwards* are no threat to the app, as neither forwards nor redirects are used.

The principles of "Privacy by Design" (see Section 2.6) were followed. The developed system does not collect unnecessary data about the users to meet the *"Minimise"* strategy. The *"Hide"* principle is enforced by the encrypted communication. The encryption of the database could be improved, but this issue has been explicitly discussed in Section 5.3. The *"Separation"* principle was not considered necessary, as data is collected, for which separation would be useful. The *"Inform"* principle is met since a user can access all data collected about him or her. Also, the users can modify or delete this data which is demanded by the *"Control"* principle. The *"Enforce"* and *"Demonstrate"* principle were ignored since they would go beyond the scope of this thesis.

6.3.5 DevOps

The term DevOps is a combination of the words "development" and "operations." The goal of DevOps is to provide the tools and infrastructure to efficiently deploy and run the software. The previous Section 6.3.1 explained the implementation of the back-end and how the web app's logic is implemented. This section discusses how back-end interacts with the third party components and how the requests from the neighboring layers are processed.

As specified in the system design (see Section 5.3) the hosting is based on virtualization. More specific, Docker containers (see Section 2.2) are used. Figure 6.9 shows an overview of the used images. What each image does in detail is described below.

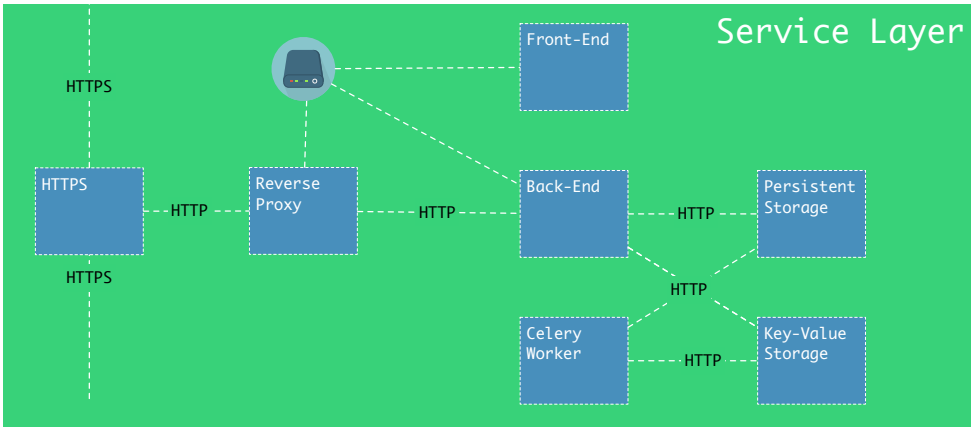


Figure 6.9: Overview of the used Docker images.

The incoming requests first arrive at the *HTTPS* container, which runs an instance of Nginx (see Section 2.2). This container translates them to plain HTTP and forwards them to the reverse proxy container. Let's Encrypt [61] is used as a CA for the HTTPS communication.

The *Reverse Proxy* container is a second Nginx instance which acts as a reverse proxy and file server. The file server part provides the files for the front-end web page. The *Front-End* container does not run any custom code but copies the front-end files to the disk, which then are transmitted by the Nginx server. The Reverse Proxy container also serves the static files of the *Back-End* container, which are copied to disk as the second mentioned container is started. The Back-End container runs an instance of the previously presented back-end Django application. The Back-End container connects to the *Persistent Storage* container, which runs an instance of the Postgres database with the PostGIS extension. The *Key-Value Storage* container runs a RedisDB instance.

The *Celery Worker* container runs the asynchronous tasks (e.g., receiving data, email sending) and is implemented with the Celery framework. As depicted, it is neither connected to the Reverse Proxy nor the Back-End. Instead, it is connected to the Key-Value Storage and the Persistent Storage container. The communication between the Back-End container and the Celery Worker container happens indirectly via RedisDB, located in the Key-Value Storage container.

6.4 Interface Layer

The Pharos object tracking is accessed by users through a website. This section explains how the website is implemented and how it is connected to the Service Layer.

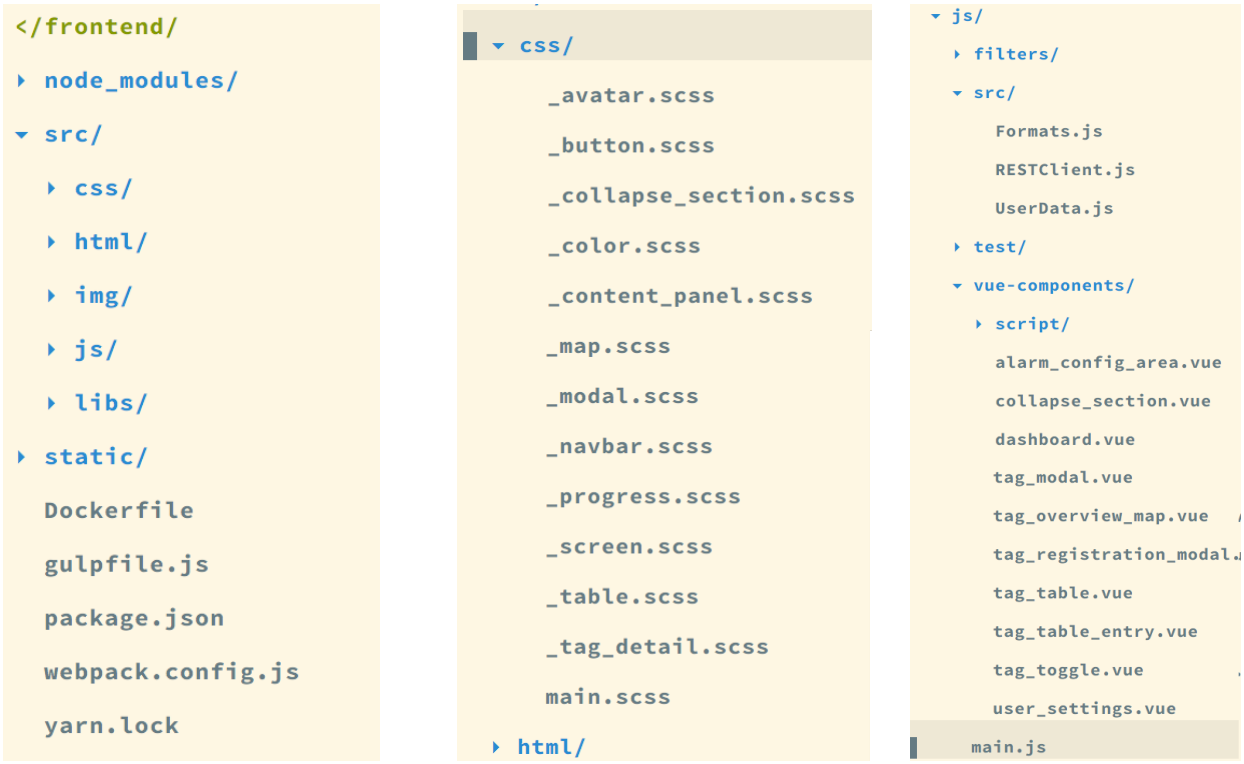
6.4.1 Technologies and Set Up

The web page is based on Hypertext Markup Language 5 (HTML5), JavaScript, and Cascading Style Sheets 3 (CSS3). Figure 6.10a shows the project structure of the front-end. The following part of this section explains the purpose of the files and folders.

The `node_modules` folder contains third-party libraries, which were imported using the npm package manager [62]. These libraries are then used by the local JavaScript code. The `src` folder is the root for all the source files of the front-end project which were written specifically for this project. The `css` folder, as shown in Figure 6.10b, contains the files to generate the front-end's Cascading Style Sheets (CSS). The files listed there are Sass files, indicated by the `.scss` ending. Sass is an extension of CSS code which allows for variables and macros. Further, the code can be split up into several files, which keeps the code well organized. These features of Sass make it better suited for projects with many CSS classes than plain CSS. The `main.scss` is the entry point for the Sass compiler which includes all the other files starting with a "`_`".

The `html` folder holds all HTML files used in the project. Similarly does the `img` folder for the image files. JavaScript files are located in the `js` folder whose internals are listed in Figure 6.10c. Some of the listed files are plain JavaScript, and some of them belong to the Vue.js framework [63]. Vue.js is a lightweight front-end framework written in JavaScript to create reactive web pages. In this context, "reactive" refers to the fact that the content is not static, but instead adapts to events such as from inputs or clicks. The components of the framework are split up into template files and script files. The template files lie in the previously mentioned `html` folder. They are not plain HTML files but allow additional, Vue.js-specific HTML tags. These additional tags allow more effective rendering of the underlying data. The script files contain the JavaScript part of Vue.js component. They define for instance which data is available, which actions are executed when a component is rendered or what happens when a specific form field is changed. The template and script files are bundled by the `.vue` files. `.vue`-files define the location of the template and script of the Vue.js components. The `main.js` file, which is the main entry point for the JavaScript code, includes the mentioned plain JavaScript files as well as the Vue.js files.

The `Dockerfile` file contains the directives to create the front-end Docker image (see Section 6.3.5). The build tool Gulp [64] is used for various tasks around the development. The `gulpfile.js` file contains the configuration for Gulp. The previously mentioned compilation of scss files is an example for a Gulp task. The `package.json` file is a utility for the package manager npm. It defines which libraries should be installed as well as some meta information about the project. A further task of Gulp is to compile the JavaScript files into a single deployable file. For that purpose, Gulp uses the Webpack [65] tool, whose configuration is stored in the `webpack.config.js` file. The last tool used in the



(a) Code structure (b) `.scsss` files (c) JavaScript files

Figure 6.10: Project structure of the front-end

front-end is the yarn packet manager [66], which uses the `yarn.lock` file. The `yarn.lock` file stores the project's dependencies, like the `package.json` file does but adds specific version numbers. Yarn uses npm under the surface but handles the package download in a more sophisticated fashion. For example, yarn avoids downloading the same package twice, when two individual libraries list it in their dependencies. Yarn instead downloads it once for both of these libraries.

6.4.2 Front-End Architecture

The implementation of the front-end web page is based on a three layer architecture. The *Communication layer* handles the communication with the back-end, by accessing the REST API. The Axios.js [67] library, a lightweight HTTP client, entirely written in JavaScript, is used for the HTTPS communication in the front-end. The communication layer abstracts away the networking, from the other two layers. This abstractin has the advantage if the back-end REST API would be altered, only this layer would have to adapt to these changes, which increases the maintainability of the front-end.

The Axios.js library is based on *promises*, which are a design pattern that simplifies the handling of concurrent tasks. The call of a promise accepts a function as a parameter that will be called once the promise's task is completed. Listing 6.4 shows how the data layer and the promise concept is used. Asynchronous code execution is crucial for a user interface, especially when dealing with network access, which often suffers from

a significant delay. If the code were executed synchronously, the entire execution of JavaScript code would wait for the response and thereby freeze the user interface. This non-responsiveness of the system would drastically reduce the user satisfaction.

```
1    var restClient = new RESTClient();
2    restClient.getTags(
3        function (tags) {
4            this.tags = tags;
5        }.bind(this),
6        null
7    );
8
```

Listing 6.4: Example code for the use of promises. As the RestClient's method for fetching the tags is called, a function is passed as parameter. Once the response of the backe-end arrives, the code from the passed function will be executed.

The second layer is the *Data Layer*. This layer stores and buffers the back-end data, such as information about the user and the tags. This layer is responsible for an efficient data management. The layer assures that the data items are only fetched once from the server and that the data is synchronous among the different parts of the front-end. Further, the Data Layer checks frequently whether the data in the back-end has changed and updates it locally when a change has been detected.

The *UI Layer* is the third layer and consists of the Vus.js components. The UI Layer implements the representation of the positional tag data on maps using *leaflet* [68]. This library simplifies the usage of available map tools such as GoogleMaps or MapBox. With this library, the same code can be used for different map data sources. Further, leaflet allows to draw on maps and to display GeoJson data. These features were used to put the tags' location markers on the map and to draw the Geofence.

Chapter 7

Evaluation

The prototype of the OTS is evaluated along the user story presented in Section 3.3.1 showing the intended operability of the tracking solution. The assumed scenario strongly influenced the requirements for the system and thereby also the design and implementation of the prototype. The testing follows the key features from the scenario, which were extracted in Section 4.3.

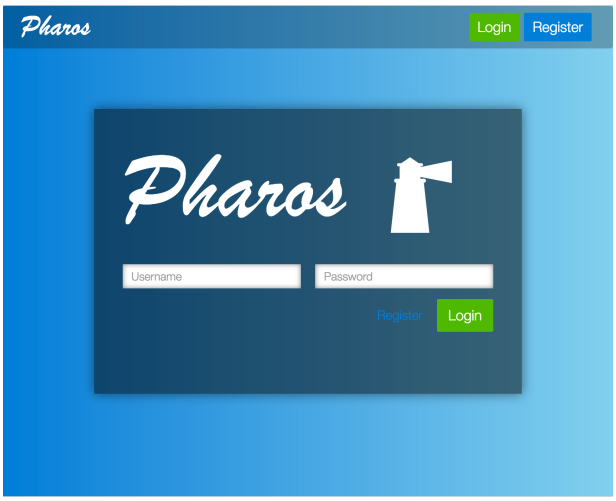
The tags consist of low-cost components in order to have solution in place that is **price-wise affordable**. Neither the communication nor the technology positioning is expensive. However, since the tag is an early stage prototype, this thesis cannot make a final verdict. The software for the object tracking system is entirely built on open source software, which keeps the costs for the software low. Currently the tag prototype is not efficient designed concerning its size as it can be seen in Figure 6.2, which can be further improved.

The web page's was requested to have a **professional appearance**, meaning it should show modern, consistent, and professional design. Figure 7.1, proofs this appearance together with all the other screenshots in this chapter.

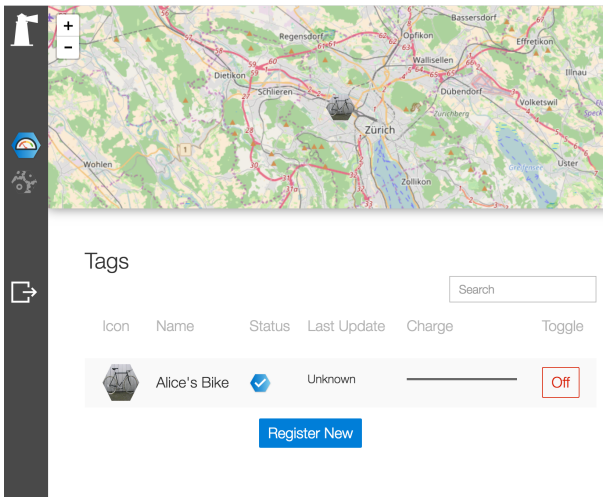
As shown in Figure 7.1, the available actions are easily visible and, thus, a **self-explanatory UI** was realized. Further, if there is detail information such as the purpose of a form field or restrictions to the allowed values, the web page displays them to the user at the right positions (see Figure 7.2a, Figure 7.5a).

The **registration process** was intuitive designed in four steps. First, the user enters his or her credentials. The username has to be unique, and the password has to be at least eight characters long, must contain at least one non-alphanumerical character and cannot be too common. Second, once the credentials are entered, the user receives an email, which contains an activation link. (see Figure 7.2b).

After the successful creation of the account, the user is asked to set up the two-factor authentication (see Figure 7.3a). Thus, the user is asked to enter a valid phone number. The system then sends a confirmation message with a code. The fourth and last step of the registration process is the confirmation of the phone number. To do so, the user has to enter the received code (see Figure 7.3b) in the web app. The user account is now successfully set up and ready to use.

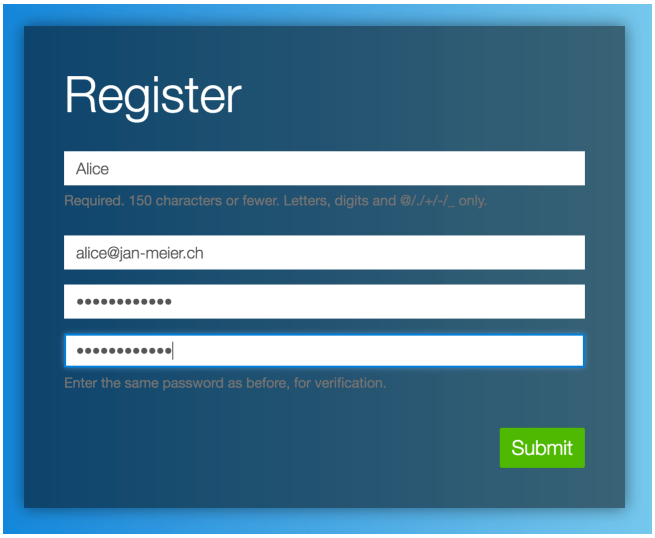


(a) The design of the home screen.

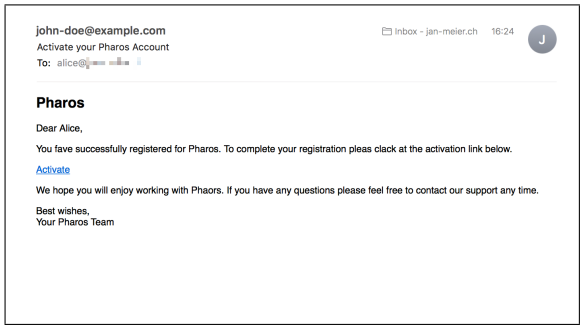


(b) The dashboard screen, which shows the list of registered tags as well as a button to register further tags.

Figure 7.1: Screenshots showing the modern and professional appearance of Pharos.

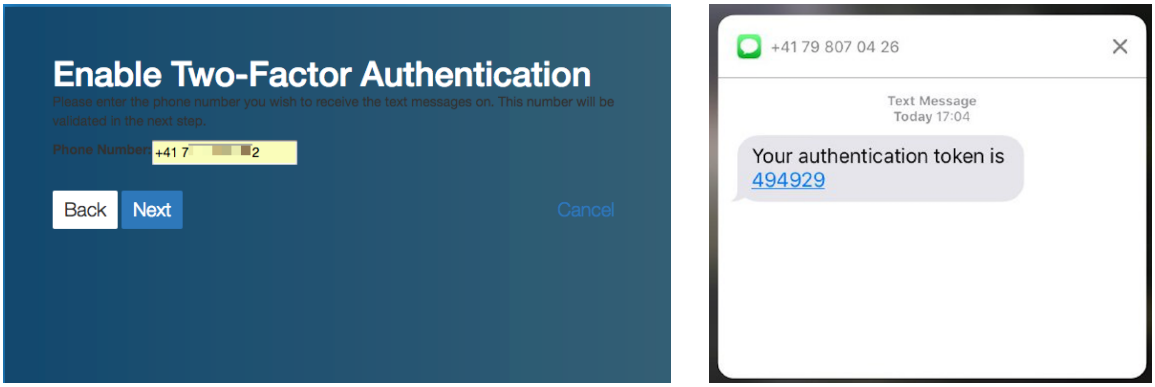


(a) The user credentials are entered to create an account.



(b) After the account is created, the user receives a confirmation mail.

Figure 7.2: The user registration process of Pharos.



(a) A phone number is entered to set up the two-factor authentication.

(b) A code is sent to the phone to verify the provided number.

Figure 7.3: The set up of the two-factor authorization

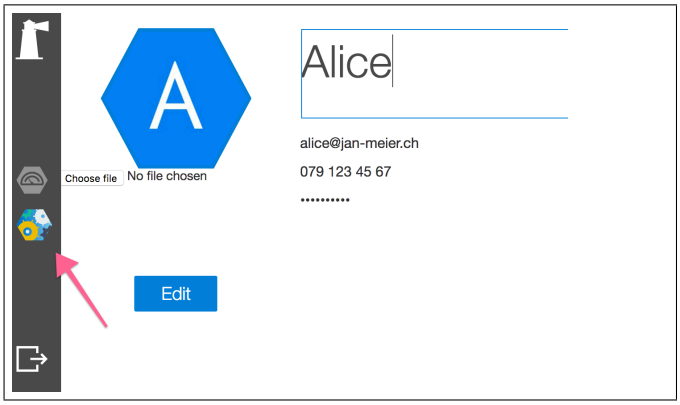


Figure 7.4: The user details are edited in a separate tab.

Once the account is created, the user details can be edited in a separate tab, as shown in Figure 7.4. This operability addresses the request for a **simple UI for account** configuration.

The request for an **easy tag set up process** was already addressed when the user starts the registration by pressing the "Register Tag" button in the dashboard view, which can be seen in Figure 7.1b. Next, the user enters a name for the tag and the tag's id. An optional image can be added to the tag, by dragging an image to the designated area. Once the form is properly filled, the user completes the tag registration process by pressing the "Create" button. The dialog will disappear, and the new tag immediately appears in the dashboard's tag list.

The user can open a tag's detail view by clicking on a tag list item in the dashboard view in order to configure the **location-based alarm conditions**. First of all, a tag's alarm settings are only modifiable when a tag is deactivated. The tag shown in Figure 7.6 is deactivated, as indicated by the "Off" button on the top of the image. The user can set the center of the circular Geofence either by providing the coordinates or by pressing the "Current Position" button. The later option sets the center to the tags current position. The circle's radius is then defined with the slider shown on the bottom of the screenshot.

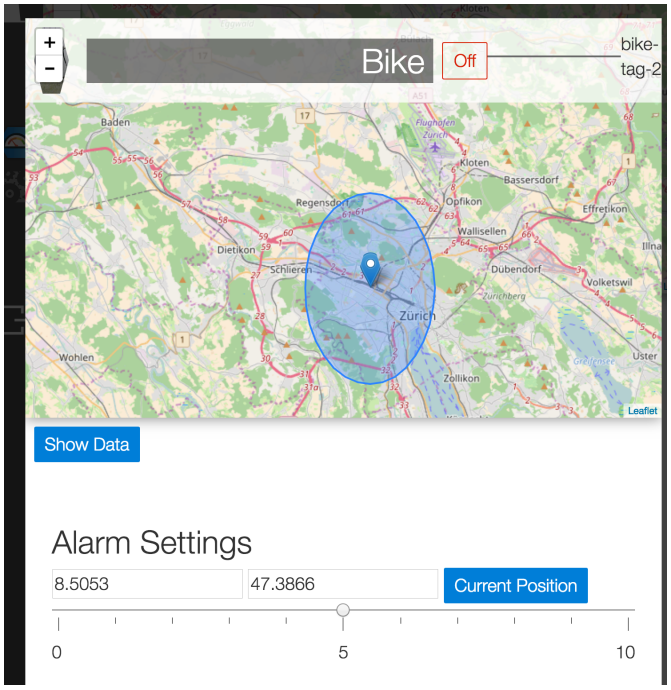


Figure 7.6: A tag's detail view with the configuration panel to define the alarm settings.

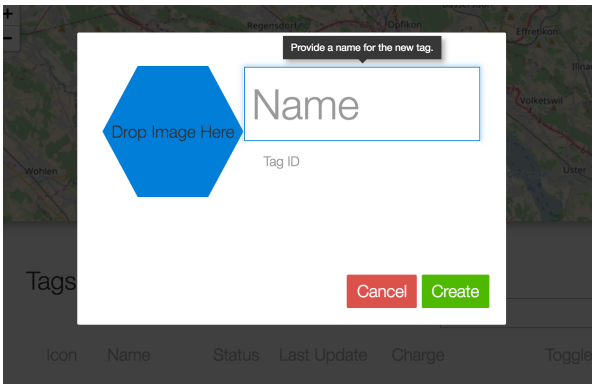
In the login screen of the app, there is a link to **reset a user account** as shown in Figure 7.7a. The user then has to enter the account's email address (see Figure 7.7b), and Pharos will send an email (see Figure 7.8a) with a link to reset the password. The form for setting the new password is shown in Figure 7.8b.

Pharos allows disabling the tags in the dashboards tag list in oder to support the user with a **fast tag management** opportunity. Figure 7.1b shows a tag, which is currently disabled. By pressing the red "Off" button, the user can activate the tag.

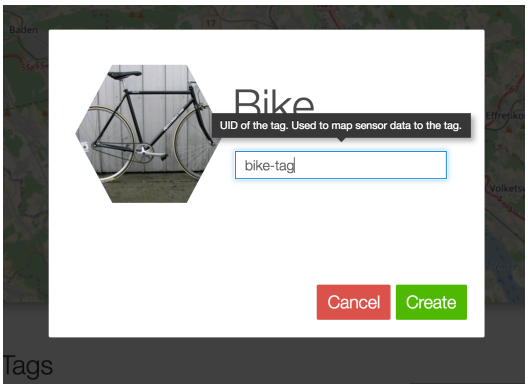
The **reporting of the tags signal strength** has not yet been implemented, even though the values were available. The tag's GPS module reports the signal strength of the positioning and the TTN platform provides the signal strength of the LoRa signal. However, the tag so far does not send the GPS signal strength on purpose to minimize the time required to transmit its position. As mentioned, TTN limits the air time per LoRa device to 30 seconds a day.

The back-end and the UI provide information about the available **power level of the tag** and display that to the user. Figure 7.9 shows a screenshot of this bar. However, the charge level is so far only a symbolic value, as the tag is not yet capable of reporting it the real charge level to the back-end.

A tag's detail view has a control panel to **share the tag**, which can be seen in Figure 7.10. Once a tag is shared with a user, the tags also can be "unshared" by pressing the "Delete" button, which removes the corresponding entry from the list. Pharos displays the shared tags in an extra section in the dashboard view, below the regular tags (see Figure 7.11). The shared tags are also displayed on the overview map. Shared tags are so far read only entities and have no detail view enforcing privacy.

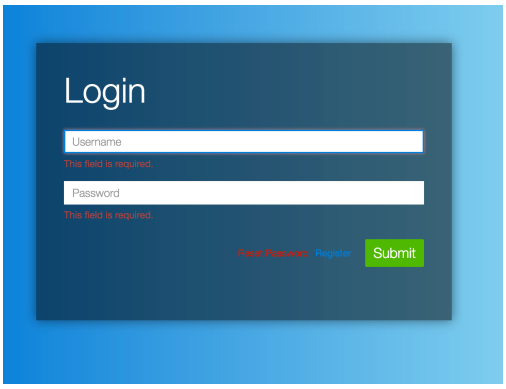


(a) An empty tag registration form

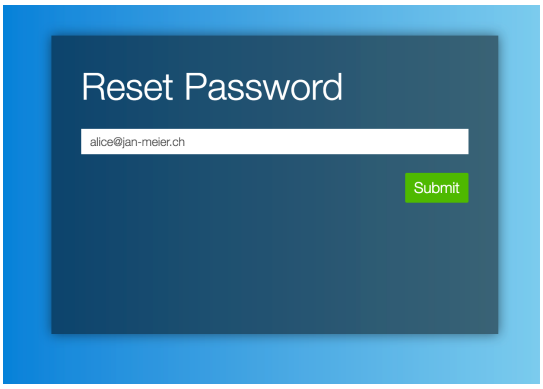


(b) The tag registration form before the registration is confirmed

Figure 7.5: Tag registration process

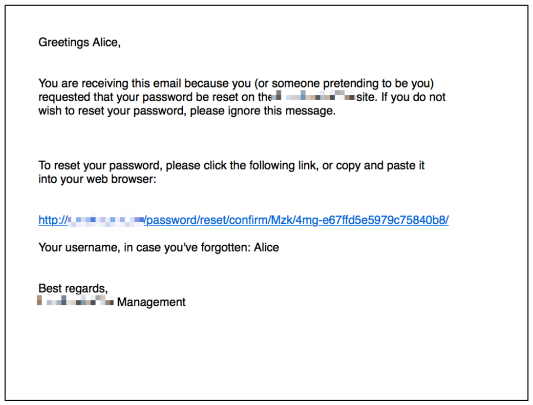


(a) After an invalid login attempt on the home screen the user has the option to reset the password.

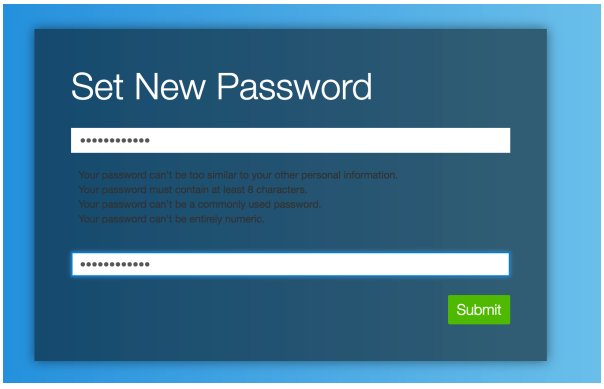


(b) An email address has to be provided to reset the password.

Figure 7.7: Password reset initiation by user.



(a) The user receives an email with a link to reset the password.



(b) The user has to provide a new password.

Figure 7.8: Screenshots showing step 3 and 4 of the password reset workflow.

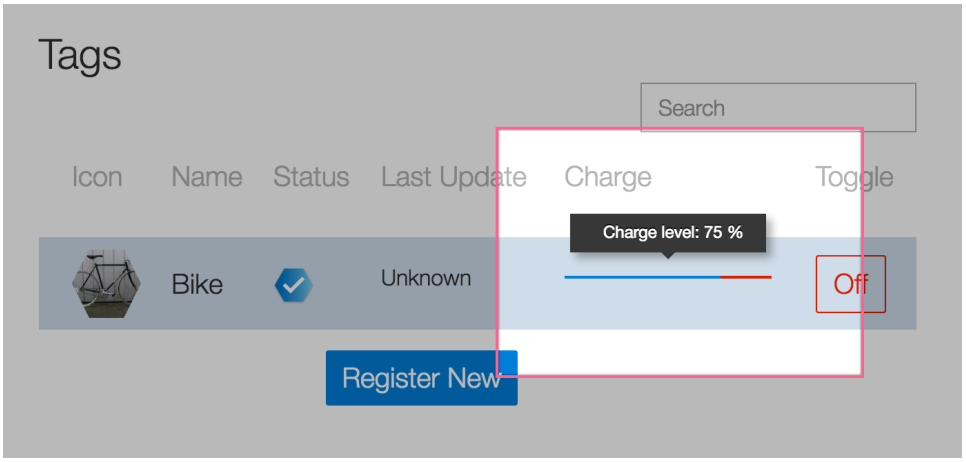


Figure 7.9: Bar indicating a tag's charge level

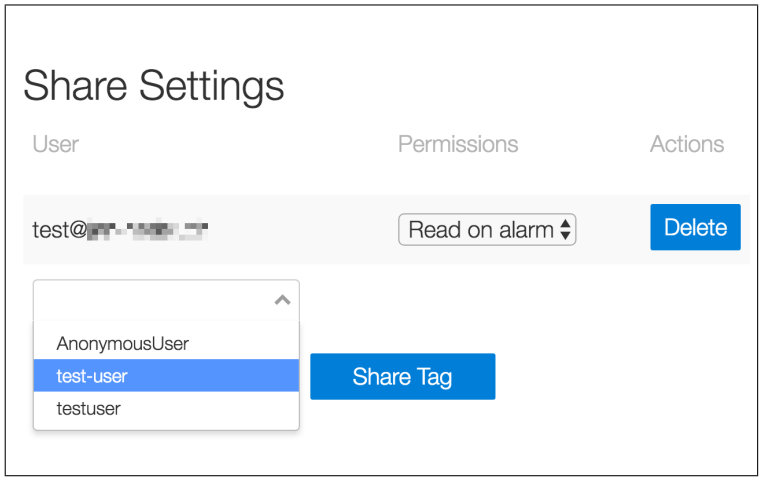


Figure 7.10: Tags can be shared with other users of Pharos.

Figure 7.11 shows a tag outside of its Geofence call the **alarm handling procedure**. According to the specification, this triggered an alarm. The owner of the tag had specified to be informed by SMS message and email. Figure 7.13 shows the sent messages. The messages both contain a link, secured with a one-time code, to cancel false alarms. The email message also contains a link to directly confirm the alarm. The SMS message has no "confirm" link as its message size is limited to 160 characters.

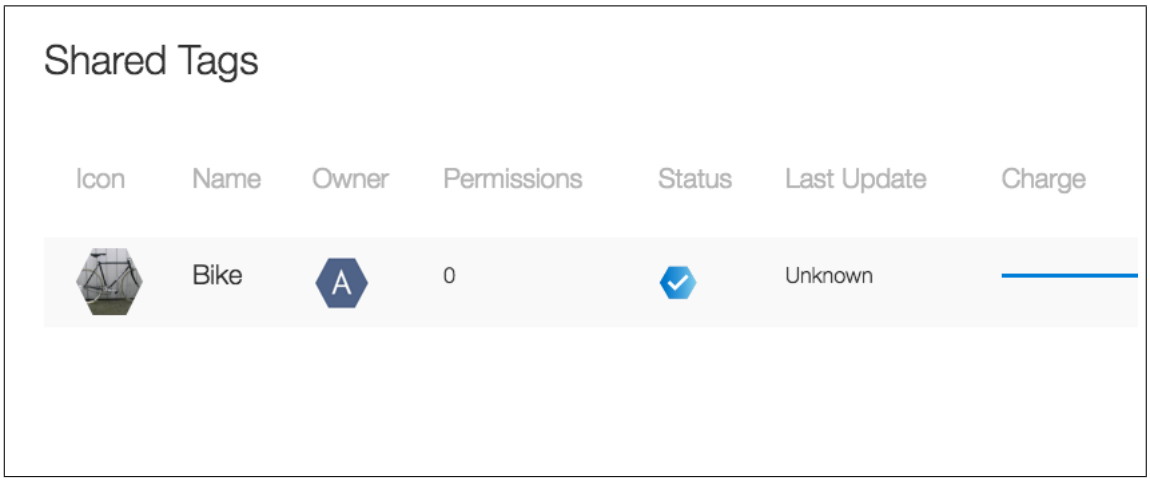


Figure 7.11: Shared tags are displayed on the dashboard below a user's regular tags.

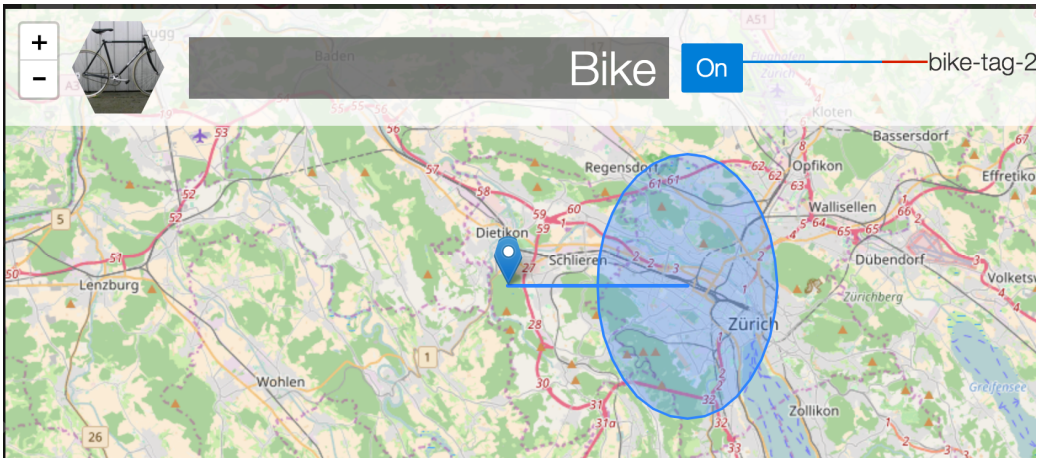
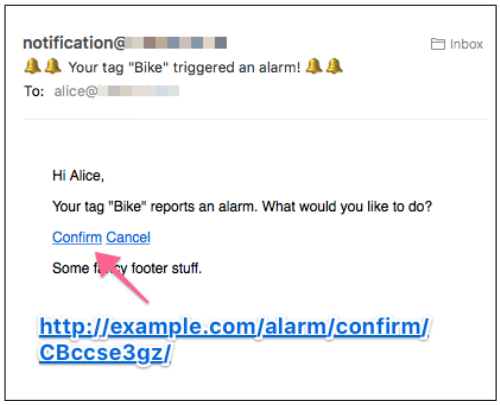
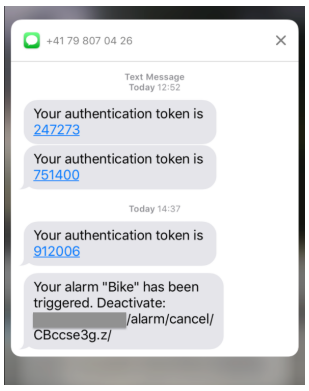


Figure 7.12: A tag's position has been reported outside it's Geofence.



(a) Announcement via Email



(b) Announcement via SMS message

Figure 7.13: Notifications types for triggered alarm.

Chapter 8

Conclusion

This master's thesis presented a web app-based OTS. The goal of this thesis was to gather the requirements for an OTS, derive a software design from the requirements and provide a prototypical implementation of the system. These goals have been achieved throughout this work. However, some of the identified requirements and part of the design could not be implemented in the prototype.

The requirements for development of an OTS were derived from three resources: an examination of existing tracking solutions, a stakeholder analysis, and a user scenario. The first sections of Chapter 4 summarized the key insight from these three resources which and Section 4.4 inferred an extensive list of requirements. Thereby, the first major goal of this thesis was met. Chapter 5 created a software design based on the previously determined requirements. The resulting design follows an SOA for IoT, which is a software architecture with four layers. This layered architecture allows Pharos to separate its hardware, software and user interface aspects and thereby achieved a high flexibility and extendability. These two requirements were of high importance for Pharos, due to its evolutionary nature. This software design satisfied the second goal.

Pharos' proposed design tackles many of the issues which are not addressed by the OTS solutions listed in Section 3.1. Many of these systems do not make security and privacy considerations. Pharos' requirements on the contrast recognized them as critical issues and the proposed design provides effective means to assure them. One of these means is Pharos' fine-grained privilege management. The academic papers about OTS do not mention usability issues. Pharos, on the other hand, considers usability highly important and aims to provide the user an appealing and effective UI. Another issue the mentioned paper do not elaborate upon is the software architecture. Pharos addresses this topic with great care and presents a sophisticated system architecture and a carefully planned DevOps setup.

The third goal is met by the prototypical implementation which is described in Chapter 6. The evaluation in Chapter 7 showed that most of the stated requirements were met. However, throughout the implementation and the evaluation, some limitations became visible which should be addressed in a future improvement of the prototype. The three major limitations are elaborated below:

1) The Pharos prototype outlined how a secure OTS web app can be implemented. Pharos' REST API was designed and implemented with the threats mentioned by the OWASP Top 10 list in mind and thereby lies the foundation for are a truly secure web app. However, a rigorous examination of the existing code base and the provided APIs has to assure that Pharos is provably secure or to locate components, whose security has to be improved. Further, the design mentioned, that the back-end implementation assumes a secure hosting. An improvement of the Pharos OTS should be assessed if there are means to overcome this bold assumption.

2) The developed tag is a mere proof of concept, which showed that the chosen technologies are feasible options. However, the tag is too large for real-world usage, and neither the power efficiency nor the reliability has been systematically tested. The current prototype further is not implemented to receive messages from the back-end, which in turn also implements the required functionality.

3) The requirements defined a fine-grained authorization management as a requirement. Pharos showed with the tag access right management that this is feasible with the presented setup. However, the tag management is not as fine-grained as defined by the requirements. For instance, the tags only have the three permission levels *None*, *Read*, *Read/Write*. Special cases such as "Read on Alert" or limited write permissions are not yet implemented. A future improvement of the Pharos prototype should again analyze the use case of an OTS, specify these permission classes in-depth and extend Pharos' authorization management accordingly.

Bibliography

[1] Bundesamt fuer Statistik (BFS), "Polizeiliche Kriminalstatistik (PKS), Jahresbericht 2015," Eidgenoessisches Departement des Inneren EDI, Bundesamt fuer Statistik (BFS), Tech. Rep., 2016. [Online]. Available: https://www.bfs.admin.ch/bfs/de/home/statistiken/kriminalitaet-strafrecht/erhebungen/pks.html

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] "TrackR," https://thetrackr.com/, [Online: accessed 30-May-2017].

[4] "FindMe," http://www.findme-tracker.ch/, [Online: accessed 30-May-2017].

[5] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[6] "Arduino," https://www.arduino.cc/, [Online: accessed 18-May-2017].

[7] E. Rescorla, "HTTPS over TLS," Internet Engineering Task Force, Tech. Rep. RFC 2818, 2000. [Online]. Available: https://tools.ietf.org/html/rfc2818

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol - http/1.1," Internet Engineering Task Force, Tech. Rep. RFC 2616, 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2616

[9] "Postgres," https://www.postgresql.org/, [Online; accessed 15-May-2017].

[10] "Web frameworks list," https://github.com/showcases/web-application-frameworks, [Online: accessed 24-May-2017].

[11] "Django," https://www.djangoproject.com/, [Online; accessed 24-May-2017].

[12] "Docker," https://www.docker.com/, [Online: accessed 21-May-2017].

[13] "REST API Tutorial," http://www.restapitutorial.com/, [Online: accessed 18-May-2017].

[14] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures.* University of California, Irvine, CA, USA, Doctoral dissertation, 2000.

[15] "JSON," http://www.json.org/, [Online: accessed 18-May-2017].

[16] "nginx," http://nginx.org/, [Online; accessed 15-May-2017].

[17] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[18] ITU-T: Overview of the Internet of Things. ITU-T Recommendation Y.2060, International Telecommunication Union, Geneva, Switzerland, Jun. 2012, URL: http://www.itu.int/itu-t/recommendations/rec.aspx?rec=Y.2060, last access May 10, 2017.

[19] R. Davis: The Internet of Things - Opportunities and Challenges. European Parliament Briefing, Briefing, Europe and Parliamentary Research Service (EPRS), PE 557.012, May 2015, URL: http://tinyurl.com/eu-briefing, last access May 10, 2017.

[20] Sundmaeker, Harald and Guillemin, Patrick and Friess, Peter and Woelfflé, Sylvie, "Vision and challenges for realising the Internet of Things," *Cluster of European Research Projects on the Internet of Things, European Commission*, 2010. [Online]. Available: http://www.internet-of-things-research.eu/

[21] R. Mautz, *Indoor positioning technologies.* ETH Zurich, Department of Civil, Environmental and Geomatic Engineering, Institute of Geodesy and Photogrammetry Zurich, ZH, Switzerland, Doctoral dissertation, 2012.

[22] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 6, pp. 1067–1080, 2007.

[23] D. Bachfeld, "Funk für maker," 2016.

[24] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Computer Networks*, vol. 67, pp. 104–122, 2014.

[25] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 60–67, 2016.

[26] M. Langheinrich, "Privacy by design-principles of privacy-aware ubiquitous systems," in *International Conference on Ubiquitous Computing.* Springer, Heidelberg, Germany, 2001, pp. 273–291.

[27] J.-H. Hoepman, "Privacy design strategies," in *IFIP International Information Security Conference.* Springer, Heidelberg, Germany, 2014, pp. 446–459.

[28] P. Schaar, "Privacy by design," *Identity in the Information Society*, vol. 3, no. 2, pp. 267–274, 2010.

[29] A. Cavoukian, "Privacy by design leading edge," *IEEE Technology and Society Magazine*, vol. 31, no. 4, pp. 18–19, 2012.

[30] C. Eckert, *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Walter de Gruyter, 2013, ISBN: 978-3486578515.

[31] T. Fox-Brewster. (2016) Yahoo: Hackers stole data on another billion accounts. [Online]. Available: http://www.forbes.com/sites/thomasbrewster/2016/12/14/yahoo-admits-another-billion-user-accounts-were-leaked-in-2013/,Accessed: 07-February-2017

[32] P. Beuth. (2017). [Online]. Available: http://www.zeit.de/digital/datenschutz/2017-02/ransomware-kreative-erpressung-herzschrittmacher-autos-bundestagswahl

[33] T. OWASP, "Top 10–2013," *The Ten Most Critical Web Application Security Risks*, 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10

[34] R. K. Banyal, P. Jain, and V. K. Jain, "Multi-factor authentication framework for cloud computing," in *Fifth International Conference on Computational Intelligence, Modelling and Simulation*. IEEE, New York, NY, USA, 2013, pp. 105–110.

[35] S. Porteck. Unzuverlaessige helfer: Bluetooth-findehelfer funktionieren nicht wie beworben. [Online]. Available: http://tinyurl.com/tracker-fails,Accessed: 20-February-2017

[36] "Lapa Bluetooth Finder," https://findlapa.com/, [Online: accessed 30-May-2017].

[37] "iTraq," https://www.itraq.com/, [Online: accessed 30-May-2017].

[38] "Ping," http://tinyurl.com/n64dp4k, [Online: accessed 30-May-2017].

[39] T. Pavithra and K. S. Ravi, "Anti-loss key tag using bluetooth smart," *Indian Journal of Science and Technology*, vol. 10, no. 4, 2017.

[40] Ramani, R and Valarmathy, S and SuthanthiraVanitha, N and Selvaraju, S and Thiruppathi, M and Thangam, R, "Vehicle tracking and locking system based on GSM and GPS," *International Journal of Intelligent Systems and Applications*, vol. 5, no. 9, p. 86, 2013.

[41] K. Rohitaksha, C. Madhu, B. Nalini, and C. Nirupama, "Android Application for Vehicle Theft Prevention and Tracking System," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 3754–3758, 2014.

[42] T. Tsung-Te Lai, C.-Y. Lin, Y.-Y. Su, and H.-H. Chu, "Biketrack: Tracking stolen bikes through everyday mobile phones and participatory sensing," in *Proceedings of the 2nd International Workshop on Sensing Applications on Mobile Phones . ACM, New York, NY, USA*, 2011.

[43] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 2012, ISBN: 978-0321815736, 2012.

[44] M. B. Rosson and J. M. Carroll, "Scenario based design," *Human-computer interaction. Boca Raton, FL*, pp. 145–162, 2009.

[45] J. M. Carroll, "Five reasons for scenario-based design," *Interacting with computers*, vol. 13, no. 1, pp. 43–60, 2000.

[46] "ArduinoUNO," https://www.arduino.cc/en/Main/ArduinoBoardUno, [Online: accessed 18-May-2017].

[47] L. Alliance, "LoRaWAN Specification," 2015. [Online]. Available: https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf

[48] "Swisscom Low Power Network," http://lpn.swisscom.ch/e/why-lpn/, [Online: accessed 18-May-2017].

[49] "The Things Network," https://www.thethingsnetwork.org/, [Online; accessed 15-May-2017].

[50] F. Reclus and K. Drouard, "Geofencing for fleet & freight management," in *9th IEEE International Conference on Intelligent Transport Systems Telecommunications,*. IEEE, New York, NY, USA, 2009, pp. 353–356.

[51] "Freaduino UNO," https://www.elecfreaks.com/wiki/index.php?title=Freaduino_UNO, May 5, 2015, [Online; accessed 13-May-2017].

[52] "Dragino LoRa Shield," http://www.dragino.com/products/module/item/102-lora-shield.html, [Online; accessed 13-May-2017].

[53] "GY-NEO6MV2 GPS Module," http://shop.boxtec.ch/neo6mv2-gps-module-p-42735.html, [Online; accessed 13-May-2017].

[54] "PostGIS," http://postgis.net/, [Online; accessed 15-May-2017].

[55] "GeoDjango," https://docs.djangoproject.com/en/1.11/ref/contrib/gis/, [Online; accessed 15-May-2017].

[56] "Celery," http://docs.celeryproject.org/, [Online; accessed 15-May-2017].

[57] "Redis," https://redis.io/, [Online; accessed 15-May-2017].

[58] "django-registration-redux," https://github.com/macropin/django-registration, [Online; accessed 15-May-2017].

[59] "Django Two-Factor Auth," https://github.com/Bouke/django-two-factor-auth, [Online; accessed 15-May-2017].

[60] "Twilio," https://www.twilio.com/, [Online; accessed 15-May-2017].

[61] "Let's Encrypt," https://letsencrypt.org, [Online; accessed 15-May-2017].

[62] "npm," https://www.npmjs.com/, [Online; accessed 15-May-2017].

[63] "Sass," https://vuejs.org/, [Online; accessed 16-May-2017].

[64] "Gulp," http://gulpjs.com/, [Online; accessed 17-May-2017].

[65] "webpack," https://webpack.js.org/, [Online; accessed 17-May-2017].

[66] "yarn," https://yarnpkg.com, [Online; accessed 17-May-2017].

[67] "axios.js," https://github.com/mzabriskie/axios, [Online: accessed 17-May-2017].

[68] "Leaflet," http://leafletjs.com/, [Online: accessed 17-May-2017].

List of Figures

List of Tables