University of Zurich[UZH]

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

Securing Goods Distribution with Smart Contracts and Sensors

Andreas Knecht
Zurich, Switzerland
Student ID: 11-916-152

Supervisor: Thomas Bocek, Andri Lareida
Date of Submission: October 21, 2016

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

ifi

Bachelor 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

New regulations in Switzerland and the European Union mandate the recording of temperature measurements for all pharmaceutical shipments. A solution is proposed herein that uses sensors to record temperature measurements and a smart contract to validate those measurements, as well as immutably store a hash of the measurements and the validation decision. A combination of pilot project analysis of a prototype system and other qualitative and quantitative analyses are used to improve the prototype system and assess usability for production usage.

Zusammenfassung

Neue Gesetze in der Schweiz und der Europäischen Union erfordern das Aufzeichnen von Temperaturen auf allen Transporten von Medikamenten. Eine Lösung wird präsentiert, die mit Sensoren Temperaturen aufzeichnet und diese in einem Smart-Contract validiert, sowie die Validierungs-Entscheidung und einen Hash der Messungen unveränderbar abspeichert. Durch Analyse der Resultate von Pilotprojekten mit einem Prototyp-System, sowie weiteren qualitativen und quantitativen Analysen wird das Prototyp-System verbessert und die Brauchbarkeit für Produktionseinsatz evaluiert.

Acknowledgments

I want to thank Thomas Bocek for his support during the entire thesis and Bruno Rodrigues for the valuable feedback.

Contents

Chapter 1

Introduction

1.1 Motivation

In 2016 regulations concerning medical shipments have changed in Switzerland (and the EU) and it is now required for temperatures to be recorded on all medical shipments and deviations to be reported to the sender and receiver of the shipment [1, 2]. In response, multiple pharmaceutical distributors in Switzerland started using temperature-controlled transports, i.e., air-conditioned trucks, for all shipments including those with non-temperature sensitive medical goods. This increases regulations overhead, increases costs and motivates the creation of a cheap and easy solution to record temperatures for pharmaceutical shipments.

Recording temperatures using sensors is cheaper than maintaining a certain temperature using an air conditioning unit, when the medicines being shipped are not sensitive to high or low temperatures. Implementing a risk-based approach, pharmaceutical companies can decide whether to ship medicines using non-temperature-controlled methods and destroy the small percentage of shipments where the logs show temperatures exceeded the required thresholds. Using temperature loggers, such as internet of things (IoT) sensors, also allows pharmaceutical companies to verify and monitor temperature controlled transports and areas such as warehouses to assess the effectiveness of temperature control systems in a given area.

The solution described in this thesis uses sensors that are read out via Bluetooth Low Energy (BLE), as opposed to USB that many existing loggers employ. This allows for time savings in handling the sensors, including not having to unpack the parcel containing the sensor.

The solution described in this thesis uses a smart contract in the Ethereum blockchain to verify that the recorded temperatures have met regulation requirements. The idea of smart contracts is to have a protocol or code representing a contract that is self-executing, making a contractual clause and the inclusion of a trusted third party, like a notary service, unnecessary by exchanging it with the consensus system provided by the blockchain. Thus, the information whether or not regulation compliance has been fulfilled

does not require manual handling. At the same time the measurements are archived and a hash thereof stored in the smart contract, enabling manipulation of the measurements to be detected and measurements to be available in an audit. Because the smart contract is public in the blockchain, it can be verified for correctness by the multiple parties involved, including pharmaceutical companies and regulation agencies. Also, smart contracts in the Ethereum blockchain enforce access rights to their state data, thus the smart contract's decision cannot be changed or manipulated if the contract is designed accordingly.

1.2 Description of Work

The goal of this thesis is to develop a working prototype system aimed at the temperature monitoring use-case. Some components of the system are simultaneously developed by Strasser [32]. The system includes a sensor that records and stores temperatures, a smartphone app that allows reading out the sensor and uploading the measurements to a smart contract, a server program that keeps all the gathered information in a database and the smart contract.

The development of the sensor device includes selecting a suitable hardware platform, programming and calibrating it.

The system is evaluated for production usability in two pilot projects. During these projects weekly shipments between pharmaceutical companies, wholesalers and hospitals are monitored for temperature with a prototype system and the accuracy, stability and usability of the system evaluated.

1.3 Thesis Outline

Chapter 2 (Background & Related Work) describes blockchain technology, smart contracts and platforms that enable such smart contracts, as well as the choices of sensors for the project and the selection process. Chapter 3 (Design) describes the design and some important requirements of the system and its components. Chapter 4 (Implementation) describes implementation choices for the different components of the system. Chapter 5 (Evaluation) evaluates the system's usability and other aspects from the pilot project results, as well as analyses of other data.

Chapter 2

Related Work

2.1 Blockchain

A blockchain is a decentralized data structure maintained over a peer-to-peer network or other distributed system, consisting of a list of blocks, in which every participating node can append new blocks containing records of transactions. Therefore, to control the inclusion of new blocks to the blockchain, consensus algorithms regulate the state the of the chain when multiple nodes want to append conflicting information to the end of the blockchain simultaneously [7, 18]. That is, the end of the chain may sometimes be ambiguous, but the consensus algorithms ensure that the state of the chain becomes consistent, i.e., avoiding forks in the chain. The goal of a blockchain is to reach consensus over a non-conflicting state encoded in the blockchain without having to trust the other nodes in the network. There are two algorithms for reaching consensus over the state of the blockchain. Proof-of-work (PoW) and proof-of-stake (PoS). PoW makes nodes prove investing an amount of work, *e.g.*, by solving a mathematical puzzle, before they can append their block to the end of the blockchain [7, 19]. Given the probability distribution of the time for committing the work for a block, multiple nodes finding blocks simultaneously multiple times in a row becomes increasingly unlikely. Thus, consensus is eventually reached. PoS makes nodes prove they have a stake in the system, *e.g.*, by proving they own cryptocurrency associated with the blockchain, in order to contribute a block [20].

The main advantage of a blockchain is to protect the data hosted in the blockchain from malicious alteration, only allow valid manipulations on the data and build consensus when multiple opinions on the state of the blockchain exist. This property comes from the fact that the blockchain is built and maintained by an immense amount of processing power shared between many peers. The processing power that all peers have to contribute must be such that any malicious single peer or group of peers cannot attain 51% of the network's processing power without investing an unviable amount of money. A user does not necessarily have to contribute processing power to a blockchain to use it, which makes it interesting to use cheap devices that only record parameters of shipments and push them to the blockchain where the data will be secured by all the peers, thus making the investment in hardware for the use case minimal. However, transaction fees are imposed and it is important to find or create a cost efficient smart contract.

These properties of immutability and the inability for any single party to manipulate transactions make the blockchain interesting for this project.

2.2 Smart Contracts

A smart contract is a program stored and executed on the blockchain with the purpose of providing the logic of a real-world contract [6, 9, 10]. Being blockchain-based, smart contracts are executed and verified by all miners participating in the given blockchain and it is thus almost guaranteed that the smart contract produces the result mandated by its code without the user having to put trust in individual participants of the blockchain network. Smart contracts can cause ledger updates on the blockchain they run on and thus cause automatic moving of funds in response to certain events and based on predefined conditions. Furthermore, smart contracts cannot interact with any application programming interface (API) outside the blockchain network for security reasons and require special smart contracts in the network called oracles that receive and relay information about the outside world. Smart contracts can and have been used as a substitute for legal contracts in the real world and can, thus, be seen as self-fulfilling automated versions of real-world contracts.

One advantage is cost because it can be chaper to employ smart contracts than legal contracts, especially when lawyer costs or costs of independent trusted third parties are incurred to enforce a real-world contract. Another advantage is that the smart contract code is a program and thus is unambiguous compared to natural language in which legal contracts are written. Because the smart contract code is public on the blockchain. anyone who knows how to read the code can verify it for correctness and absence of security vulnerabilities (intentional or unintentional).

One disadvantage of smart contracts is the risk of security vulnerabilities. Because the contract is a computer program, it will execute exactly what was written in the program, even though this may turn out to be in the interest of neither party participating in the contract. With a real-world contract, fights over its interpretation can be taken to court and the parts of the contract that violate the law even found to be invalid. One could argue that a smart contract cannot be hacked, only exploited if the contract code was written without enough consideration, and secure smart contracts still a better alternatie to real-world contracts in many situations. Smart contracts are used for a variety of applications, which include:

Escrow services Hold money between two parties and release it to the intended receiving party when a condition is met – such as a product shipped or service rendered – or returning it to the sending party when the condition isn't met [5].

Automated gambling platforms Allow people who verified the smart contract for absence of scam potential to gamble without having to trust the operator of a gambling site.

Financial contracts Allow the banks to trade directly without having to employ a trusted third party that doesn't favor the interest of any one bank.

Smart property Track property on the blockchain and can enable loans that do not require trust or a bank [4].

Custom cryptocurrency Create custom cryptocurrency tokens with the properties of the currency – such as the nature of creation of new tokens and additional features such as destruction of stolen coins by an administrator – verifiable by everyone who reads the smart contract code.

Voting schemes There have been ideas and projects that attempt to enforce political (democratic) or organizational (shareholder based) voting schemes in a smart contract. One high-profile case, the DAO (Distributed Autonomous Organization), failed with a lot of media coverage when it was hacked. The DAO hack also affected the project described in this work as explained in Section 6.2 [11, 12].

2.3 Ethereum

Ethereum is a blockchain-based platform whose main purpose it is to allow for the creation of arbitrary smart contracts by providing a virtual machine that can run Turing complete programs [3, 21, 23, 24]. Thus, Ethereum can be used more generally than other blockchain-based platforms. Every node verifying the blockchain has to execute all submitted smart contracts to ensure that the state recorded in the blocks actually matches the local state changes from executing the smart contracts.

Multiple implementations of Ethereum exist, including ones written in Go[1], Python[2], Java[3] and C++[4]. The implementation in Go is the farthest in terms of development and also supported by the Ethereum Foundation, thus the most likely to be well-supported in the future. It was decided to implement the server component of the system in Go because bindings for the remote procedure call to geth were already available for the Go language [23].

The best supported language for writing Ethereum smart contracts is Solidity[5]. Other languages exist, such as Serpent[6] and LLL, the former being deprectated in favor of Solidity and the latter being a rather low-level language and not actively developed. Solidity was chosen for the project because the language and its compiler are the most stable and furthest developed option for Ethereum.

Ethereum incorporates its own cryptocurrency named Ether (abbreviated ETH) [22]. Smart contract executions have to be paid for in Ether. Similar to Bitcoin, Ether are awarded to miners contributing blocks and are also traded for other currencies on online exchange sites. Each operation in the Ethereum virtual machine has a cost that is declared in a unit called gas. Since predicting the cost of a smart contract before running it is

[1] https://github.com/ethereum/go-ethereum
[2] https://github.com/ethereum/pyethereum
[3] https://github.com/ethereum/ethereumj
[4] https://github.com/ethereum/cpp-ethereum
[5] http://solidity.readthedocs.io/en/latest/
[6] https://github.com/ethereum/wiki/wiki/Serpent

impossible, analogous to the halting problem, the originator of a smart contract execution has to declare a limit of gas that he is willing to spend (gas limit), as well as the amount of Ethers he will pay for each gas used. The originator can thus also prioritize contract execution by paying more Ethers per gas than the network average, which will likely result in faster mining of the contract. If the gas limit is used up for a contract execution, but the contract execution not finished, the state is rolled back to before the partial contract execution, but the originator still billed for the gas used. This is necessary to avoid the obvious denial-of-service (DoS) attack on the network where someone submits never-ending loops to the network as smart contracts, but doesn't have to pay for the execution.

Ethereum is backed by the Ethereum Foundation, which is registered in Switzerland and leads Ethereum's development [25]. This merits slightly more trust in the future development of a blockchain platform than those backed by anonymous or non-legal persons or organizations. However, the Foundation's course of action in the DAO hack tainted this trust.

2.3.1 The DAO Hack

The DAO was introduced as a smart contract-based platform for autonomous organizations where stakeholders could use tokens to buy into various organizations [11]. These tokens would then ensure a voting right in the organization to holders of the tokens. To ensure the voting rights and correct voting procedure, votes would be carried out over the smart contract. The matters on which token holders would be able to vote were part of the organization describtion submitted to the smart contract.

A security vulnerability in the DAO code allowed an attacker to drain funds valued at around 50 million USD from the DAO in June 2016 [12]. The contract code still locked these funds from withdrawal for about another month, giving the community time to discuss on how to proceed. Lastly, the Ethereum Foundation decided to push a hard fork, returning the stolen funds to the previous owners. A hard fork is a situation in which incompatible changes are made to the rules governing transactions in a blockchain, so that miners unaware of these changes will see transactions implementing these changes as invalid. Sometimes, transactions following the rules before the hard fork, but submitted after the fork, will be seen as invalid by miners following the hard fork. If the blockchain is associated with a cryptocurrency this spawns two new incompatible cryptocurrencies. Thus, there is a risk for people unaware of the fork, for example in accepting payments on the old chain and sending out goods, only to later realize that noone will accept these funds because the transaction is only part of the discontinued blockchain after the fork.

In general, transactions after a hard fork are only valid on one branch of the blockchain and changing the branch later will "revert" some of one's transactions. Thus, hard forks most often create a lot of anxiety in the participants of the network and undermine trust in the platform. This makes the exchange rates of the associated cryptocurrencies drop, which in turn can further undermine trust in the platform. Therefore, a hard fork should be a measure used only very sparingly, if at all, and only if a vast majority of the network's participants support the fork. If both those proposing and opposing a hard fork are large

enough groups, implementing the hard fork will cause two incompatible blockchains to live on, both with less mining power to support it than before the fork.

The Ethereum hard fork created this situation, spawing two new cryptocurrencies: Ethereum (also called Ethereum Core – the branch backed by the Ethereum Foundation) and Ethereum Classic [14, 15, 16, 13]. Many in the blockchain and cryptocurrency community oppose the hard fork because the concept of the smart contract being an undisputable enforcer of its code was violated in the hard fork. Also, many of the DAO investors were Ethereum Foundation members, thus making the decision to hard-fork one of their own interest, which has lead to tainted trust in the Ethereum Foundation. Also, trust in Ethereum in general was also tainted merely by the fact that a hack on Ethereum was reported by the media, despite the vulnerability for the DAO hack being in the smart contract code and not Ethereum code.

2.4 Other Blockchain Options

Bitcoin is a blockchain based cryptocurrency that existed since 2009 [7]. It was the first usable cryptocurrency and currently the largest in terms of market volume and network hashrate backing it up. Its implementation is very stable compared to many others. The Bitcoin Foundation that backs the development consists of known and trusted experts in the cryptocurrency field, and has proven in the past to make very reasonable and careful decisions concerning the future of Bitcoin. For example, they are very averse to forks and understand that any forking change in the Bitcoin protocol has to be backed by a vast majority of the community. Bitcoin transactions have a script associated which is usually deployed to determine who can claim the funds contained in the transaction [17]. Since the language of these bitcoin scripts is quite powerful, it also allows the creation of smart contracts. The language is, however, not Turing complete, and thus doesn't allow for the creation of arbitrary smart contracts, while a large number of use cases can still be covered in Bitcoin smart contracts. Also, there is no high-level language for Bitcoin script, such as Solidity for Ethereum.

IOTA is a platform backed by a blockchain-like data structure designed for smart contracts [8]. It is not associated with a cryptocurrency and contract execution doesn't have to be paid for in money. Instead to have a transaction executed one has to process two previous, unprocessed transactions. This leads to an acyclic graph of transactions where each transaction has two parents. This graph is called the tangle. Like with Bitcoin and Ethereum, conflicting transactions will eventually be orphaned. Their whitepaper shows mathematically that the number of tips of this tangle graph will remain somewhat stable and not increase indefinitely. The developers argue that IOTA is very suited to IoT applications because small, cheap IoT devices with limited processing power can participate in the network directly. The implementation of IOTA is, however, in an early stage and cannot be used for anything but experiments at this point.

2.5 Existing Solutions

Many existing loggers are read out via USB and produce a PDF document containing the recorded temperatures. The solution described herein uses sensors that are read out via Bluetooth Low Energy (BLE). This allows to save time in handling the sensors by not having to unpack the parcels containing the sensors, as well as reading out more sensors in parallel than would usually be the case via USB. The PDF produced by many sensors contains the measured temperatures but no automated assessment whether or not the temperatures contained comply with regulation requirements. With the system described in this work the measurements are automatically assessed by the smart contract and the result immutably stored. This results in time savings for the quality assurance (QA) manager responsible for discovering shipments that violated regulations compliance.

Other loggers can be programmed with user defined thresholds, but only record if and for how long the temperature exceeds the thresholds. Thus, the recorded temperatures cannot be extracted and archived. Thus, if a pharmaceutical company using such sensors is audited they cannot really prove regulations compliance because they do not have the archived measurements. The system developed in this thesis stores the recorded measurements in a Postgres[7] database and indirectly in the Ethereum distributed system (via the input arguments supplied to the temperature checking method). Also, a hash is stored in the blockchain, resulting in a manipulation-proof storage of the measurements.

Some existing loggers are single-use, i.e., they can only be used to record temperatures on one shipment, which results in a waste of resources, is not environmentally friendly and often expensive even if the piece price of a single-use sensor is low. The sensors used in this work can be reused, thus being more environmentally friendly and cheaper to employ.

2.6 Sensors

Generally, reading out the sensors could be done via a wired interface or wirelessly. The advantages of reading them out wirelessly include that the shipment containing a sensor do not have to be opened in order to read out the measurements, as well as the ability to read out multiple sensors faster because the physical process of unplugging one sensor and plugging in the next is eliminated and in addition multiple sensors can be read out simultaneously. Reading out the sensors wirelessly requires a reader device that interfaces with the sensors via radio frequency. In order to upload the measurements to a smart contract, the measurements will eventually have to be passed to an Internet-connected device. Another requirement for the reader device is that it should be small and portable to allow easy use in a warehouse. It was decided that a smartphone should be used as the reader device because of its capability of connecting to the sensor (via Bluetooth), as well as to the Internet, the general availability of smartphones and the convenience of not having to work on another hardware platform besides the sensor. For production use it is likely that special reader devices will have to be developed.

[7] https://www.postgresql.org/

The most basic requirement for the sensor device is that it should include a temperature sensor and battery socket on board without soldering anything onto the board for the most basic operation. The advertised battery life should be multiple months or longer. Further, Bluetooth Low Energy should be supported for an easy way to read out sensor data from a sensor contained in a parcel without having to open the parcel, while allowing the readout from a common device such as a smart phone. It should support a large enough MTU (maximum transmission unit) for the readout process to be rather quick ($<$ 10 seconds). The device should either store temperature measurements by default or be programmable to do so. The memory available to store measurements should be non-volatile and support storing at least 1000 measurements (which amounts to over 10 days of measurements at 1 measurement per 5 minutes). It should further allow setting a custom interval for temperature measurements and be able to store calibration coefficients to correct the measured temperatures. Developer tools should be available for either Windows or Linux. The development platform should ideally not be proprietary, allowing open-sourcing of source-code and potential use of extensions developed by third parties. Good support for the developers should be assured through either a large community or good documentation by the hardware manufacturer.

Especially the requirement for Bluetooth LE (Low Energy) eliminats most of the available sensor boards from the evaluation so that only a few devices needed to be considered.

The Oberon HAP [28] was not chosen because the developer tools seemed to be mainly written for Apple computers with unclear extent of Linux support and no support for Windows. Also Apple's HomeKit standard, which the project employs, is a proprietary standard and is slow over Bluetooth [30].

The BTnode [29] was discarded because it is a research board and the website didn't show any news more recent than 2008. Thus, it is unclear how mature the board is and whether the supplier or a community could provide a lot of support in case of problems.

The remaining two boards were ordered and tested.

The PunchThrough Bean [27] is Arduino [31] based and easy to program. It was discarded soon, since the available EEPROM non-volatile memory of 1KB is not enough for the project and the flashing procedure is tedious using a Windows 10 app. Also, the Bluetooth API offered by the platform is very simplistic and caused concern that there might be very little control over specific implementation details such as setting a larger MTU, inserting custom data in the advertise packet, setting a custom advertisement interval and transmission power.

Thus, the remaining platform, the TI (Texas Instruments) SensorTag [26], was chosen despite concerns about being very dependent on TI, since the developer tools, license and codebase are proprietary. However, during the project the community on the TI forum turned out helpful and all required features could be implemented in TI-RTOS (Texas Instruments-Real Time Operating System), which runs on the sensor device.

Chapter 3

Design

3.1 Components of the System

The system consists of the sensors running customized sensor software, a smartphone application, the server software and the smart contract (cf. Figure 3.1). The smartphone application interfaces with the sensor via BLE to start and stop the recording and read out the measurements, as well as setting and retrieving other parameters such as the shipment ID. The server software allows the smartphone application to indirectly create smart contracts in the Ethereum network and upload measurements. In addition, the server stores measurements, user accounts and shipment information in a Postgres database. The smart contract is responsible for verifying regulation compliance of the uploaded measurements and storing the result and a hash of the measurements immutably.

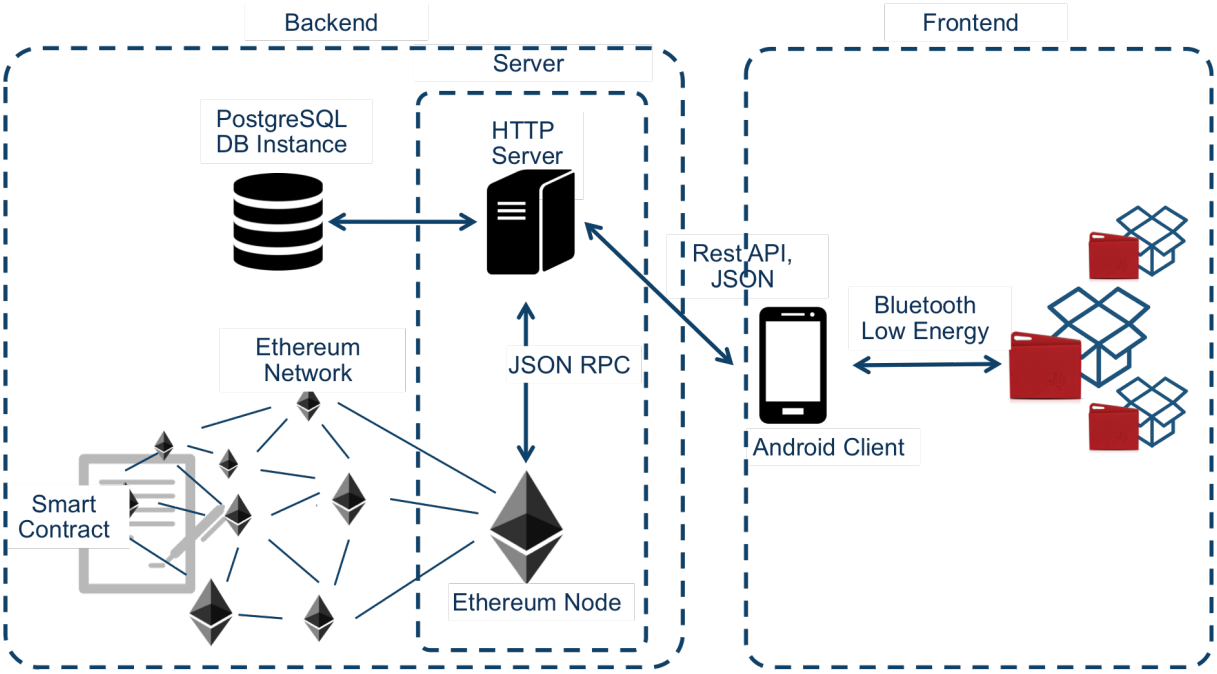

Figure 3.1: System Architecture Diagram

Due to the size of the blockchain and intense memory and CPU usage when running a full node, it is only viable to run the Ethereum light client protocol on a smartphone. Because the Ethereum light client protocol implementation has not been completed during this thesis, a centralized server was developed by Strasser [32] that runs a full Ethereum node and allows the smartphone app to interface with Ethereum smart contracts without running an Ethereum node itself.

When the Ethereum light client protocol development is concluded it is planned for the smartphone app to run an Ethereum light node and many core aspects of the system to be transferred to the smartphone app and smart contract, with the central server mainly serving archive purposes. The system is designed in a way that allows these changes to be implemented easily because the goal of eventually running an Ethereum light client node on the smartphone was determined from the beginning of the project.

3.2 Users of the System

The prototype system only has one interface to the user: the smartphone. All interactions with the sensors and shipments have to be provided by the app. This was a requirement to start the pilot projects as soon as possible. A production system, however, needs to provide appropriate user interfaces to the following users:

Shipping manager The person preparing the packing list for a parcel that knows the parcel's temperature requirements, destination, shipment duration and other variables. Ideally, this person enters all the required information for a shipment using a web or desktop application.

Warehouse worker The person packing the parcel and starting the sensor using the smartphone or special sensor interface device. Ideally, all the shipment's information have already been registered when the parcel is packed and the warehouse worker only needs to start the sensor and pack it.

QA manager The person responsible for the regulation compliance of the shipped products. This person needs a quick overview over failed and successful shipments with good search functionality. This person might as well be interested in aggregated data that help compare the quality of the shipping companies used.

Manager This person might be interested in reporting functionalities of the system, as well as key performance indicators that could be deduced from the data gathered by the system.

Developers The developers of the product require a good overview over all shipments, the current location of all sensors and strong data search functionality to provide quick support for the customers and investigate bugs.

Regulation agency They might want direct access to the gathered data in the future.

A web application is currently being developed to provide an appropriate user interface to these stakeholders. The ability to enter shipment parameters in the web application and use the smartphone application only to start and read out the sensor is a key requirement. Integration with the pharmaceutical distributors' existing enterprise resource planning (ERP) systems is also planned.

3.3 Blockchain Used

In the end the decision fell on Ethereum becasuse of the availability of the Solidity language to write smart contracts and its more mature development level. It is reasonably easy to transfer the smart contract to another platform if the Ethereum platform fails for some reason or is no longer deemed appropriate.

3.4 Shipping Process

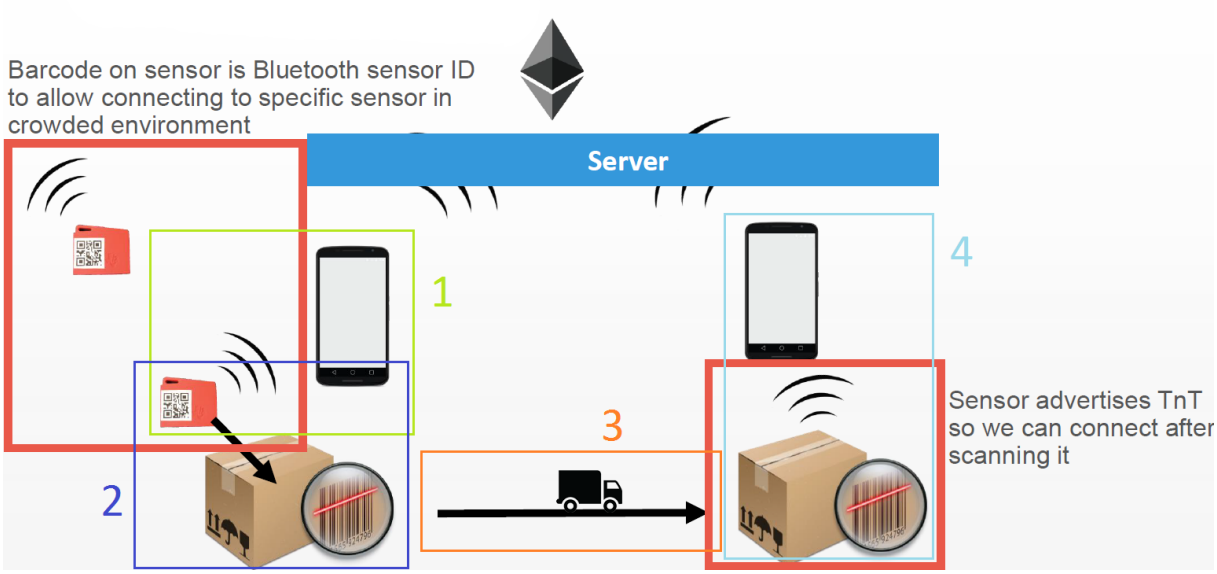

Figure 3.2: Illustration of the Process

The high level process of sending a temperature-monitored shipment is starting a sensor (1), placing it inside the shipment (2) and shipping it (3). At the destination the sensor is read out (4) and the measurements are uploaded to a smart contract (cf. Figure 3.2). The contract is designed so that the temperature range, that a shipment must stay within, is set when the contract is created and cannot be changed later. Thus, when the sender creates a smart contract for the shipment the range is fixed and cannot be changed by the receiver. Thus, the sender also learns the smart contract's address directly when creating it and no system has to be designed to notify him later. The sender can regularly check the smart contract to see if temperatures have been reported and whether or not the smart contract found them to comply with the previously set temperature range. At a

high level, the receiver needs to know the contract address and authorization to upload measurements to the smart contract. With the centralized server implementation that the project currently employs, the server holds the user account used to create the contract and upload measurements and the same account is used for both tasks. The server also keeps track of the created contracts. Authorization is still required for the receiver to upload measurements, but it differs from the authorization in the smart contract and is implemented on the server.

Figure 3.3: Two Sensors Labelled with Bluetooth Address as QR Code

The sensor's Bluetooth address is printed as a QR code on the sensor to allow connecting to a specific sensor after scanning its barcode (cf. Figure 3.3). This enables connecting to a specific sensor in an environment in which multiple sensors are advertising Bluetooth simultaneously. After starting the sensor the shipment's ID is part of the sensor's Bluetooth advertise packet to allow the receiver to connect to the sensor without having to open the parcel.

3.5 Sensor

This section describes design decisions for the sensor platform, including the encoding of measurements, how the internal state is communicated to the user using the on-board LEDs of the sensor platform and battery life requirements for the sensor software.

3.5.1 Storage of Measurements and Data

It was decided that the sensor should be able to record measurements in the interval [-10, 50]°C because the medicines shipped in the pilot projects all needed to be transported in temperatures ranges within that interval. For example, any temperature over 50°C does not need to be recorded exactly because it is destructive to the transported medicine anyway. In order store a large number of measurements it was decided to store each measurement in a byte, interpreted as equally spaced points in the range [-10, 50]°C. Thus, to encode a measurement temperatures are cropped below -10 and above 50 °C (by setting them to -10 and 50 °C respectively) and then $\frac{x+10}{60} \cdot 255$ where x is the cropped temperature measured. To decode a measurement $\frac{y}{255} \cdot 60 - 10$ is used where y is the encoded measurement.

Table 3.1 gives an overview over how long the sensor can record with different measurement intervals as limited by the space for measurements on flash (the sensor flash memory usage is explained in Section 4.1.1).

Measurement interval	**Recording duration capability**
1 minute	2.81 days
2 minutes	5.62 days
5 minutes	14.06 days
10 minutes	28.11 days
15 minutes	42.17 days
30 minutes	83.33 days
60 minutes	168.67 days

Table 3.1: Recording Duration Capabilities

3.5.2 Sensor Interface

The sensor presents its internal state to the user with a red and a green light-emitting diode (LED). The green LED flashes when the sensor is advertising Bluetooth. Since the sensor advertises at all times except when a Bluetooth connection is established, this is equivalent to the sensor being powered on and ready to accept connections. When the sensor's green LED is not flashing it is in a connected state or the battery is empty, or rarely it might have crashed into an undefined state. Even in the rare event of a crash, no LED glowing should be a rare outcome because the exception handlers are programmed to turn on the red LED to glow steadily. It cannot be completely ruled out, however. The red LED flashes when the sensor is recording, thus allowing the users to easily check if a sensor is recording and ready to be read out or stopped and ready for a new recording. As mentioned, the red LED glows steadily when the sensor software crashes into one of the defined exception handlers. It is not easy to communicate the internal state of a

device such as the sensor used in this project to the user using only two LEDs in three possible states – off, flashing, glowing – without overwhelming many users and risking user confusion. Such confusion occurred in the pilot projects and is explained in Section 5.1.

3.5.3 Sensor Battery Life

The battery that supplies the voltage to the sensor is a CR2032 coin cell battery. They typically have a capacity of around 225mAh. Which is limited compared to the capacity of a AAA battery which can go over 1000mAh. Thus, it was important that the sensor software consume the minimal amount of power required for operation.

In the early stages of the prototype the power consumption was unsatisfactory. The batteries lasted about one day. The reason turned out to be that, in an overzealous effort to cut all unnecessary code from the TI SensorTag example project that the sensor software is based on, the code that disables all sensors other than the temperature sensor was removed, resulting in all sensors being constantly enabled and drawing vast amounts of power. In the current implementation, even the temperature sensor is disabled between measurements, resulting in much greatly reduced power consumption.

Other parameters that potentially affectin battery life are the sensor's Bluetooth advertise interval and transmission power. A quantitative analysis of these parameters and resulting battery life is conducted in Section 5.3.

3.6 Security and Confidentiality

The current system needs security improvements for production deployment. Security was not a goal for this prototype system, focusing instead of being usable in the pilot projects to further evaluate its usefulness, requirements for the future and market. For example, with the current system, anyone with a login to the server and knowledge of an ongoing shipment's ID number can report temperatures for that shipment. Further, temperatures reported can be any constructed set of numbers and are not checked that they stem from one of the system's sensors.

Since the public Ethereum blockchain is used and recorded temperatures from the sensors uploaded to the smart contracts, anyone with access to the Ethereum network can recover these temperature measurements. Although the smart contract does not contain any information about the sender or receiver of a shipment, it is still possible to match contracts with shipments based on its metadata, such as duration of a shipment, and the date and time it was sent. Thus, confidentiality of the reported temperatures is fundamentally lacking in the system. Other security concerns might include:

DoS attacks Failing a parcel on purpose or with less gravity refusing to report a received parcel's temperature recording to the smart contract.

Sensor theft Sensors might be stolen and reused in a parallel malicious supply chain. Noticing theft or making them useless for a thief might thus be necessary.

Manipulation of sensors Such as extracting a private key from the sensor's memory or replacing the temperature sensor on the board with one that always reports favorable temperatures.

Upload of temperatures from wrong sensor This includes sensors that are valid, but not linked with a given shipment.

Uploading valid temperatures from a past shipment

Cloning of sensors To get additional sensors matching an existing sensor's ID, private keys and other characteristics.

Arguments about the security of a system should start with a number of claims about the system, followed by reasoning why these claims will always hold. For example, a claim would be that the smart contract only accepts measurements from the sensor that was initially linked with the contract and reasoning why the claim holds would include steps like signing the measurements on the sensor, as well as steps to make cloning a sensor impossible. It is important to note that all security measures in software are only good if the sensor hardware is also secure, and that protecting the hardware is complex and costly. Some solutions might not be entirely waterproof, but only make it really inconvenient to manipulate, which might be a solution, depending on the requirements, but if the system is marketed system as secure, the responsibility and risk lie also with the creators of the system.

It is still an open task to elicit the detailed security requirements of the system as imposed by the regulation agency and the customers. The customers buying the system might not pay for a system that is more secure than they strictly want. Currently, the responsibility for a shipment's compliance lies with the QA manager, that takes responsibility by signing a document after deciding on a shipment's compliance from the output of whatever logger system they used. Thus, all potential work on a more secure system should only be done after the exact security requirements have been elicited and is out of the scope of this work.

3.7 Unreliable Communications

One big problem in the production environment of the pilot projects with the system was that the Internet connection, including WiFi and mobile data, in some warehouses of pharmaceutical distributors was mostly nonexistent resulting in failed server connections. An Internet connection is, however, required to create the smart contract and upload the recorded measurements to it. The problem was partially solved by putting all uploads in a queue in the smartphone app so and regularly attempting to upload the parcels in the queue. Additionally, the queue is serialized into persistent storage so that uploads can be attempted regularly even across app restarts.

An additional problem was discovered later that when the sender of a shipment has a bad Internet connection, a situation can arise whern the receiver attempts to report temperatures for a parcel that the sender has not yet submitted to the server. Thus, the server does not know that parcel and no smart contract to receive the temperature measurements has been created. This was fixed by adapting the server so that it stores those temperatures reported before the respective smart contract exists in a separate table, so that problems of this type can then be manually fixed, i.e., manually creating a new smart contract and initiating the server to report the temperatures to it.

It is not possible to require from the pilot partners that they have good WiFi infrastructure in the warehouses because they are already doing us a service by participating in the pilot projects and taking additional time out of their busy days to interact with the system and help us improve it.

From the production system customers, however, a working WiFi infrastructure will be required because a good Internet connection is a basic requirement of the system and these requirements will be communicated before the system is sold to them. Naturally, the fixes countering bad internet connections in the warehouses will remain in the product and can still benefit production system customers.

Chapter 4

Implementation

4.1 Sensor

This section describes implementation details of the sensor software, including the storage of measurements in the sensor's flash memory, the interface exposed via Bluetooth and parameters affecting connection time.

4.1.1 Storage on the Sensor

The flash memory of the sensor board is organized in pages of 4096 bytes. It allows word-aligned writes of an arbitrary number of bytes (as long as they don't cross page boundaries), arbitrary reads and erasing only of entire pages. After erasing, all bits of the erased page in flash memory are set to 1. Writing allows 1 bits to be set to 0, but not 0 bits set to 1 (without erasing the entire page again). This limits the ways in which bytes can be rewritten on memory between page erases.

Storage of Measurements and Data

The measurements are stored in one flash page that is unused on the sensor by the operating system. The page size is 4096 bytes, thus theoretically allows to store 4096 measurements at 1 byte per measurement. Since the flash memory only allows word-aligned writes with a word size of 4 bytes, 4 measurements are cached before they are written to the flash memory at the same time.

Because the early prototypes of the sensor software made the sensors crash or restart regularly additional information is written to the flash memory. The parameters written to flash memory are explained in Table 4.1. These parameters written to flash allow resuming the recording after the sensor randomly reboots, but still detect the reboot and resulting possible gap in measurements when reading out the sensor.

Attribute name	**Size**
Start counter	1 byte
Interval	1 byte
Enable	1 byte
Start time	8 bytes
Shipment ID	20 bytes

Table 4.1: Restart Information

The **enable** flag is set nonzero when the sensor is recording and allows resuming the recording if the sensor reboots. The **interval** flag is set to the measurement interval whenever it is set via Bluetooth or to the default value upon booting a non-recording sensor and read back and reused as the measurement interval after rebooting a recording sensor. The **start counter** is incremented whenever the sensor resumes recording from a nonzero enable flag and is reset to 0 when the recording is enabled via Bluetooth on the sensor. Because of the properties of the flash memory described at the beginning of Section 4.1.1, the count is encoded such that the number of 0 bits indicates the count value. Thus, the value starts at 0xff and is decremented to 0xfe, 0xfc, 0xf8, 0xf0 and so on. Thus, the start counter byte can count up to 8 restarts. The **shipment ID** is set to the shipment ID whenever the characteristic is set via Bluetooth and read back and reused when rebooting a recording sensor. This includes resetting the shipment ID from flash in the sensor's advertise packet. The **start time** is set to the start time provided by the phone when starting the sensor and read back and reused after rebooting a recording sensor.

Calibration coefficients are also stored in flash, taking up 16 bytes, encoded as two double precision floating point numbers. The erase method in the sensor software is written so that the coefficients are cached in RAM before the flash page is erased and written back to flash afterwards.

Table 4.1 demonstrates that the debug information takes up 31 bytes in flash memory, thus, with 47 bytes used by both calibration coefficients and debug information, considering only word-aligned writes are possible, 4096 - 48 = 4048 measurements can be stored in flash memory while 1 byte is unused.

The exception handlers in the sensor system software were extended to write the program counter and link register contents to the end of the measurement flash page, as well as switch the red LED to glow steadily. If the measurements on flash already extend into the last 8 bytes of flash memory this causes the program counter and link register to overwrite the measurements, resulting in neither of them being retrievable due to the nature of how the flash memory handles overwrites. In all other cases it can, however, be very beneficial in finding the cause of errors that crash the sensor. Also, the red LED enables easy recognition of a sensor crash, whereas otherwise a non-responding sensor may or may not have crashed.

4.1.2 Sensor Interface

The sensor interface is exposed via the Bluetooth Generic Attribute Profile (GATT). This profile allows reading and writing of arbitrary characteristics and for the sensor software to execute arbitrary operations upon reading and writing these characteristics.

The maximum size of data that can be transmitted at once is limited by the maximum transmission unit (MTU), which is limited beyond the Bluetooth specification's limit by the sensor board's operating system implementation and indirectly by the sensor board's limited random access memory (RAM). To use a larger than default MTU the client, which is the phone in this project, has to request it and the peripheral has to acknowledge it. Because the size of all measurements on the sensor is larger than the MTU, the measurements have to be read out in multiple reads. However, throughput is maximized in this project by requesting the largest possible MTU on the phone, and writing the sensor software in a way that allows the MTU size limit imposed by the sensor to be as large as possible.

Characteristic	**Size**
Data	variable, as large as possible, dictated by the MTU, max. 103 bytes with current implementation
Enable	1 byte
Interval	1 byte
Measurement count	4 bytes
Readout position	4 bytes
Shipment ID	20 bytes
Start time	8 bytes
Start count	1 byte
Battery level	2 bytes
Calibration coefficients	16 bytes
Sensor count	1 byte
Single measurement	1 byte

Table 4.2: Sensor Characteristics

Table 4.2 lists all the sensor's Bluetooth characteristics. The **enable** characteristic is used to enable and disable the recording on the sensor. The **interval** characteristic is used to set the measurement interval in minutes, thus allowing an interval to be set between 1 minute and 255 minutes (4.25 hours). The **shipment ID** characteristic allows storing and retrieving the currently monitored parcel's shipment ID. In addition, the shipment ID is added to the sensor's advertise packet to enable the phone app to connect to sensors still enclosed in a parcel. This characteristic is designed to be able to store the contract address

of the contract linked with the shipment once the light client protocol implementation is ready. The **start time** characteristic allows storing and retrieving the timestamp of when the sensor was started. This value is decided on the smartphone app because the sensor doesn't have a clock with reference to the outside world's time. The **start count** characteristic returns the start count from the debug information stored on the sensor, thus allowing the phone to find out if a sensor has restarted during recording. The **calibration coefficients** characteristic allows storing both calibration coefficients on the sensor at the same time encoded as an array of doubles, each taking up 8 bytes. The **sensor count** characteristic allows storing and retrieving of a count of sensors on the same pallet to improve reading out multiple sensors associated with the same shipment. The **single measurement** characteristic allows the immediate readout of the temperature currently measured by the sensor which is used in the calibration process. The **measurement count** characteristic allows the smartphone app to learn the number of measurements on the sensor in the beginning of the readout process. The **readout position** characteristic is used to set the offset into the measurements that are returned when the data characteristic is read. This allows reading out of all measurements despite the MTU imposed limit on the maximum number of measurements that can be read out simultaneously. The phones knows which readout positions to request after it learns the measurement count on the sensor by reading out the measurement count characteristic. The **data characteristic** is used to read out the measurements from the sensor. The number of measurements returned is limited by the MTU. If the readout position is larger than the measurement count minus the MTU, only the remaining measurements are returned and thus the response packet smaller than the MTU. The **battery level** characteristic adheres to the Bluetooth specification concerning its identifier and encoding of returned value. The first byte of the value returned encodes the battery level on the sensor as a percentage value between 0 and 100.

Given the characteristics described above the readout process of measurements from the sensor works as follows:

1. The phone connects to the sensor.
2. The phone requests the largest possible MTU from the sensor.
3. The phone disables the recording on the sensor.
4. The phone retrieves the number of measurements from the sensor.
5. The phone sets the readout position to 0.
6. The phone reads a batch of measurements.
7. If the total number of received measurements is smaller than the measurement count retrieved earlier the phone sets the readout position to the index of the first measurement it hasn't yet received.
8. The phone repeats steps 6 and 7 until all measurements have been read out.

The Bluetooth advertise packet regularly broadcast by the sensor has a section that can contain arbitrary information. This section is used in this project to advertise the shipment ID, sensor count and sensor software version. The shipment ID in the advertise packet is used by the phone to connect to sensors related to a specific shipment without the user having to unpack the parcel and scanning the sensor ID. The sensor count information is used to improve reading out multiple sensors associated with the same shipment. The sensor software version is used by the phone to detect incompatible API changes of the sensor API, as well as the developers to quickly find out a given sensor's capabilities.

When one of the start time, shipment ID or calibration coefficient characteristics is written, the sensor erases the flash page containing the measurements. This is a security measure, disallowing changing the start time, shipment ID or calibration coefficients after the measurements have already been recorded and also allows the recipient to manually erase the measurements after reading them out in order to protect the confidentiality of the measurements when the sensor is shipped to some other sender of parcels.

4.1.3 Connection Time

During the first and second pilot project the software system produced long wait times until a sensor connected. This is due to a combination of the advertise interval set in the sensor software and the smartphone app implementation. The advertise interval is the interval at wich the sensor sends out the advertise packet. Information from this packet is required by the client to continue the connection process. In a crowded radio frequency (RF) environment, many of those packets might be missed or corrupted before a correct one can be received. During the first and second pilot project the advertise interval was set to two seconds. The average connection time was above 20 seconds. After the finding that the advertise interval doesn't strongly correlate to battery life the advertise interval was lowered to 100ms. The smartphone app was implemented during the first two pilots to start scanning for a particular sensor once its ID or shipment ID were known. After these pilots the smartphone app was changed so that background scanning for all sensors would commence as soon as the sending or receiving process of a parcel was started, thus allowing the phone to discover sensors even while the user is still scanning barcodes. Then in most cases when the sensor ID or shipment ID is known the phone already has all necessary information to start connecting to that sensor immediately.

A quantitative evaluation of the advertise interval and other parameters and resulting connection times and battery life is conducted in Section 5.3.

4.2 Server

The server component of this system currently runs a full Ethereum node and is responsible for all communication with the Ethereum network before the Ethereum light client protocol is ready to run on the smartphone directly.

Functions of the current server implementation include:

Creating a new parcel When a new shipment is started the server is notified of the shipment attributes, which stores the attributes in a Postgres database and creates a new smart contract that is linked to the shipment.

Upload temperatures for a completed shipment When a shipment is received the temperatures are read out from the sensor and uploaded to the server, which stores them in the database and also uploads them to the smart contract. The smart contracts decision can be checked by the phone via the server that retrieves the smart contract status.

Authentication & Authorization With the current implementation the smart contract is implemented so that the Ethereum account creating the smart contact must be the same as the account that uploads measurements. The server holds this single account and enforces authorization itself. For authorization currently there is a user management by the server and a login is required for all API calls. Authorization to upload measurements requires a valid server login, as well as knowledge of the shipment ID of the parcel that the measurements are to be uploaded for. User accounts can only be created by us directly. The token returned by the server after login is valid for 3 months.

Retrieving the temperature categories that can be selected for a parcel The temperature category and additional information formatted as free text are currently the only possible shipment attribute values that can be set in the app. In the future more attributes can be set and their possible values are retrieved from the server.

Handle push notification on Android devices The server keeps track of the devices' notification keys and sends push notifications via Google Firebase to the sender of a parcel once the parcel has been received.

Store reported temperatures for a parcel even if the parcel isn't known to the server This is to ensure that no measured temperatures go lost, even in case of bugs. These measurements can later be linked to parcels manually via SQL queries. This functionality was built in due to problems caused by bad Internet connections in some warehouses, as described in Section 3.7.

Store debug information The API includes a path that receives arbitrary strings that are printed to the server log. This is used by the app to send debug information to the server. This includes:

- Exceptions that are caught in the smartphone application and displayed to the user, but still warrant logging in the server logs.
- The version of any sensor connected to by the smartphone application.
- The battery level of any sensor connected to by the smartphone application.
- The sensor restart count if it is > 0 after connecting to a sensor.
- The app version is included in all requests sent to this API path.

Once the light client implementation is ready for the smartphone app it is planned to move the tasks of creating new parcel smart contracts and uploading temperatures to smart

contracts to the phone and potentially move the task of authorization into a distributed authorization framework, reducing centralization of the system drastically.

The server implementation is described in detail by Strasser [32].

4.3 Smartphone Application

For the smartphone platform Android was selected because an abundance of test devices was already in the possession of the persons involved in the project.

The purpose of the smartphone app is to read out the measurements from the sensor and upload them to the server from where they are in turn uploaded to the smart contract. The app was developed both by the author and Strasser [32]. The following features are supported:

Login This function allows the user to log into the server (Figure 4.1). The token returned by the server is used to authenticate most API calls. This token is also persisted across app restarts and if there is no Internet when the user starts the app, but a token has been persisted, the user is allowed access to the rest of the app.

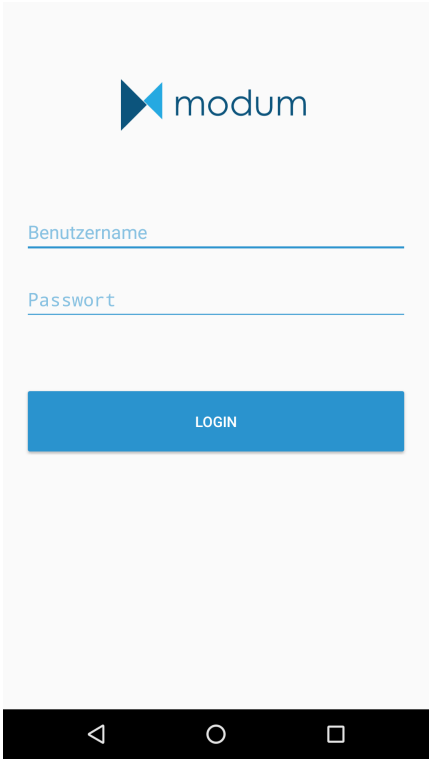

Figure 4.1: Login

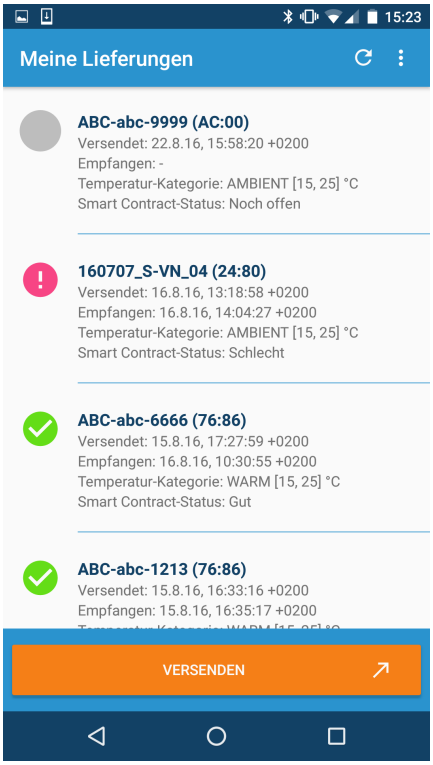

Figure 4.2: Starting a Shipment 1

Starting a shipment This includes setting all the necessary parameters for the shipment, connecting to a given sensor that the warehouse employee picks from a pile, starting the recording and uploading the shipment information to the server. The user clicks on the button send (Figure 4.2), scans the ID of the sensor to be placed in the parcel (Figure 4.3), scans the shipment's ID number (Figure 4.4), selects the desired temperature category for the shipment (Figure 4.5) and waits for the app to complete the process of connecting to the sensor, checking that it is indeed not currently recording, set the appropriate parameters and start the sensor. The order of scanning the two barcodes doesn't matter, although the user interface guides the user toward scanning the sensor ID number first.

Figure 4.3: Starting a Shipment 2

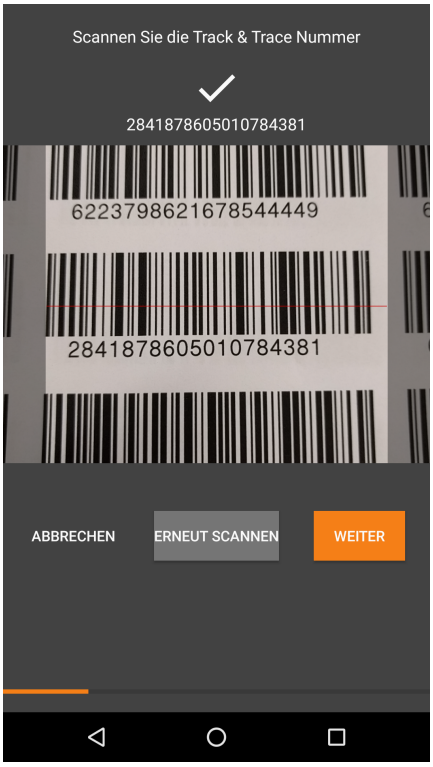

Figure 4.4: Starting a Shipment 3

Receiving a shipment This includes stopping the recording, reading out all measurements and other parameters from the sensor, displaying the measurements to the warehouse employee, uploading the measurements to the server to be uploaded to the smart contract, waiting for the smart contract's result on the shipment's compliance and displaying this result. The user clicks on the button receive (Figure 4.6), scans either the shipment's ID number or the sensor's ID barcode (analogous to the sending step) and waits for the app to complete the process of connecting to the sensor, checking that it is indeed recording, stopping the recording and reading out the measurements. The user can scan either the shipment's ID number or the sensor's ID barcode, but the user interface guides the user towards scanning the shipment ID, which is also faster at this point of the process because the user doesn't have to open the parcel to retrieve the sensor. If the connection to the sensor via the advertised shipment ID fails, the user is asked to scan the sensor's ID barcode, with which another connection attempt is made.

Listing the sent and received shipments and displaying their details and measurements when the user clicks on them This also allows the sender of a shipment to look at the measurements and the contract's decision once the shipment has been received. The main screen of the app is a list that contains all shipments that were sent or received by the logged in user. The items for this list are fetched/updated from the server when the user revisits this screen and when the user drags down on the list view or presses the refresh button. When the user clicks an item in this list he is taken to the detail view of the clicked shipment (Figure 4.7), which includes the date sent, date received, temperature category, additional information entered as free text by the sender and the smart contract's status. When the user clicks on the measurements button, he is taken to the graph view of the shipment's sensor's measurements (Figure 4.8). This button is grayed out if no measurements are available for the parcel, for example if the parcel hasn't been received yet. The measurements graph view includes measurements with the axes of time and temperature and the lower and upper threshold for regulation compliance as set by the temperature category. When the user clicks on a measurement it is selected and the timestamp and exact temperature of this selected measurement displayed. If no calibration coefficients are known for a sensor, the raw measurements are displayed with a warning "raw measurements (uncalibrated sensor)".

The upload of a new parcel or measurements of a received parcel is not part of the user processes above. Instead these uploads are performed in the background whenever the phone is connected to the Internet.

The app has a notification functionality that delivers push notifications to the sender of a parcel once it has been received. This functionality was mainly built by Strasser and

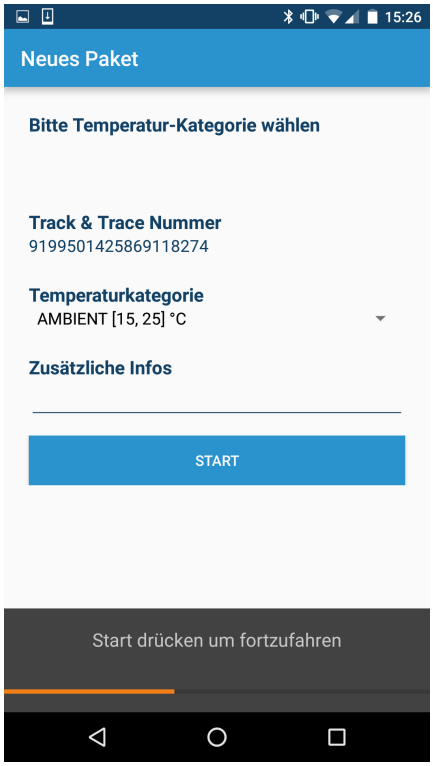

Figure 4.5: Starting a Shipment 4

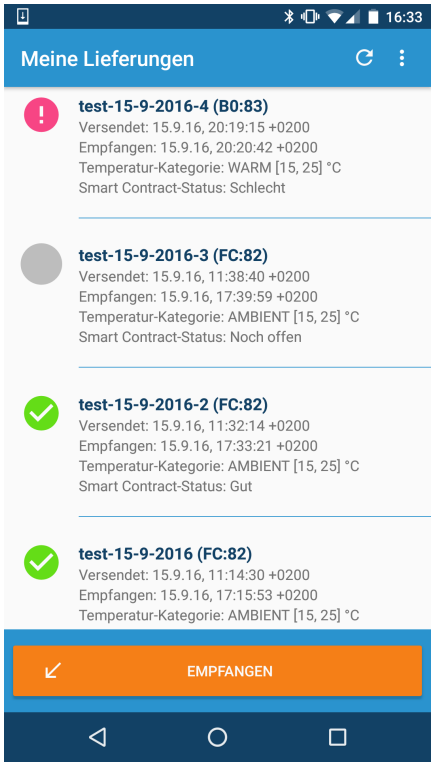

Figure 4.6: Receiving a Shipment 1

described in more detail in [32].

4.4 Smart Contract

The smart contract stores in its fields (Figure 4.9) the address of the creator (the owner), the address that is authorized to report temperatures and a storage location. The storage location can be a file path or IPFS location string with the idea that if the measurements are stored publicly, anyone can verify the measurements' hash by looking them up from the storage location and calculating and comparing the hash. The minimum and maximum temperature are stored when the contract is created and cannot be changed later. The hash of the measurements is stored, allowing anyone with a set of measurements stored outside the smart contract to verify if they match those that the smart contract's decision was based on. Additionally, the smart contract stores timestamps and temperature of the first few measurements that failed, the total number of measurements reported, the number of measurements that were out of range and the timestamp of the first and last temperature reported.

When reporting temperatures the smart contract keeps all fields that will be updated in local variables and writes these values once at the end of the reporting process (Figure 4.10). This is because changing a field, i.e. state variable of the smart contract, is more expensive than changing a local variable that isn't part of the smart contract's state between method calls.

Figure 4.7: Detail View 1

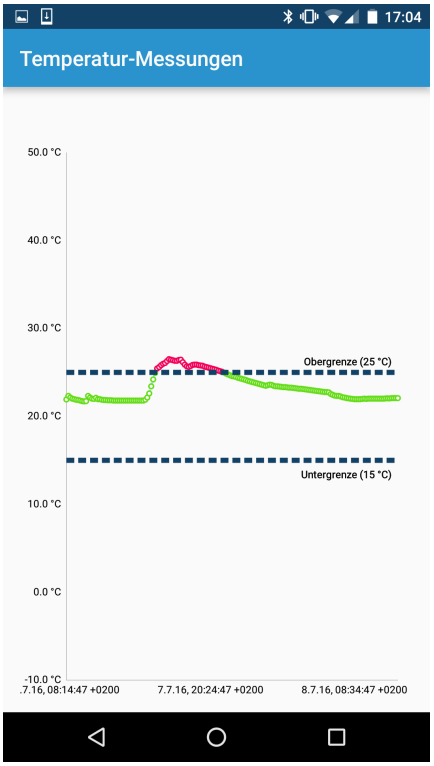

Figure 4.8: Detail View 2

```
address owner;
address temperatureWriter;
string storageLocation;

int8 minTemperature;
int8 maxTemperature;

uint32[] failedTimestampSeconds;
int8[] failedTemperatures;
uint16 maxFailureReports;
uint32 measurements = 0;
uint32 failures = 0;
uint32 firstTimestamp = 0;
uint32 lastTimestamp = 0;
bytes32[] hashes;
```

Figure 4.9: Fields of the Smart Contract

All properties of the smart contract can be retrieved via constant getters, which are calls that an Ethereum node will run locally on the stored state, instead of submitting it to the network and incurring code execution charges. Constant functions cannot alter the state of a contract, but might well include calculations based on multiple state variables in a contract.

4.5 Calibration

The sensor boards that were selected for the project come uncalibrated. Thus, the measured temperature returned by the sensor may deviate considerably from the actual temperature of the sensor's environment. The goal of the calibration is to find coefficients for a function that when applied to the raw measured temperatures produces temperatures that are close to the actual temperature in the sensor's environment. This property should hold over the entire intended temperature range that the sensor will be used in. There are two types of calibration: finding individual coefficients for every sensor or finding coefficients for all sensors of equal type from sample measurements of only a subset of all individual sensor devices.

To find the calibration coefficients, samples are recorded from the sensor at known reference temperatures and the coefficients calculated so that the error between the corrected measured temperatures and the reference temperatures is minimized.

An ideal calibration process would calibrate every sensor individually, consider multiple samples per sensor per reference temperature (to account for random error in sensor output between measurements of the same reference temperature), use 5 or more reference temperatures and use a nonlinear function that is closely matched to the sensor's measurement behavior.

```
function reportTemperature(int8[] _temperatures, uint32[] _timestamps) public {
  // ...

  /* read state variable, writing directly to it is expensive */
  var _measurements = measurements;
  uint32 _failures = failures;
  var _firstTimestamp = firstTimestamp;
  var _lastTimestamp = lastTimestamp;
  int8 _maxTemperature = maxTemperature;
  int8 _minTemperature = minTemperature;
  uint16 _maxFailureReports = maxFailureReports;
  var _currentFailureReports = failedTimestampSeconds.length;

  // ...

  for (uint32 i = 0; i < _temperatures.length; i++) {
    _measurements++;

    if(_temperatures[i] > _maxTemperature
    || _temperatures[i] < _minTemperature) {
      _failures++;
      // ...
    }
  }
  /* write state back, this is the expensive work */
  if(measurements != _measurements) {
    measurements = _measurements;
  }
  if(failures != _failures) {
    failures = _failures;
  }
  if(firstTimestamp != _firstTimestamp) {
    firstTimestamp = _firstTimestamp;
  }
  if(lastTimestamp != _lastTimestamp) {
    lastTimestamp = _lastTimestamp;
  }
}
```

Figure 4.10: Report Temperatures Method of the Smart Contract (shortened to retain only exemplary parts)

Further considerations when calibrating sensors are the rate at which the sensor sensor's die temperature assimilates to the environment temperature and how homogenous the temperature is in the sensor's immediate environment. The sensor's die temperature might assimilate only slowly to the environment temperature and time has to be given until the sensor's die temperature doesn't change over many minutes. Air isn't very suitable for an environment with homogenous temperature. Even within an isolated box, the temperature at the top might be slightly higher and clouds of unequal temperatures might waft around in the box. Often, temperature sensors are calibrated in a bath of liquid, such as water or oil because it is easier to generate a homogenous temperature in such a liquid and the transfer of heat from the liquid to the sensor is quicker than from air, resulting in faster and more accurate assimilation between the sensor's die temperature and the environment temperature.

Sensors are usually also calibrated in the final packaging that they will be shipped in, in case the packaging distorts the sensor's temperature measurements somehow.

Due to time and resource constraints, it was decided to start with a simple calibration process and improve it only when the need for improvement would become evident. The calibration process currently used finds individual coefficients for each sensor, but uses a linear function, resulting in 2 coefficients, and records temperatures at 3 reference temperatures. The environment used is a camping fridge that also has a heating function. The 3 reference temperatures are thus those generated by the camping fridge in cooling mode (8-9 °C), heating mode (55 °C), as well as room temperature (23 °C). An air environment was chosen over water or oil because the sensor boards aren't waterproof and concerns existed about cleaning the sensors after an oil bath and the radio frequency conductivity of oil, which is required because the calibration process requires a smartphone app to connect to the sensors while they are in the reference temperature environment.

In order to limit compiled program size of the sensor software the calculation of the calibration coefficients is executed in a smartphone app specifically designed for the sensor calibration process. The app works as follows:

1. The sensors to be calibrated are selected from a list of sensors currently advertising Bluetooth in proximity of the phone (Figure 4.11).

2. After the sensors have been put in an environment where the temperature is known and enough time given for the sensors to attain a die temperature equal to the environment, the environment temperature is read from the reference thermometer and entered into the phone app (Figure 4.12).

3. The app collects a temperature measurement sample from all sensors and stores them together with the environment temperature as reported by the reference thermometer (Figure 4.13).

4. After the required number of reference temperature measurements as defined by the calibration process (3 in case of the proposed system) have been collected, the app calculates coefficients for the correction function for each sensor and writes these coefficients to the respective sensors (Figure 4.14).

4.5.1 Quality Assurance

To verify the accuracy of the sensors, as well as assure the quality of the calibration process, the calibration app has a QA mode. In this mode reference measurements of

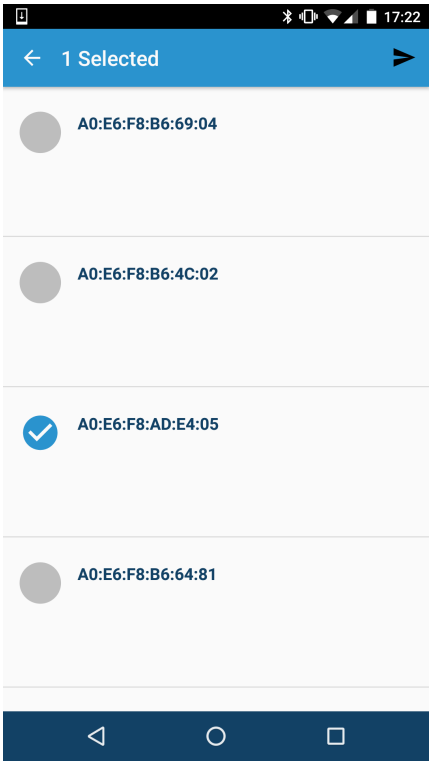

Figure 4.11: Calibration 1

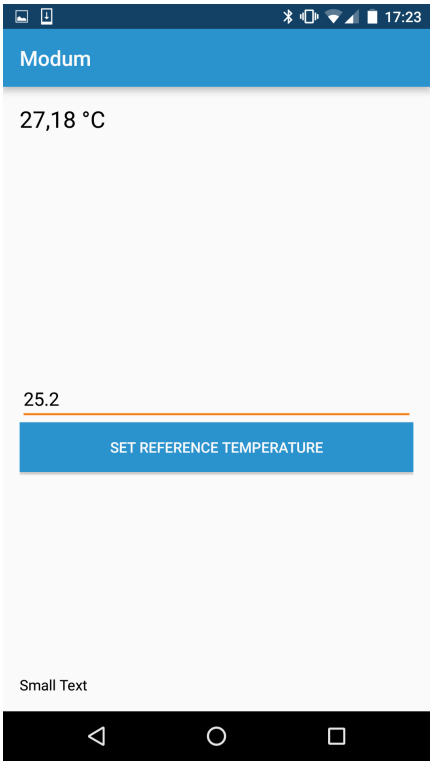

Figure 4.12: Calibration 2

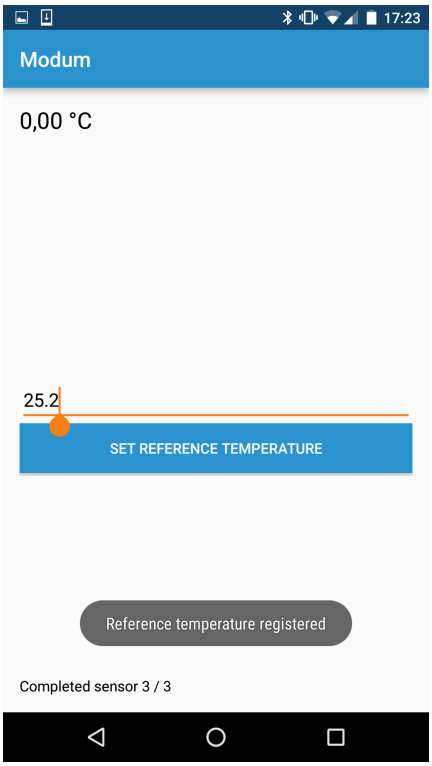

Figure 4.13: Calibration 3

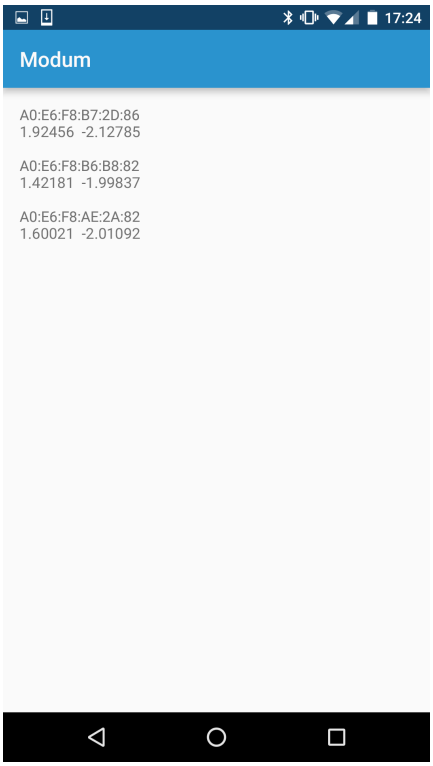

Figure 4.14: Calibration 4

multiple sensors are added at multiple temperatures, just like the calibration process (steps 1-3). In the last step, however, instead of calculating calibration coefficients, the QA mode calculates the mean squared error (MSE) for each sensor and displays it in a QA report together with the seperate measurements of the sensor and the reference temperature. An MSE close to 0 indicates good effectifeness of the calibration of the assessed sensors and allows to compare the calibration effectiveness between different sets of sensors. Thus, it also allows to compare calibration methods by comparing the MSE of sets of sensors calibrated using different processes.

Chapter 5

Evaluation

5.1 Pilot Projects

In the time until the finalization of this thesis the project was evaluated in two pilot projects. The pilot projects had the following goals:

- Evaluate the production environment usability of the system
- Learn specifics about the processes in the pharmaceutical supply chain from the examples of the pilot partners
- Get feedback from the pilot partners about optimization potential in their specific processes
- Learn specifics about the requirements for temperature trackers from the pharmaceutical companies
- Learn about the interpretation of the new GDP regulations by pharmaceutical companies and how they plan to meet them
- Learn about the volume of different market segments in the pharmaceutical logistics business and the potential to deploy the system in the different supply chain segments, as well as the potential revenue that can be created
- Improve the stability of the prototype system and find bugs
- Provide the pilot partners a satisfactory experience with the system and gain them as potential first buying customers of the finished product

The first pilot was with a small Swiss pharmaceutical company and one of the few large Swiss pharmaceutical wholesalers. Shipments were sent out from July 7 to August 11 once a week and received the day after. The shipments consisted of multiple pallets and about 3 sensors were packed on each of the pallets. Shipments were sent to two

different warehouses operated by the receiver company. Most pallets were sent with non-temperature controlled truck transports, but some were sent on air-conditioned trucks. In addition, some separate parcels were monitored that were sent using the postal service.

The second pilot was with two hospitals in a large Swiss city. Shipments were sent out from August 16 to September 20 once a week and received about two hours later in the other hospital. Two parcels were fitted with one sensor each and sent using cooled transport.

The findings of these pilot projects were immensely valuable in improving the prototype, as well as understanding the market for the solution better and coming closer to building a business case out of it.

Additional pilot projects are lined up, including international shipping, shipping in ship containers, shipments with a duration of multiple weeks and shipments into desert countries.

In the first pilot projects the emplyees were instructed to fill out a bug report form and send us an email with a screenshot attached whenever a bug occurred. This proved to be ineffective very soon because the users, being non-developers, had difficulties describing the relevant details about the bug and the circumstances that lead to it, as well as missing immediate feedback about the severity of the bug and reassurance that they didn't operate the app wrong. Thus, for part of the first pilot project and all of the second pilot project the users were asked and encouraged to call a modum employee any time a bug occurred. We also provided the pilot project participants with pre-stamped envelopes that they could use to send defective sensors or smartphones back to us.

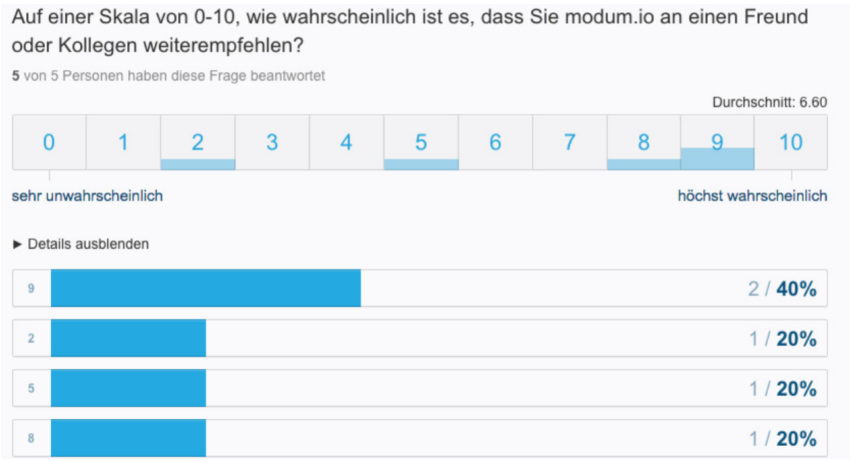

Figure 5.1: General Rating

A questionnaire (see Appendix C) was given to the participants of the first pilot project after completion of the pilot to assess their impression of the product, happiness, usability and possible improvements. 80% of people questioned think that the product can solve the new problem imposed by the new regulations while 20% were unsure. No negative answer was given. This suggests that the system is effective in its goal of solving the problem caused by the new regulations. Overall, the users of the system seem reasonably happy with the system (cf. Figure 5.1). They also gave decent ratings to the efficiency, usability

and our support (cf. Figures 5.2, 5.3, 5.4). These metrics suggest that the process doesn't have to be adapted much in terms of usability. The opinion on the reliability wasn't too great (cf. Figure 5.5), indicating that the reliability is an issue that needs to be improved. Many users desired a better overview of the shipments (cf. Figure 5.6) which is hard to provide in the smartphone application due to the limited screen size. The web application that is being developed will solve this issue. They showed some interest in ERP integration (cf. Figure 5.7), which is planned for development. The biggest problem as perceived by the users was the bad mobile Internet coverage in the warehouses, followed closely by the wait time to read out the sensors and the wait time to connect to the sensors (cf. Figure 5.8). This coincides with our impression of the biggest problems during the pilot.

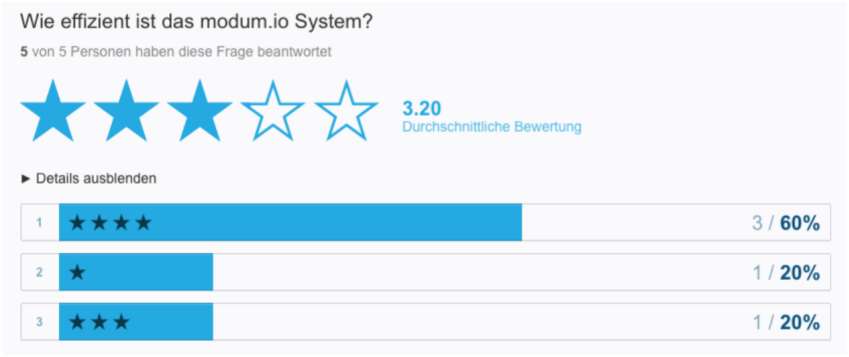

Figure 5.2: Efficiency

Figure 5.3: Usability

Figure 5.4: Support

As main measures for future improvement the pilot partners are promised increased reliability and stability of the system, better efficiency, a web app dashboard for better overview over the shipments than in the smartphone app and integration with existing ERP systems to minimize the amount of information the user has to enter directly into the app (via the smartphone keyboard and generally smartphone user interface (UI)). To increase efficiency promises are better integration in the pilot partners' processes, association of multiple sensors to one pallet via one shipment ID barcode, minimizing barcode

Figure 5.5: Reliability

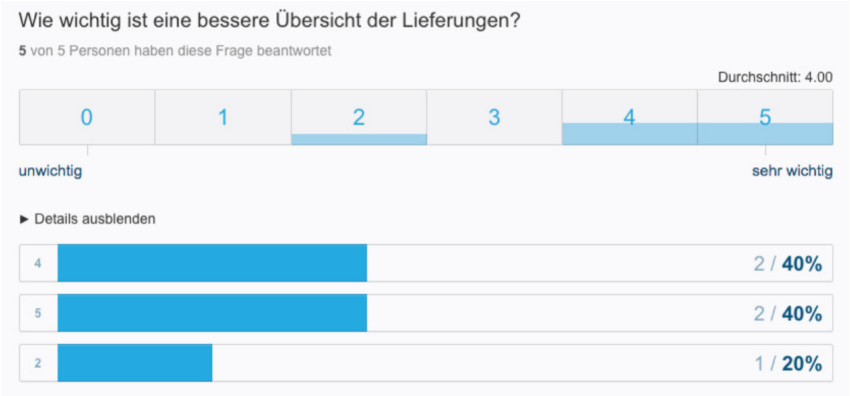

Figure 5.6: Better Overview

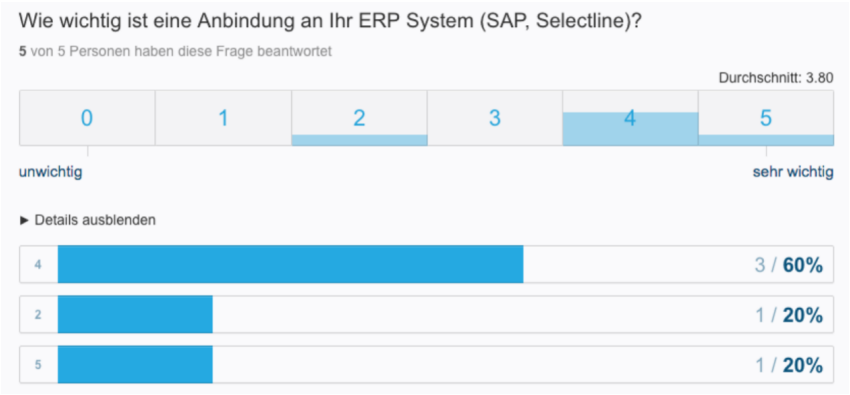

Figure 5.7: ERP Integration

scanning required by the users, and faster readout of the sensors. Concerning efficiency, a goal was defined of scanning the shipment ID barcode after unloading the pallet from the truck and performing the pallet level stock receipt. The sensors should then be read out in the approximately 30 seconds while moving the pallet to the area for the individual box level stock receipt. This is feasible when all optimizations, such as background scanning for Bluetooth devices, parallel connection to devices for readout and short advertisement intervals on the sensors, are applied.

Some pilot project participants also claimed interest in the system to monitor inbound products from different suppliers, monitor the temperature in the warehouse and monitor temperature on the last mile, that is the shipments to the pharmacies, doctors and hospitals that will prescribe and distribute the medicines to the patients. Monitoring all inbound, warehouse and outbound logistics with the system was mentioned as a potential goal because it reduces complexity and employee learning by having only one system to monitor temperatures, as well as saving time when reading out the warehouse sensors because the current system used isn't wireless.

In the first pilot 17 sensors were used, monitoring 55 shipments. Of the 55 shipments, temperature measurements could be retrieved on 52 shipments (95% success rate). 54% of shipments were within the specified temperature range. Most of the violations of the temperature range took place in the period before loading the shipment on the truck or after unloading it before moving it into the warehouse. This makes sense and points to an important problem of using temperature-controlled transports: although the temperatures during the transport may be within specifications, temperatures can easily exceed specifications before and after loading and if the shipment isn't temperature monitored noone will be notified about the deviation.

The second pilot project was plagued by many problems, including very bad internet connection at the sender and receiver, usability challenges for the users and shipments where the logistics employees forgot to add sensors. A full analysis and debrief with the participants is still pending.

The issues that occurred during the pilot projects can be classified into technical problems, including bugs, and usability problems, including bad UI design and processes that are confusing to the user. Confusing processes might either be changed to be less confusing

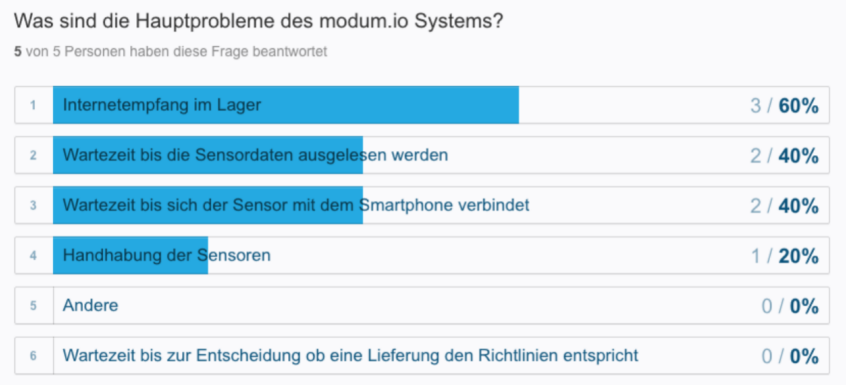

Figure 5.8: Main Problems

in the first place or the UI might be adapted so as to ideally guide the user through the confusing process. The following two subsections explain an example of each type of problem that occurred.

5.1.1 Usability Problems

In the beginning there was no distinction in the user interface between sending and receiving. Instead, the app would figure out if a parcel was being sent or received from the scanned barcodes and state of the connected sensor. This had pitfalls of its own, however, including difficulty in guiding the user through the process – the instruction step "Scan any barcode (sensor ID or shipment ID)" was just as confusing to the users – as well as lower efficiency due to scanning of barcodes that could be avoided if the user told the app whether he was sending or receiving a parcel. Since the user knows whether or not he is sending or receiving a parcel, it was decided to change the app to learn this from the user. One big usability mistake in UI design is putting two buttons that initiate an action next to each other. In the case of this project the buttons for sending a new shipment and processing a received shipment were put next to each other. In such situations it happens frequently that the user just clicks the button that is better reachable – e.g., the right one for right-handed people. This problem was exacerbated by the fact that the distinction between sending and receiving a parcel was confusing to the users. Both proceses involve scanning some barcodes and the phone interfacing with the sensors, and thus the importance of understanding the difference was lost on the users. An attempt to lessen the problem was made by putting only one button – either send or receive – at the bottom of the screen and have the user change the mode through a menu that is located in the upper right corner of the app. This reduced the instances of users selecting the wrong mode somewhat.

In general, the users seemed confused with the two barcodes – sensor ID and shipment ID. They use different barcode symbologies – Code 128 and QR code – and each scan dialog instructs the user to scan a specific barcode and also includes a little icon that shows the general look of the barcode the user is supposed to scan. Nevertheless, session recordings of user interaction often showed users not scanning the barcode than instructions demand.

Generally, the effect warnings and error messages on the users was underestimated. In one case a warning about measurements from an uncalibrated sensor caused the user to switch the app into sender mode and starting the just-received shipment with the same sensor- and shipment-ID again. Coulpled with Internet connection problems at the original sender's location, this caused the receiver to register the parcel on the server before the sender did. Thus, it is important to learn that error messages and even warnings can cause user errors and this should be anticipated when designing the user interface.

In the first shipment of the first pilot project the employees in the receiver warehouse had difficulties finding the parcels fitted with sensors out of all the parcels on the pallet. This was improved from the second shipment on in packing the sensors into parcels on the top layer of the pallet only and additionally marking these parcels with red stickers.

The time it takes for the smart contract to be mined and thus reach a decision about the compliance status of a shipment was confusing to the pilot users. In some cases when they received a parcel and the status was still undetermined afterwards they thought they made a mistake. Thus, the app was adapted to locally decide on a shipments compliance status. The smart contract's result overwrites the phone's after it has been mined.

5.1.2 Technical Problems

The changes described in Section 3.7 that cache and reattempt parcel and temperature uploads regularly weren't implemented at the beginning of the first pilot project. Thus, some temperature measurements were irretrievably lost when the upload failed at the receiver due to a missing Internet connection. After implementing the background upload for parcels, the situation where the receiver reported temperatures for a parcel that hadn't yet been created on the server arose multiple times. A database table was added to store temperatures reported to the server for which no parcel was known. Thus, when the parcel was created later by the sender moving to an area with better Internet connection, neither temperatures nor parcel would be lost and could still be linked.

During the first pilot multiple sensors were sent on each pallet. Sometimes only a subset of these sensors could be read out due to various problems, but the case that no sensor on a pallet could be read out never arose, thus there were always measurements for each pallet.

The smartphone app was deployed on the pilot phones via the Google Play store and set to automatically update the app when a new version was released. Sometimes this was still too slow, especially in conjunction with the unreliable Internet connections, and in once case the app failed to update in time after a non backward-compatible API change.

The decision to provide envelopes for returning sensors and phones experiencing problems proved valuable. Some sensors were sent back to us after not flashing either green or red after completion of the shipment at the receiver's warehouse. In once case this was due to a bug in the sensor software and in one case the sensor hardware was broken and the sensor had to be discarded. In one case deserialization problems of a cached shipment to be uploaded caused the upload to fail so that the pilot partners had to send us the smartphone for us to deserialize and upload the shipment manually. Thus, providing the envelopes was a good decision.

In the first and second pilot project the calibration coefficients were not yet stored on the sensor flash memory, but instead on the server. The smartphone app retrieved the coefficients using an API call and corrected the measurements after reading them out from the sensor. If the calibration coefficients for a sensor could not be fetched from the server, the measurements were marked as uncalibrated and a warning displayed when viewing the measurements. Since the Internet connection was bad in some of the warehouses failure to fetch the calibration coefficients occurred multiple times and the warning displayed for some shipments. For 70% of shipments in the first pilot project the calibration correction could be applied to the measurements, however.

5.2 Sensor

Reading out the sensor when it holds the maximum amount of measurements, given the current MTU maximum, takes about 5 seconds. This is acceptable, considering that multiple sensors could be read out in parallel.

After conclusion of the first two pilots, changing the advertise interval and sanning for sensors in the background in the smartphone app has made connecting the sensors quicker and will be employed in the upcoming pilot projects.

If a sensor unexpectedly restarts during recording the restart count is incremented on flash memory and uploaded to the server debug API path when the measurements are read out via the smartphone. The server logs of the pilot projects contain no message indicating a sensor count > 0 was ever encountered in the pilot projects. This strongly suggests that the sensor software is very stable and stability and reliability problems mainly stem from the smartphone app.

5.2.1 Calibration

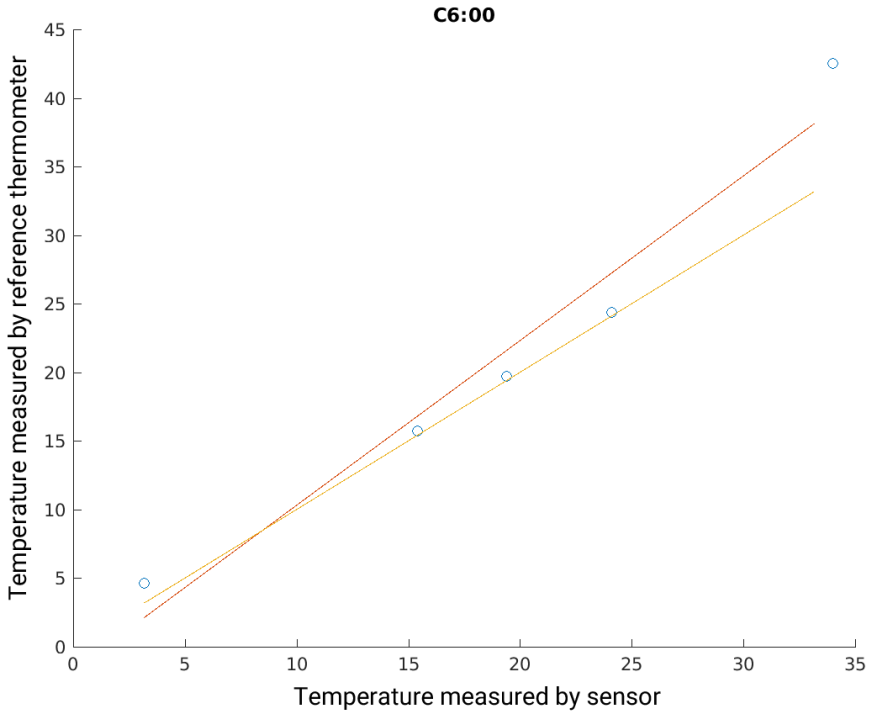

Figure 5.9: Quality Assurance Measurements

Independent from the quality assurance app, sample measurements were taken from 20 sensors at 5 different reference temperatures. Figure 5.9 depicts an exemplary set of measurements. The yellow line is the identity function and the red line is a linear function fitted to the data points in a least-squares regression model. From the graphic it is evident

that the measurement error of the sensor is non-linear, that an uncalibrated sensor can perform well between 5°C and 25°C and that linear regression is counterproductive in this range and for low temperatures. Thus, the calibration process and mathematical model need to be improved in future work.

5.3 Battery Life & Connection Time Tests

During the first two pilot projects the sensor's advertise interval was set to 2 seconds and its transmission power to -30dBm. This resulted in very long connection times to the sensor, but didn't require any battery to be replaced during the pilot projects.

For future pilots a systematic test of these parameters and their effect on battery life was conducted in order to improve connection times without sacrificing battery life.

Sensor ID (last 2 bytes)	**Advertise Interval**	**Transmission Power**	**Measurement 1 (s)**	**Measurement 2 (s)**	**Measurement 3 (s)**	**Measurement 4 (s)**	**Measurement 5 (s)**	**Measurement 6 (s)**	**Measurement 7 (s)**	**Measurement 7 (s)**	**Mean Value**	**Standard Deviation**	**95% Confidence Interval**
e0:81	2s	-30dBm	6	11	69	8	14	16	12	13	18.625	20.60	14.28
98:01	2s	-15dBm	11	10	7	10	82	22	19	6	20.875	25.32	17.55
19:80	2s	0dBm	8	29	7	17	12	7	9	83	21.5	25.93	17.97
2d:87	2s	10dBm	7	10	40	6	17	10	37	31	19.75	14.06	9.74
fc:82	1s	-30dBm	7	5	6	8	4	11	32	21	11.75	9.79	6.79
16:85	500ms	-30dBm	5	25	13	8	7	40	5	13	14.5	12.21	8.46
fd:03	100ms	-30dBm	2	1	2	2	1	4	5	6	2.875	1.89	1.31

Table 5.1: Connection Time Measurements

Table 5.1 shows the sensor connection time with different advertise interval and transmission power settings. The variance is very high, resulting in a very large 95% confidence interval, yielding the data only moderately useful. From the data it is evident that the connection time doesn't strongly correlate with the transmission power. The connection time does correlate with the advertise interval, but does only really attain acceptable values at a 100ms advertise interval.

Sensor ID (last 2 bytes)	Advertise Interval	Transmission Power	Battery Level Before	Battery Level After	Delta Battery Level
e0:81	2s	-30dBm	37	30	7
98:01	2s	-15dBm	34	empty	
19:80	2s	0dBm	70	66	4
2d:87	2s	10dBm	66	62	4
fc:82	1s	-30dBm	34	empty	
16:85	500ms	-30dBm	38	26	12
fd:03	100ms	-30dBm	46	empty	

Table 5.2: Battery Level Measurements 1

Sensor ID (last 2 bytes)	Advertise Interval	Transmission Power	Battery Level Before	Battery Level After 2 Days	Delta Battery Level	Battery Level After 4 Days	Delta Battery Level
98:01	2s	-30dBm	84	69	15	69	0
16:85	1s	-30dBm	86	66	20	66	0
fc:82	500ms	-30dBm	84	69	15	69	0
fd:03	100ms	-30dBm	77	66	11	66	0

Table 5.3: Battery Level Measurements 2

Tables 5.2 and 5.3 show the sensor battery level before and after three two-day test runs with different advertise interval and transmission power settings. From the data no correlation between the advertise interval or transmission power and the delta in battery level can be deduced. Especially the third test-run is interesting where the battery level didn't change at all. This reinforces the suspicion that the battery level doesn't decrease linearly, but remains around 60% longer than at different percentages.

Sensor ID (last 2 bytes)	**Advertise Interval**	**Transmission Power**	**Measurement Interval**	**Battery Level Before**	**Battery Level After 2 Days**	**Delta Battery Level**	**Battery Level After 4 Days**	**Delta Battery Level**
98:01	100ms	-30dBm	1min	69	65	4	61	4
02:04	100ms	-30dBm	2min	98	66	32	66	0
30:85	100ms	-30dBm	5min	38	14	24	empty	
e8:80	100ms	-30dBm	10min	100	65	35	65	0
2e:05	100ms	-30dBm	10min	93	66	27	66	0
24:80	100ms	-30dBm	10min	96	65	31	65	0
e0:81	100ms	-30dBm	10min	92	65	27	69	-4
2d:87	100ms	-30dBm	20min	30	empty			
16:85	100ms	-30dBm	30min	70	62	8	62	0

Table 5.4: Battery Level Measurements 3

Table 5.4 shows the sensor battery level before and after two two-day test runs with fixed advertise interval and transmission power settings, but different measurement intervals. This investigates whether one of powering up the sensor for measurement or writing measurements to flash memory dominates the sensor battery consumption. From the data no correlation between the measurement interval and the delta in battery level can be deduced. The second test run shows again the behavior of battery levels remaining around 60% for a longer period than at other levels.

As a consequence of these tests the advertise interval was set to 100ms while the transmission power was left at -30dBm.

Chapter 6

Summary and Conclusions

During the thesis a system was developed by Strasser and the author that allows for recording of temperatures during medical shipments and verification of the recorded measurements in a smart contract. The system includes a sensor platform running custom software, a smartphone application, a server application and the smart contract. A method to calibrate the sensors was developed and used for calibrating the sensors. The system was evaluated by analyzing the results of pilot projects in which the system was used for weekly shipments, as well as additional analyses of the server logs and connection time- and battery life measurements. The system could be much improved during the course of the pilot projects from the feedback gained and is now closer to production use readiness than at the start of the first pilot project. Many aspects of the system still need to be improved, as well as more to be learned about the market for the product, which is a goal of the currently ongoing second batch of pilot projects.

6.1 Pilot Conclusion

The system could be improved greatly from the feedback of the pilot projects, including the fixing of bugs, but also improving the user interface and finding and fixing usability drawbacks of the system. In addition, a lot could be learned about the market for the system and the specifics of the pilot partners' processes.

One of the companies participating in the first pilot project and a subsidiary of the other company agreed to participate in a future pilot project with an improved system, thereby demonstrating satisfaction with the prototype product and trust in the product's potential.

6.2 Hard Fork

When the Ethereum hard fork was announced it was decided to not update Ethereum right away in order to wait and see which branch of Ethereum would attract more miners.

After the Ethereum Classic branch of the fork seemed to retain only very few miners, the node was switched to the hard fork branch. One interesting finding was that not updating Ethereum at all after the fork made transactions valid on both branches. Thus, the transactions were mined by both Ethereum Classic and new Ethereum miners and the funds basically spent twice. Now, after switching to the hard fork branch the Ethereum Classic balance on the other branch could theoretically be sold off.

Bibliography

[1] Arzneimittel Bewilligungsverordnung, https://www.admin.ch/opc/de/classified-compilation/20011780/index.html, 1.9.2016.

[2] Guideline of 5 November 2013 on Good Distribution Practice of medicinal products for human use, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:C:2013:343:0001:0014:EN:PDF, 1.5.2016.

[3] Ethereum Whitepaper https://github.com/ethereum/wiki/wiki, 16.10.2016.

[4] Smart Property, https://en.bitcoin.it/wiki/Smart_Property, 16.10.2016.

[5] Bitcoin Escrow Service, https://en.bitcoin.it/wiki/Bitcoin_Escrow_Service, 16.10.2016.

[6] Contract, https://en.bitcoin.it/wiki/Contract, 16.10.2016.

[7] Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf, 16.10.2016.

[8] Serguei Popov: The Tangle, http://iotatoken.com/IOTA_Whitepaper.pdf, 16.10.2016.

[9] Josh Stark: Making Sense of Blockchain Smart Contracts, http://www.coindesk.com/making-sense-smart-contracts/, 16.10.2016.

[10] Antony Lewis: A gentle introduction to smart contracts, https://bitsonblocks.net/2016/02/01/a-gentle-introduction-to-smart-contracts/, 16.10.2016.

[11] Michael del Castillo: The DAO: Or How A Leaderless Ethereum Project Raised $50 Million, http://www.coindesk.com/the-dao-just-raised-50-million-but-what-is-it/, 16.10.2016.

[12] David Siegel: Understanding The DAO Attack, http://www.coindesk.com/understanding-dao-hack-journalists/, 16.10.2016.

[13] Ethreum Classic, https://ethereumclassic.github.io/, 16.10.2016.

[14] Alyssa Hertig: Ethereum's Two Ethereums Explained, http://www.coindesk.com/ethereum-classic-explained-blockchain/, 16.10.2016.

[15] https://www.reddit.com/r/ethereum/comments/4oithy/a_too_big_to_fail_political_hard_fork_is_very/, 16.10.2016.

[16] David Z. Morris: The Bizarre Fallout of Ethereum's Epic Fail, http://fortune.com/2016/09/04/ethereum-fall-out/, 16.10.2016.

[17] Script, https://en.bitcoin.it/wiki/Script, 16.10.2016.

[18] Block chain, https://en.bitcoin.it/wiki/Block_chain, 16.10.2016.

[19] Proof of work, https://en.bitcoin.it/wiki/Proof_of_work, 16.10.2016.

[20] Proof of Stake, https://en.bitcoin.it/wiki/Proof_of_Stake, 16.10.2016.

[21] Ethereum Project, https://www.ethereum.org/, 16.10.2016.

[22] Ether, https://www.ethereum.org/ether, 16.10.2016.

[23] https://github.com/ethereum/, 16.10.2016.

[24] Ethereum Homestead Documentation, http://www.ethdocs.org/en/latest/, 16.10.2016.

[25] Ethereum Foundation, https://ethereum.org/foundation, 16.10.2016.

[26] SensorTag, www.ti.com/sensortag, 20.7.2016.

[27] PunchThrough Bean, https://punchthrough.com/bean, 20.7.2016.

[28] Oberon HAP, https://oberonhap.com/, 20.7.2016.

[29] BTnode, http://www.btnode.ethz.ch/Main/Overview, 20.7.2016.

[30] Aaron Tilley: Apple's HomeKit Is Proving To Be Too Demanding For Bluetooth Smart Home Devices, http://www.forbes.com/sites/aarontilley/2015/07/21/whats-the-hold-up-for-apples-homekit/, 20.7.2016.

[31] Arduino, https://www.arduino.cc/, 20.7.2016.

[32] Tim Strasser: Managing Goods Distribution with Smart Contracts, 21.10.2016.

Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
DAO	Decentralized Autonomous Organization
DoS	Denial of Service
ERP	Enterprise Resource Planning (System)
GATT	(Bluetooth) Generic Attribute Profile
IoT	Internet of Things
LED	Light-Emitting Diode
MSE	Mean Squared Error
MTU	Maximum Transmission Unit
PoS	Proof-of-Stake
PoW	Proof-of-Work
QA	Quality Assurance
RAM	Random Access Memory
RF	Radio Frequency
UI	User Interface

List of Figures

List of Tables

Appendix A

Installation Guidelines

Git is required for all installation steps described here.

A.1 Smartphone Application

Retrieve the source code from the CD or from Github using:

```
git clone https://github.com/modum-io/android .
git checkout 3195ae1
```

Add a *google-services.json* file to the *app* directory. Such a file can be generated on the Google Developer Console.

Import the Android project into Android Studio and build and deploy it on any smartphone with Android version above 5.0.

A.2 Sensor Software

Texas Instruments Code Composer Studio and Bluetooth Low Energy software stack are required for these steps. The setup has been tested with Code Composer Studio version 6.1.2.00015 and BLE stack version 2.01.01.44627.

Import the SimpleLink projects *SensorTag* and *SensorTagStack* for *CC26xx* devices.

Copy the source code from the CD into the respective directories of the newly imported projects, replacing existing files.

Include the directories *PROFILES*, *Application*, *ICallBLE* and *Include2* in the project settings of the *SensorTag* project. Also define the `HEAPMGR_SIZE=5000` in the *SensorTag* project settings.

Include the directory *Include2* in the project settings of the *SensorTagStack* project.

Build and flash both projects to the SensorTag device using the Debugger DevPack device.

A.3 Server Software and Smart Contract

Docker and *Docker-Compose* are required for these installation steps. *Docker* commands need to be run as root.

Retrieve the source code from the CD or from Github using:

```
git clone https://github.com/modum-io/go_server .
git checkout 562221e
```

Generate a partial JSON file named *config.json* in the directory *goserver* with the following content

```
  "keyFilePassword": "",
  "pathToIPCEndpoint": "/datadir/geth.ipc",
  "dbHostUrl": "postgres",
  "dbPassword": "asdf",
  "dbUser": "postgres",
  "dbName": "modum",
  "dbSsl": false,
  "adminUser": "admin",
  "adminPassword": "####",
  "notificationKey": "####",
  "pathToJWTKey": "jwtkey.txt"
  }
```

The admin password marked with `####` can be chosen arbitrarily. The notification key marked with `####` should be a valid Google Firebase Notifications key lest the notification functionality will not work.

Build and run the server software, including Ethereum using:

```
docker-compose up
```

Appendix B

Contents of the CD

The CD contains this thesis and the source codes of the sensor software, Android application, server software and the smart contract, as well as the midterm presentation slides.

Appendix C

Pilot Project Evaluation Questionnaire

The questionnaire was presented as an online form on the website *typeform.com*. The questions are given below (in German):

1. Auf einer Skala von 1-10, wie wahrscheinlich ist es, dass sie modum.io an einen Freund oder Kollegen weiterempfehlen? [1-10]
2. Löst modum.io das Temperaturüberwachungsproblem? [Ja, Nein, Weiss nicht]
3. Wie zuverlässig ist das modum.io System? [1-5]
4. Wie effizient ist das modum.io System? [1-5]
5. Wie einfach ist die Handhabung der App? [1-5]
6. Wie gefällt Ihnen das Design der modum.io App? [1-5]
7. Was sind die Hauptprobleme des modum.io Systems? [gegebene Auswahlmöglichkeiten]
   - Wartezeit bis die Sensordaten ausgelesen werden?
   - Wartezeit bis der Sensor sich mit dem Smartphone verbindet.
   - Wartezeit bis zur Entscheidung, ob eine Lieferung den Richtlinien entspricht.
   - Internetempfang im Lager
   - Handhabung der Sensoren
   - Andere
8. Wie bewerten Sie den Support von modum.io? [1-5]
9. Stellen Sie sich vor, dass das modum.io System permanent und für alle Lieferungen in Ihrem Betrieb eingesetzt wird?
   (a) Was müsste man am modum.io System ändern? [freier Text]

(b) Wie wichtig ist eine bessere Übersicht der Lieferungen? [0-5]

(c) Wie wichtig ist eine Anbindung an Ihr ERP System (SAP, Selectline)? [0-5]

(d) Wie wichtig ist eine App fürs iPhone? [0-5]

10. Was hat Ihnen besonders am modum.io System gefallen? [freier Text]

11. Was hat Sie am modum.io System besonders gestört? [freier Text]

12. Bei welcher Unternehmung arbeiten Sie? [Auflistung der Pilot-Partnerfirmen]