University of Zurich^UZH

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

Improving the Blockchain System for Temperature Monitoring

Marc Heimgartner
Zurich, Switzerland
Student ID: 11-729-423

Supervisor: Dr. Thomas Bocek, Bruno Rodrigues
Date of Submission: November 16, 2017

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

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

Zusammenfassung

Transaktionen werden traditionell über eine Institution abgewickelt, welche sicherstellt, dass der Handel reibungslos und rechtmässig durchgeführt wird. Dabei entsteht ein gewisser Zeitaufwand sowie Kosten für die Abarbeitung der Institution. Die Blockchain Technologie bietet eine effiziente und sichere Möglichkeit die virtuelle Abwicklungen von Transaktionen und deren Integrität zu gewährleisten. Kombiniert mit "Intelligenten Verträgen" (smart contracts), durch welche Vertragsbedingungen in der Blockchain hinterlegt und Handlungen automatisch ausgelöst werden können, wenn die vorausgesetzten Bedingungen erfüllt wurden, ermöglicht die Blockchain eine voll automatische Abwicklung ohne dass eine intermediäre Institution manuell eingreifen muss. Das Potenzial solcher Anwendungen wurde unlängst erkannt, welche fundamentale Veränderungen mit sich bringen, nicht zuletzt weil die Rolle von Intermediären, wie z.B. Banken, Notariat, Markler, etc. nicht in dem System vorgesehen ist.

Das Start-up modum.io AG (modum) kombiniert IoT and Blockchain Technologien um die Datenintegrität von Operationen in der Zulieferkette sicherzustellen. Dabei konzentriert sich modum initial vor allem auf die Zulieferkette vom Pharma Sektor, da dort eine Änderung in den Richtlinien zur Distribution von medizinischen Produkten (GDP) zu Mehrkosten geführt hat. Grossisten müssen die Qualität und Integrität von medizinischen Produkten während des Transports sicherstellen und dies auch belegen können. Durch das Temperaturüberwachungssystem von modum kann diese Richtlinie kostengünstiger eingehalten werden.

Diese Arbeit verbessert und erweitert das Blockchain-basierte Temperaturüberwachungssystem von modum, durch Implementation am Back- und Frontend sowie theoretischen Analysen zu noch offenen Problemen.

Abstract

Transactions are traditionally handled through an institution that ensures that trading is smooth and lawful. This generates a certain expenditure of time and costs for the processing of the institution. Blockchain technology provides an efficient and secure way to ensure the virtual execution of transactions and their integrity. Combined with "smart contracts" that allow contract conditions to be stored in the blockchain and actions to be triggered automatically when the prerequisite conditions have been met, the blockchain enables fully automated processing without the need for an intermediate institution to intervene manually. The potential of such applications has recently been recognized, which bring along fundamental changes, not least because the role of intermediaries, such as *e.g.* banks, notary, marketer, etc. is not intended in the system.

The start-up modum.io AG (modum) combines IoT and Blockchain technologies to ensure the data integrity of operations in the supply chain. Initially, modum primarily focuses on the supply chain of the pharmaceutical sector, because a change in the guidelines for the distribution of medical products (GDP) has resulted in additional costs. Wholesalers have to ensure the quality and integrity of medical products during transport and be able to prove it. The modum temperature monitoring system allows compliance with the guideline and reduce costs.

This work enhances and extends modum's blockchain-based temperature monitoring system, through back-end and front-end implementation, and theoretical analysis of unresolved issues.

Acknowledgments

I would like to thank Prof. Dr. Burkhard Stiller and the Communication Systems Group (CSG) for providing me with the possibility to conduct my master thesis in an interesting and motivated start-up.

Special thanks go to my supervisors Thomas Bocek and Bruno Rodrigues, without their help and patient support the completion of this thesis would have certainly not been possible.

Finally I would like to thank Modum and all the members of the team, especially Malik El Bay and Sacha Uhlmann, for the exciting and fast paced work environment and their continuos support and encouragement.

Contents

Chapter 1

Introduction

Crypto currencies and especially bitcoin are nowadays widely known, not least because of the massive growth in the exchange rates [1]. The details of the underlying blockchain technology, however, is not known to the same extent yet. At core, blockchain technology is not technically revolutionizing. The two major components of which it consists 'distributed systems' and 'cryptography' have been known since the 1980s and agreement algorithms have been known in distributed systems long before the advance of crypto currencies [2]. What makes blockchain technology so intriguing is the potential impact and change it can have on a multitude of aspects of our daily life, like digital currencies, digital identity, improving supply chains, displaying ownership or show the provenance of products (*e.g.* show the original mine of a diamond necklace) [3]. This potential has been identified by many start-ups, as well as multinational corporations, and lead to boom of discovering applications which were previously not possible, and how already existing applications can be improved using blockchain technology.

Blockchain, described in simple terms, is a distributed decentralized system which records and validates transactions without the need of a third party of trust. In that sense the blockchain works similar to a traditional ledger but is distributed, therefore it is a subtype of the more general term 'distributed ledger'. Other data structures can be used for distributed ledgers, one such example is IOTA which constitutes of a directed acyclic graph. A blockchain can also be differentiated by its accessability. Generally public accessability, where everyone can access the data, and private, where only a defined set of people can access, is considered. Another often referenced term in conjunction with blockchain are smart contracts which allow to automate certain steps without the involvement or control of another entity [4]. Smart contracts are based on code, which includes the prerequisites of a contract and includes the steps which have to be executed. The next defined action can be executed if certain predefined conditions are met in the contract. This all can happen quickly without the need of manual or personal inspection or any intermediate entity like lawyer or notary. It is anticipated that this could revolutionize many to date manual tasks like for example the way how property, *e.g.* a house, is exchanged without the involvement of any third party and in a more efficient and faster way.

The advantages of the blockchain technology supplement the supply chain by providing data integrity while removing the need for a third party which ensures trust by audits. The complete traversal of products through the supply chain can be mapped publicly accessible and immutable resulting in a complete audit trail and documentation of the traversal. Additionally certain checks upon quality and integrity can be efficiently automated by smart contracts. This is desirable for almost all products and can lead to a new awareness and identification on the side of the customers when they can trace back their product to its origins and rely on the quality without trusting any authority.

1.1 Motivation

modum.io AG (modum) is a start-up which combines IoT devices and blockchain technology in order to address the recent tightened regulatory requirements imposed by the Good Distribution Practice of medicinal products for human use (GDP) [5]. To date, pharmaceutical companies comply with the regulation by the use of active cooled trucks which are excessive from both a economically and environmental perspective. The regulation under chapter 9 transportation mentions the documentation and maintaining the conditions necessary, especially the temperature, during the transport, however it also suggests utilizing a risk-based approach. Therefore the modum solution allows significant cost saving for products that do not require active cooling, incorporating non-temperature sensitive drugs while still complying with GDP [6]. Especially for the last mile transportation, where it is logistically and financially not possible to operate with active cooled trucks, the modum system could prove the integrity and quality of the shipment. Although mail order pharmacies are not yet governed by GDP, it is expected that the regulations for them will tighten too.

The previously built system of modum had been built as a fast moving forward prototype in order to validate the functionality of the system. The advantages and the potential reduction of costs have been shown in several pilot projects. As in the nature of developing a prototype in a new, fast evolving area a certain tradeoff between agile development and traditional best practices have to be made. With shifting from validating the prototype to a long term maintainable software project, the perspective of adhering to best practices for code quality and maintainability adjusts as well.

Therefore, the motivation is centered around improving code quality and functionality of the system and theoretical research on topics which could solve still open challenges to the system.

1.2 Description of Work

As outlined in the previous section, this thesis can be summarized as improving the blockchain based temperature monitoring system of modum. Because of the fast evolving environment around blockchain technology, the initial description of tasks has intentionally been kept broad to be able to react and adjust if necessary. It incorporates the following tasks:

1. A workshop with clients in order to validate previous assumptions and find not yet addressed requirements and concerns of customers.
2. An evaluation of the used backend technologies based on the workshop.
3. Implementation of improvements of the backend and frotend with respect to source code quality and features.
4. An evaluation of cross platform mobile clients.
5. A customer UI for the blockchain.
6. An evaluation of public / private blockchain.

1.3 Thesis Outline

This thesis is structured like the following: Chapter 2 introduces the modum shipment process, GDP, the guidelines for distribution of medicinal products, as well as competitors which build a similar system. The subsequent Chapter 3 presents the agenda and the results of the client workshop. Chapter 4 discusses the code improvements and additions to the previous system. The next Chapter 5 contains several theoretical evaluations of topics which could solve open challenges to the Modum system including cross platform mobile clients or public / private blockchain. The last Chapter 6 summarizes the results of the thesis, finalized by a conclusion.

Chapter 2

Background and Related Work

This chapter introduces the Good Distribution Practice (GDP) regulation, as well as the workshop which was a cornerstone to the back- and frotend improvements and showed itself as extremely valuable for better understanding customer processes and hence the arising requirements from it.

Additionally this section showcases a competitors of modum which are operating in the field of combining IoT with blockchain technology.

2.1 Good Distribution Practice (GDP)

The Good Distribution Practice of medicinal products for human use (GDP) are guidelines for the appropriate storage and distribution of medicinal products in the European Union [6] as well as in Switzerland [7]. The two main goals compose of preventing counterfeit medicinal products from entering the legal supply chain and ensuring the quality and integrity of pharmaceutical drugs by tightened requirements on the documentation during storage and transportation.

Any wholesale distributor has to comply with GDP or risks their license to be revoked. Furthermore it encourages any entity which participates in the distribution to adhere to GDP as stated in the guidelines themselves [6]: "Relevant sections of these Guidelines should also be adhered to by other actors involved in the distribution of medicinal products".

The GDP contains guidelines for ten topics with which a wholesale distributor has to comply [6]:

1. **Quality Management**. Wholesale distributors have to build and maintain a quality system where the responsibilities, procedures and risk management principles are defined according to their activities.

2. **Personnel**. It is crucial for a wholesale distributor to have sufficient competent personnel in order to warrant that all the necessary tasks are executed reliably and correct. Additionally the wholesale distributor has to employ a designated responsible person which is responsible for the compliance with GDP. This includes among other things that the quality management is established and maintained, certify that suppliers and customers are approved or keeping appropriate records.

3. **Premises and Equipment**. The premises and the used equipment has to be adequate to operating with medicinal products. This includes keeping them clean, dry and in a temperature range according to the product. Hardware which is used to supervise the condition of the storage room has to be tested frequently on accuracy and reliability. The calibration of such devices has to be based on national or international standards. Appropriate alarm systems need to be installed for preventing unauthorized access and to monitor deviations in storage conditions over a certain threshold. This threshold has be defined reasonably.

4. **Documentation**. Good documentation is an essential part of the quality system and should contain the original version as well as any later alteration. The documentation should not be handwritten and needs to be dated and signed by an authorized person. It should be stored for the period stated in the national legislation but at least five years. After that period personal data has to be deleted or anonymised.

5. **Operations**. The wholesale distributor has to ensure that the identity of the medicinal product is not lost. Especially should he use all options possible to prevent counterfeit medicinal products from entering the legal supply chain.

6. **Complaints, Returns, Suspected Falsified Products and Recalls**. All complaints, returns, suspected counterfeit products and recalls have to be carefully documented and handled. Responsible person and authorities should have access to those documents. An assessment of returned medicinal products is necessary before the decision for resale can be made.

7. **Outsourced Activities**. It is necessary to clearly define, agree and regularly control any activity which should be outsourced in order to avoid misunderstandings which could affect the quality or integrity of the medicinal product. The contract giver and contract acceptor have to agree upon a contract where the duties of both parties are defined.

8. **Self-Inspections**. Regular self-inspections should be executed to ensure compliance with GDP guidelines and disclose where corrective measures are necessary. Audits from extern experts can only support the self-inspections but not replace them.

9. **Transportation**. The wholesale distributor is responsible for the integrity and quality of the medicinal product during transportation and has to be able to prove that the medicinal products where not exposed to conditions which could compromise them. This includes the protection against breakage, alteration and theft. The necessary conditions for storage defined by the manufacturers have to be maintained during transportation and in case of deviation from those conditions or product damage both parties, the distributor and recipient, have to be informed. A process has

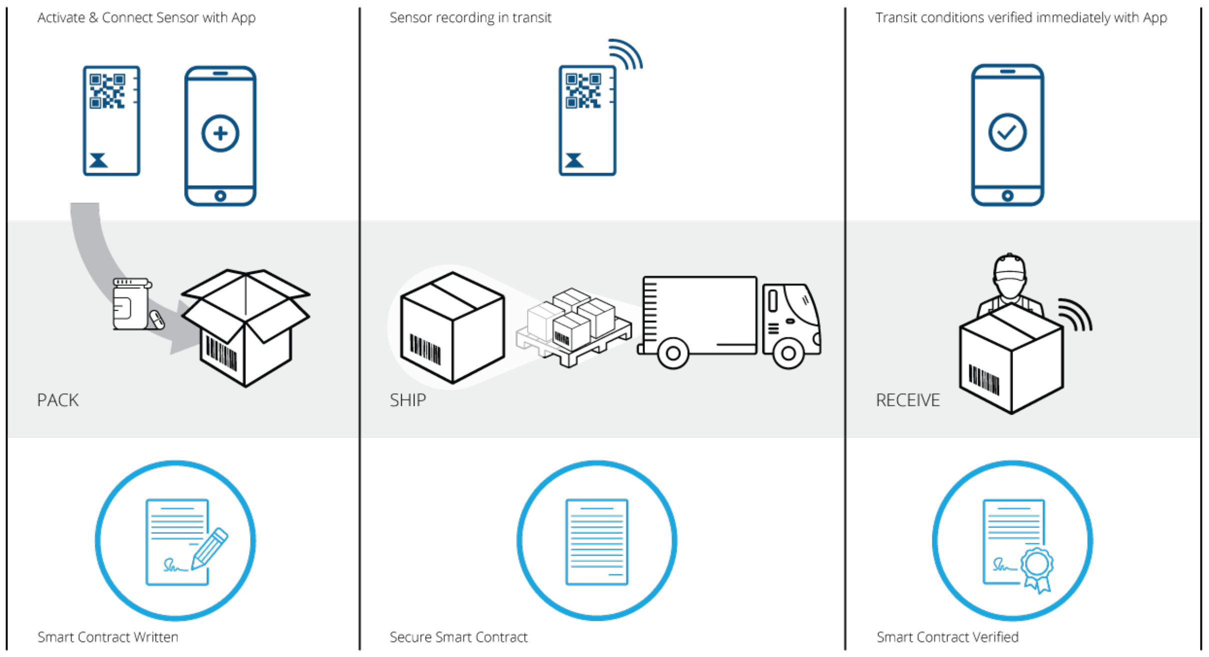

Activate & Connect Sensor with App
Sensor recording in transit
Transit conditions verified immediately with App
PACK
SHIP
RECEIVE
Smart Contract Written
Secure Smart Contract
Smart Contract Verified

Figure 2.1: Shipment process of the Modum system [5]

to defined for investigating and handling temperature excursions. When planning said transportations a risk-based approach should be utilized.

10. **Specific Provision for Brokers**. Those guidelines apply also to a broker with the difference that they do not procure, supply or store medicinal products and therefore those respective guidelines do not apply. Brokers have to register with permanent address and contact detail and in case of those details change they have to inform the authorities.

2.2 Blockchain-based Shipment Process

Users of the modum system can monitor the temperature of their shipments with the advantages of the blockchain, *i.e.* data integrity, reliability, public accessability and trust. Figure 2.1 visualizes the process of a shipment which is divided into 3 steps: (1) packaging the shipment with the sensor device, (2) transit of the shipment and (3) receiving of the shipment.

(1) To start the process the user utilizes the mobile client and scans the QR code on the sensor device which contains connection information of the sensor. In the next step the track-and-trace number of the shipment is scanned to combine the sensor with that shipment. Finally the user has the option to define metrics like measurement interval,

temperature range (alarm criteria) or a possible delay to recording start. All those information are transferred to the sensor device and the measurement recording is initialized. Upon packaging the shipment the operator has to include the product and the sensor device in the parcel. From the blockchain perspective a corresponding smart contract is created which checks the adherence to the metrics set with the android application.

(2) During transportation the sensor records, according to the metrics defined in the previous step, the temperature and stores it.

(3) Upon receiving the shipment at the recipient an operator has to read out the data from the sensor device. This is possible without opening the shipment itself by scanning the track-and-trace number with the mobile application which establishes a connection over bluetooth low energy (BLE). The data is read out, interpreted and transferred to the backend from where it is validated by the smart contract. The smart contract stores the hash of the measurements as well as yes or no outcome based on the requirements set in step 1. This proof-of-existence is publicly available and immutably stored.

2.2.1 System Architecture

The Figure 2.2 shows the architecture of the modum system separated into backend and frontend. The frontend clients (web client and mobile client) connect to the the backend via a JSON rest API developed in the Go programming language. The mobile client interacts with the sensor device over bluetooth low energy (BLE) and initiates the recording and reads out the temperature at the end. All the information from clients is sent to the modum backend server. The backend HTTP server acts as relay for the data and stores relevant data to the Postgres database and manages interaction with the Ethereum blockchain over an Ethereum node running on the backend server.

2.3 Competitors

modum is not the only organization in the IoT/Blockchchain space. Other companies operating in the same area include chronicled [8], an US based company focusing on IOT and blockchain technology to improve supply chains, providing digital identities and automation, and Ubirch GmbH [9], a German company providing an IoT and Blockchain platform.

Another more recent company to enter the market is Ambrosus, a swiss based startup "Combining high-tech sensors, blockchain protocol and smart contracts, we are building a universally verifiable, community-driven ecosystem to assure the quality, safety & origins of products." [10]. Ambrosus is very similar to modum at first glance, however there are some key differences between the two companies. modum has a strong focus on releasing a product for pharmaceutical shipments first, while Ambrosus is developing more general solution for the health sector, including food and medicine. While Ambrosus is also listing sensor systems as part of there ecosystem, the development of such seems to be at an earlier stage than the one from modum. Finally, modum has run multiple pilots,

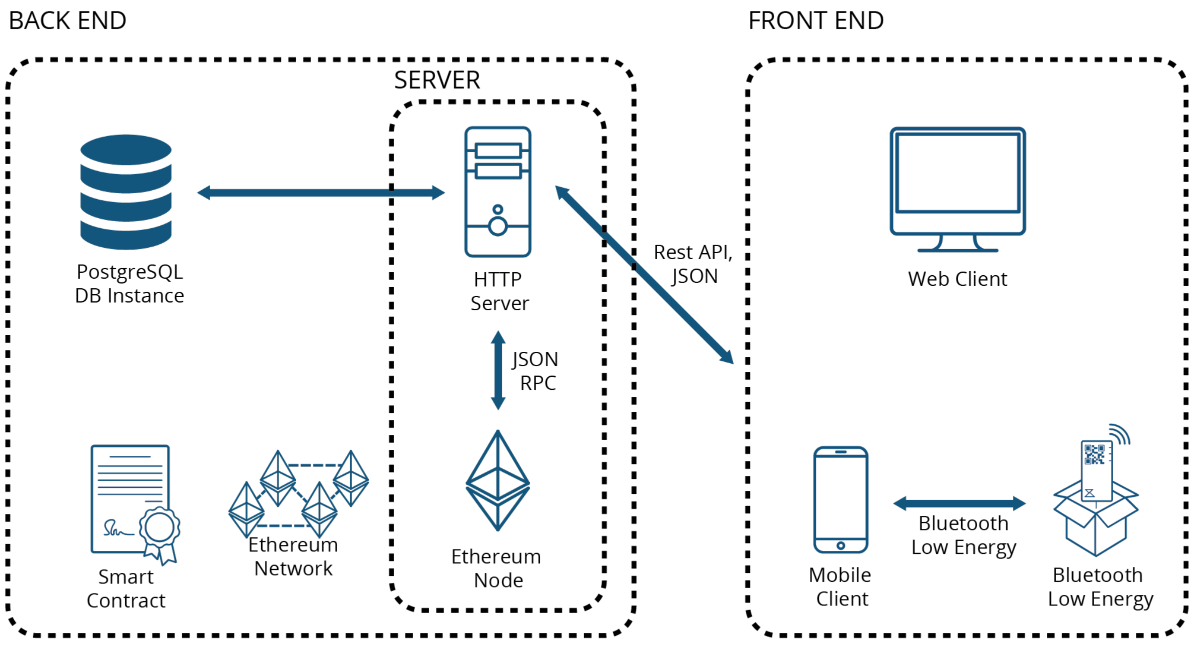


Figure 2.2: Architecture of the modum system [5]

within Switzerland and internationally, to understand the user requirements and test its business use case, while no information can be found of pilots for Ambrosus.

Operating in a rather young field, there is more than enough space for multiple organizations in the space. modum however stands out, as being focused on a specific first use case and clear focus, which can be built upon once successful in the market.

Chapter 3

Requirements

Recurrent verification if the development of software is aligning with the requirements of the users is vital for the success of a software project [11]. For the temperature monitoring system of modum an initial requirements engineering phase was conducted in order to build a prototype. Based on that prototype, a workshop was designed in order to revalidate if the implementation is meeting the requirements, present new envisioned features and discuss work processes and opinions.

3.1 Description of the Workshop

The preparations for the workshop started by inviting a group of responsible person for the compliance with GDP (see Section 2.1) which were identified as potential future users of the system. The knowledge from these users ranged from such which were previously involved in requirements engineering steps and already used the modum system to such which had no previous experience with it.

They were invited to the Technopark Zurich, where a meeting room was prepared with computers for a hands-on session. Apart from the participants, four members of the modum team were present to help and observe their interactions.

The workshop had a clearly structured agenda:

1. **Introduction**. An introduction round of all the participants ensured that they get accustomed to the people and helped them to better understand who was participating, which role they would take in the workshop and what their background was. Therefore a introduction generally contained their name, who they work for and what position they have in the company as well as their goal and expectations for the workshop.

   Important for that was to ease them into their position in the workshop by ensuring that they had nothing to prepare for, there are no wrong answer or questions and if they do not understand something they should ask. All of the information is valuable and it helps a lot to find out what is not as easy comprehensible as was thought.

2. **Presentation of modum**. Not everybody had the same knowledge about modum. Some participants were previously involved in pilot projects and therefore had a good understanding of the system, whereas others had no previous experience. The presentation of modum gave them an overview of what the system should accomplish, why it is built in the first place and how it might solve a problem they have themselves.

3. **Hands-On**. This phase deliberately contained throwing them into cold water by letting them work with the dashboard without much prior explanation of how certain things should work. To provide a structure for them they got a list of tasks on which they should work for example 'filtering all the shipments which are still in transit' or 'exporting a PDF report for a shipment'. Members of the modum team were observing their interaction and noted the problems of the participants and further remarks. At first, the participants were reluctant not to make mistakes or rush, most likely because they were observed but they got used to it quickly after they were again reminded that 'errors' can give a good indication about what is unclear to the general user of the system. The observers would only give hints when the users after some effort still not could proceed, however only minimal information was given and the user was encouraged to try it out and give feedback.

   Some of the participants were slower than others. In order to not rush or pressure them, the faster ones were engaged in conversation about how their experience was, where they see potential for improvement or what is missing. This allowed to let everybody go at their own pace and still ensured that others would not be bored or left alone without something to do. While not exactly planed for this had benefits to already pre-discuss features in one-on-one conversation and generate topics for the open discussion later on.

4. **Demonstration**. Some potential features had already been discussed in the development team, but were not yet implemented, either because the implementation would take too much time, or as it was not clear if it would provide additional value for customers. For example, comparing different shipments or very complex filtering mechanisms. Such features were presented with mockups and discussed if those features are interesting and how they would like them to be implemented. This feedback again was noted down by all the observers.

5. **Feedback and Discussion**. The final phase was constituted of a feedback and open discussion session. As mentioned previously, topics arose from the one-on-one discussions which then could be discussed in the whole group.

The feedback from the participants about how the workshop was organized and conducted was throughout positive. They all participated actively and many interesting discussion were held, resulting in a lot of implementation ideas. In total the workshop took about four hours.

3.2 Discussion of the Results

Suggestions from the discussions and observations during the whole workshop were noted down individually by each observer. This produced redundancy in the information over the four team members, but also ensured that little information could be lost. However, the information had to be consolidated from all sources in order to make it useable. In meetings, all the individual topics were categorized, discussed again to gather all the arguments and rated based on their importance for the next product state. The result of this process are accumulated in three lists. Table C.1 includes all the features suggested for the dashboard, Table C.2 lists usability issues identified in the workshop and Table C.3 a few bugs which were found by the participants during the workshop. All the tables can be found in the appendix. Subsequently a subset is discussed which were very surprising or critical to the success of the product.

An interesting observation was that the participants were used to Windows desktop applications and not that much to the web. This fact is the underling cause for several usability issues in Table C.2. As an example, some users always used double click which in most cases does not interfere with anything in the web, however some components in the dashboard can be expanded by clicking on them. A double click results in quickly expanding the component and collapsing it again. This flickering confused the users and they thought it was not working correctly. Another such example would be right clicking in the browser. They expected to get a menu with possible actions similar to desktop applications and not the browser menu. Finally, one user could not navigate back, apparently it was not intuitive for him to use the browser back button and suggested a back button in the web app itself. All those insights are extremely valuable since developers know how they have intended a feature to work and will most likely not try something out of the ordinary.

Another topic which was discovered to be of more importance than initially thought by the developers is role and permission management. Although clear that this has importance and is necessary, it was expected that there is a defined set of roles which are known, *e.g.* ‘Auditor‘, ‘User‘ or ‘Admin‘. However, the discussions showed that there is a need for defining roles with a set of permissions themselves.

The overall verdict of the participants was that the already implemented functionalities are useful and they are interested to work with it, but for using it on a daily operational level some of the features mention in Table C.1 rated as ‘Critical‘ had to be implemented first. Especially more meta information about the sensor devices, being able to approve any shipment with a comment and the ability to define roles themselves.

Chapter 4

Design and Implementation

This chapter introduces the implementation changes in the back- and frontend. The backend is implemented in the Go programming language and the frontend uses the modern JavaScript framework Vue.js for building a single-page application (SPA).

4.1 Design

The programming language Go was chosen for the initial prototype because the development of the library Go-Ethereum to interact with the Ethereum network was by far the most advanced. This is not necessary anymore since the connection to Go-Ethereum can be established over RPC calls and clients for that haven been written in multiple other programming languages. However the second reason is still valid since Go-Ethereum is a crucial dependency it makes sense that developers can fix problems in the library itself if anything would happen. It was decided for those reasons to keep the development in Go although not strictly necessary.

4.2 Go

Go (often called golang to solve the ambiguity) is a programming language which was developed at Google and was publicly release in 2009. The syntax and functionality was designed by Robert Griesemer, Rob Pike and Ken Thompson which also explains the resemblance Go has with more traditional compiled and statically typed programming languages like C. Go has some pretty interesting features and design decisions on a language level, which it enforces rigorously and it is up to the user to decide their opinion on them. Like in any other programming language there are good and bad parts of which subsequently a non exhaustive number of examples are given.

Go is not a object-oriented programming language. It takes a pretty clear position to favour object composition rather than inheritance. A composite data type with methods can be defined via 'struct' and other structs can include the functionality and state by composition, however unlike the ‘is-a relationship’ from inheritance, composition defines a ‘has-a relationship’.

Polymorphism is achieved in Go through the use of interfaces. They are implicitly implemented contrary to for example Java where this has be done explicitly. A variable of the interface type can hold any value which implements the interface. Special is the empty interface, which has no method defined and therefore is implicitly implemented by any data type. This can be used to accept any value.

There are several language tools included among others: ‘go build’, ‘go test’, ‘go fmt’, ‘go get’, ‘go generate’. While the use of build and test are quite obvious, build the executable and execute the test runner respectively, the others might not be. Go is quite opinionated on styling and formatting code which goes to such an extent that it does not compile if unused variables or imports are present. ‘Go fmt’ does automatically format your code according to the golang style guide which should help to have all Go code uniformly styled and therefore easier to understand.

The dependency management is based on remote repositories such as for example Github. Imports are written in the code according to the Github path, *e.g.* `import "github.com/gin-gonic/gin"` and can be downloaded and cached locally with ‘go get’. Go does dictate the workspace folder structure according to this naming scheme so that is able to resolve dependencies which are on the system. This might seem like a good way to do dependency management at first glace but comes with many problems the way it is implemented. The verdict from Google is that for major releases of packages a new repository should be opened, otherwise there should be no breaking changes to the master branch. However package developers rarely follow this guideline resulting in a dependency wild west where builds can break because of changes from the authors of those packages. An example which occurred during the development of the application showcases good the unexpected impact this can have: The developer of a well known logging library for Go changed his username from starting with a capital letter to a lower letter without considering the impact that might have. Effectively this changed the import path from `github.com/Sirupsen/logrus` to `github.com/sirupsen/logrus` (note the lowercase letter of the username). The result of this was, that in order to resolve this issue, it was necessary to change the import statement. Although this can be simply done for your own dependencies the problem starts with the dependencies of the dependencies. Since this library was so popular many libraries used it internally which meant that all had to change the name otherwise the binaries would not compile.

Go is most criticized for the lack of compile-time generics which leads to code duplication or the use of reflection, with which the advantage of the statically typed checks are lost. A simple example which illustrates this problem is the implementation of a data structure like for example a stack for integers. In Go there are different types for integers, *e.g.* uint8, int16, int32. There is no simple way to support all these data types when building a data structure. One from the language suggested way to solve this is code generation, where the same code would be generate for each type, effectively duplicating the code several times with the sole difference of supporting another intrinsically same working data type.

A positive aspect of Go is how well concurrency is built in the language with goroutines and channels. A goroutine is a function which can run concurrently to other functions. It is not a thread on system level but one managed by the Go runtime. Channels provide a way for goroutines to communicate with each other and synchronize their execution. In this model of sharing memory by communicating, data races can not occur by design.

Building for different platforms can be a cumbersome and time-consuming process. Go does provide capabilities to cross compile for other platforms. For this only some environment variables have to be changed and it works very reliably as long as it is pure Go code. However potential problems can arise from packages which use cgo, Go packages which call C code. In such cases it can be very difficult to install a C compiler which can be used for cross compiling.

4.3 Implementation

The initial goal for the implementation contained refactoring the prototype backend into a long term maintainable form, adhering to common best practices in software development and adjusting the frontend to the respective changes. In a second step additional features and issues which surfaced from the workshop (see Chapter 3) should be added or resolved on both sides. However some of the more crucial features, like for example a permission system, showed that deeper and more fundamental change is necessary to the backend. Therefore it was more efficient to start a rewrite completely from scratch and only loosely base the new version on the prototype.

4.3.1 Structural Changes

The importance of the structure of source code for maintainability and efficiency is often underestimated in software development. It is important to have a logical easy to follow layout which is consistent. The prototype showed a mixture of concerns where the separation was not thoroughly consistent [12]. Such inconsistencies can often be found in the evolution of a software project and they potentially increases the more developers are involved. To resolve the issue, which was understandable for the prototype, a structure in the form of MVC was closely followed. However since the frontend is a SPA there is no traditional view layer. Apart from separating *models*, which contain all the interactions with the postgres database, and *controllers*, which handle the incoming requests and render the respective JSON response, there are packages for *middleware*, which handle authentication and authorization, *config* and *migrations*. Migrations allow to define database changes alongside the code changes and efficiently keep them under version control.

4.3.2 Generalization

The verbosity of the code in the prototype was mainly introduced through aspects of the Go programming language itself, like for example the lack of generics as mentioned in Section 4.2. As a result much code was duplicated in models and controllers. Therefore a main goal of the backend refactoring contained removing code duplication within the boundaries of the Go programming language. A REST API [13] contains many parts which do exactly the same in essence, but only differ in the underlying composite data type. As an example a REST API generally contains controllers for *create*, *read*, *update* and *delete* (often referred as CRUD) and only differ on which model they execute this action.

A generic implementations of a controller for each action was defined by a common interface which models have to implement. In the routing definition the correct model for this endpoint is set in a closure. This allows to use the same general controller definitions where possible and only define separate ones for special cases.

Despite successful reduction of the duplication of the controllers, the models still had to implement the definition of the common interface defining the SQL queries necessary to do the actions (CRUD) individually. The most used object relation mapper (ORM) in Go was used in the prototype which was cumbersome to work with and often the query still had to be defined in raw SQL. Additionally a minor SQL attack injection vector was found during experimentation. All those issues showed the need for a reliable and generalizable way to interact with the database.

Although initially not planned the end result was the creation of a Go ORM which needs minimal definition and implements general CRUD operations. Listing 4.1 shows how an example of how a model is defined. On the model "company" now several functions can be called like, Create, List, ListWhere, Read, etc. This works internally by defining company to have a DBModel, which is an object which does implement the interface defined for the generic controllers. Go does object composition which allows company to directly use functions from DBModel. Additionally Go knows the concept of tags which allow to define information of a field. In this case the tag "db" is defined of the field which defines the name of the field in the database and additional properties like *PRIMARY_KEY*, *SORTABLE* or *SEARCHABLE*. The ORM does analyse the defined fields along with the tags behind the scene and generates queries accordingly. It supports additional necessary things like pagination, foreignkey resolution with efficient JOIN query building and a filtering and sorting system which works for nested fields as well. Additionally the generic definitions can be overwritten by defining them on the model. As an example the method List could be overwritten to use ListWhere and always only show companies updated in the last day. Even when overwritten it is still possible to take advantage of the original implementation by accessing it over the DBModel in the model making it a very versatile approach.

The only disadvantage is that the DBModel instance has to be initialized, which can be solved with the factory pattern, shown in Listing 4.1 in function NewCompany, because Go has no constructors.

Listing 4.1: Definition of a model based on the modum ORM

```
type Company struct {
  DBModel

  ID          int          `db:"id,PRIMARY_KEY"`
  CreatedAt   *time.Time   `db:"created_at,READ_ONLY,SORTABLE"`
  UpdatedAt *time.Time `db:"updated_at,READ_ONLY,SORTABLE"`
  Name      string     `db:"name,SEARCHABLE,SORTABLE"`
}

func NewCompany() *Company {
  tablename := "companies"
  c := new(Company)
  c.DBModel = NewDBModel().Bind(tablename, c)
  return c
}
```

4.3.3 Documentation

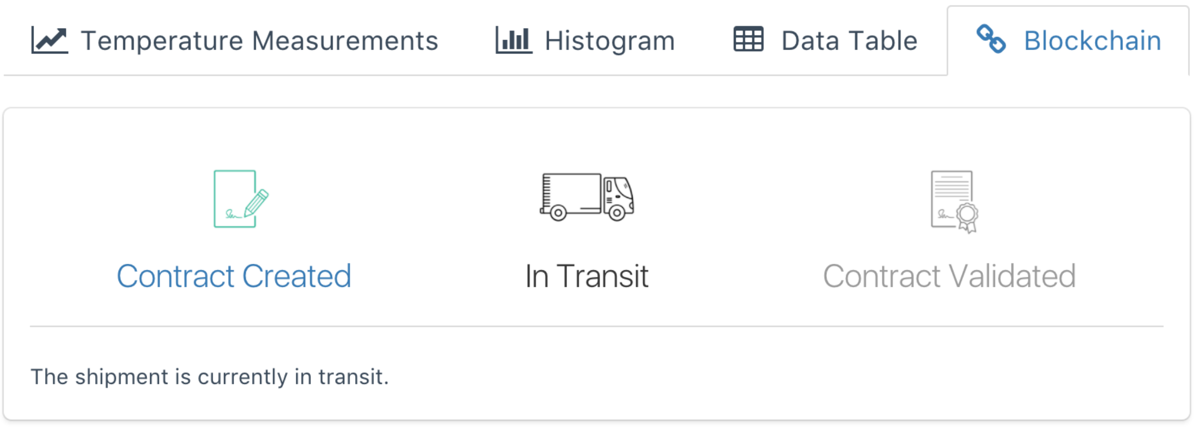
Documentation can drastically decrease the time necessary for a developers to understand the intention and functionality of source code [14]. Go does encourage writing documentation be providing functionalities to generate nice looking documentation files along with examples based on the comments in the source code. However, documentation of an application is not necessarily finished with providing source code documentation. Especially important is a clear documentation at system boundaries on how interactions are defined. In case of modum the interaction happens over a REST API which is used by the modum clients and will most likely be integrated by some business partners. Therefore a clear need for a well defined API documentation exists. One of the most known REST API documentation forms is swagger. The definition of swagger is written in JSON, but a special UI allows to represent it in a nice form which also can show examples and directly issue queries.

The likelihood of forgetting to adjust the documentation when changing code is higher when the source code and documentation are separated [14, 15]. Therefore a Go library was used which allows to define the documentation in source code comments and build it from there.

4.3.4 Frontend

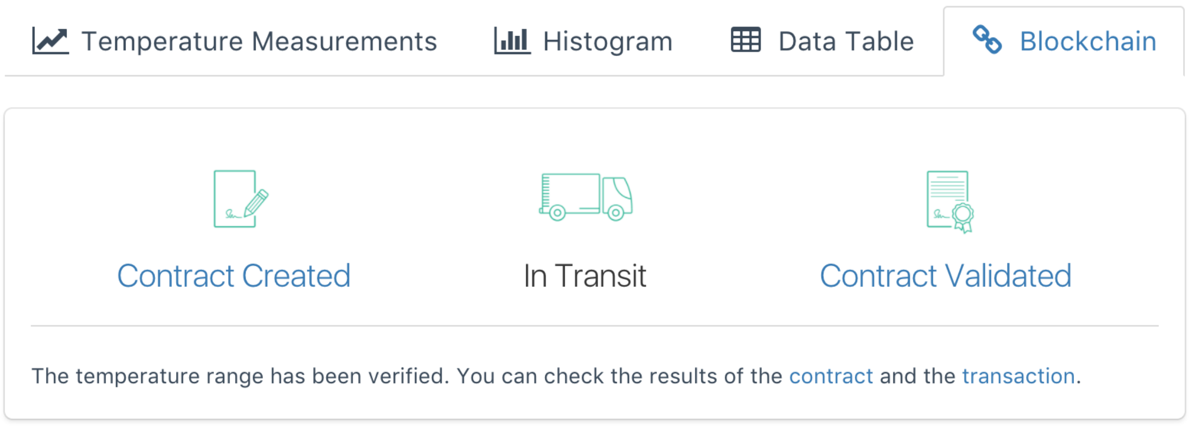
The frontend changes mainly mirror the adaptions which were necessary based on changes of the backend including the adaption of endpoints and model mappings. Based on the usability issues identified by the workshop several improvements were implemented to address those. On a more structural level the biggest changes were based on the permission system from the backend, where the frontend receives a list of permission the user has and the frontend should only show navigation elements or actions like buttons if the user is actually allowed to see them.

Blockchain visualization is an additional feature which should give the user access to information about the involved smart contract which were previously not available in the prototype. The initial design idea was to include all the interactions from the blockchain, IoT and shipment process perspective enhanced with animations and with help text on



The shipment is currently in transit.

Figure 4.1: Blockchain visualization of a shipment in transit



The temperature range has been verified. You can check the results of the contract and the transaction.

Figure 4.2: Blockchain visualization of a shipment with all steps completed

hover. However discussions with users showed that it not only did not serve the purpose well, but actually had the opposite effect as intended. It was overly complex to them and they were not able to find the information they wanted.

Therefore the new goal was a visualization with minimal information while still conveying the shipment process and status of the blockchain events, which Figure 4.1 and Figure 4.2 show. This extremely simplistic visualization can be understood easily and includes the two blockchain events of creating a smart contract and executing it, which validates the measurements of the shipment. It is important to note that those three events in Figure 4.1 not necessarily have to happen linearly. The colors grey (not yet done), black (in process) and green (completed) symbolize the progress. It is entirely possible that a shipment is already in transit, but the creation of the smart contract is not yet finished. Such events can be represented in this form. Access to more specific blockchain related information is given by linking to Etherscan, a block explorer for the Ethereum blockchain, with the corresponding hashes for the transactions.

Chapter 5

Evaluation

This chapter introduces the results of several theoretical evaluations which might solve open problems to the modum system. Technologies for the front- and backend are compared based on if they meet the requirements and could be used for implementing the system. Ways to share the stored information with customers and replicate them are researched. Finally the usability of the dashboard is computed based on the software usability scale (SUS).

5.1 Backend Evaluation

The requirements obtained in the workshop (see Chapter 3) showed the need for fundamental changes in the backend. This opportunity allowed to analyse and reevaluate the backend technologies and frameworks, validate previous assumptions on the backend and confirm that the initial reason to chose the technologies was still valid.

5.1.1 Backend: Golang Web Frameworks

The opinions about web frameworks drift quite far apart, one sees them as monolithic disease while other see it as time saving blessing. The truth is both are probably correct and it is the exact same reason why one person likes them while the other hates them. Web frameworks tend to be quite opinionated on how they expect for example project structure, which dependencies are necessary and how certain things have to be implemented. For that they provide easy riding an do a lot of the heavy lifting behind the scene. It is possible to achieve good results in a short period of time and for most problems others have already encountered them and written a solution or workaround. However this also comes to the price of freedom to decide on all the parts and can be frustrating and time consuming when something has to be fit into the boundaries of the framework for which it was not intended for.

For those reasons a comparison of common Go web frameworks and the prototype system was made to evaluate the current status of web frameworks in Go and if one is suited for

Table 5.1: Go web framework evaluation Mai 2017

	Buffalo	**Revel + GORP**	**Rest-Layer**	**Beego**	**Goa**	**modum Prototype**
REST support	5	5	5	5	5	5
JWT support	0	0	5	0	5	5
Authorization	4	2	5	2	2	0
Pagination	0	0	5	5	0	0
Sorting	0	0	5	0	0	0
Filtering	0	0	5	0	0	0
ORM	5	2	4	5	3	3
Referential data integrity built in	?	3	?	5	0	0
Testing	5	5	5	5	5	5
API Documentation	0	0	3	5	5	0
DB migrations	5	0	?	5	0	0
Active Development	5	4	2	5	5	0
Additional Comments	no jwt	stagnating contributions	only MongoDB and not actively developed		code generation from definition + customization	structure not consistent
Result	29	21	44	42	30	18

the development of the modum backend. Table 5.1 shows rating criteria and how much they are covered by the frameworks on a Likert scale from 0 to 5, where 0 represents not implemented and 5 desirable implementation. Unfortunately no framework has all the necessary features and they seem not enough developed yet, when compared to web frameworks of other programming languages. Some things are just not possible in a simple way, because of the lack of generics in Go which leads to a lot of duplicated code in most frameworks.

Based on the results, none of the web framework was usable for the modum backend, or not without painfully fitting the requirements into the framework boundaries, which led to the decision to build the backend based on the standard library of Go and specific third party packages. This allows to use any library by self wiring them together, but comes with the tradeoff of needing more development time for finding and evaluating the correct libraries.

In October 2017 the progression of those frameworks was rechecked and most frameworks made a lot of progress. Some do now fullfil the initial requirements which lead to an additional implementation of the backend based on 'Beego' to see how much effort it is to reproduce the backend with the framework. While the development was simplified in many aspects especially via automatically generate code based on models, there were again limitations which were hard to work around. As an example, the framework 'Beego' does not work with some data types which are not supported, like an array of bytes. Other limitations show that those frameworks can not yet serve a bigger and complex use case or do not provide capabilities to extend the feature set, which makes them currently not useable for modum.

5.2 Frontend Evaluation

The frontend of the modum system consists of the web client as well as the mobile client. A way to reduce the effort and improve the maintainability of the mobile client is researched.

5.2.1 Cross Platform Mobile Client

The mobile client is currently an important part of the modum system and serves as bridge between the sensor device and the backend as shown in Figure 2.2. It is mainly involved with initiating the recording of the sensor device according to definable metrics and reading out and transferring the data to the backend on the receiving side.

Pilot projects of modum with customers have shown that the diversity of mobile operating systems is approximately uniformly distributed between iOS and Android. Furthermore employees are not willing to user their private phone for work and especially don't want to install work related applications on them. This means for modum in order to reach a broad number of customers both platforms have to be supported and maintained. This issue was circumvented for the pilot projects, to not slow the development, by providing Android phones and concentrating the development on one platform. In a separate project a prototype iOS client was developed as proof-of-concept which showed problems ranging from the effort involved to maintain two separate mobile applications to how corporate identity can be maintained in both of them. This demonstrated that at the current moment it is not possible to maintain two separate platforms with the limited resources at hand.

Cross platform mobile clients were identified as a potential solution to effectively maintain multiple platforms with a single code base. Generally two types of cross platform applications can be differentiated, (1) native cross platform apps and (2) hybrid "HTML" cross platform apps. Additionally a new special form of hybrid cross platform apps "progressive web apps" has recently become popular.

5.2.2 Native cross platform apps

Mobile operating systems have their own preferred programming languages which are supported by the OS vendor. In case of Android this is Java and for iOS it is Objective-C and Swift. The vendor creates for those a software development kit (SDK) in order to create mobile apps. Applications developed by using the official SDK are generally called "native apps".

However development is possible in programming languages not supported from the vendor by using the API (Application Programming Interface) provided by the native SDK. A platform for creating "native cross platform apps" provides a uniformed API on top of the different SDK APIs which then allows to build from the same source for different target platforms.

Prominent examples for "native cross platform applications" are:

Xamarin was founded in 2011 and has been bought by Microsoft in 2016. It uses C# as its programming language and can build native applications for Android, iOS and Windows.

Appcelerator Titanium founded in 2008, is one of the earliest players in this segment and was initially intended for developing cross-platform desktop applications. However they added support for iOS in 2009 and for Android in 2012. It uses JavaScript as its programming language.

NativeScript is a open-source framework which was first released in 2015. It uses JavaScript as main development language, but supports angular and typescript and uses CSS for styling.

QT is one of the oldest cross platform frameworks for desktop around, released in 1995. Support for Android and iOS was added in 2013. QT uses C++ and its own markup language QML (Qt Modeling Language) which is similar to HTML. However Qt components don't adhere to the native style of Android and iOS.

RubyMotion is a commercial implementation of the ruby programming language that runs on mobile operating systems and was first released in 2012 for iOS. Support for Android was added in 2014. It works exclusively on OSX.

5.2.3 Hybrid HTML cross platform apps

Hybrid cross platform applications leverage the WebView component of iOS and Android to display the graphical user interface (GUI) and provide custom plugins to access the native API & sensors. This allows developers to use and potentially reuse standard HTML5 technologies from their web app to design their mobile app. The hybrid framework provides a JavaScript to native bridge to access files, use geolocation, camera access or the likes.

Cordova originally called PhoneGap which launched in 2009 from Apache, is one of the most popular hybrid cross platform frameworks. Its plugin system supports most of the modern native functionalities. The use of standard HTML & CSS gives access to a lot of reusable resources as well as themes based in CSS which mirror the native look of the operating systems. The static HTML and CSS files are shipped with the native wrapper around the application, which allows it to render instantly like a normal native application.

5.2.4 Progressive Web Applications

A glimpse on the future for handling cross platform applications could be progressive web apps (PWA). It is a new, modern concept for building web apps which blur the boundaries of the web and native applications. The concept is actively pushed from Google. It resembles the “mobile first” paradigm which ensured that mobile users have a pleasant experience with web pages. Progressive web apps should bring this pleasant feeling for web applications on mobile by heavily leveraging browser capabilities. It can be seen as a special case of hybrid cross platform applications in which the mobile browser takes the role of the hybrid framework. They idea is that the mobile browser can display the application full screen and also install it to the homescreen without the need of an app store. The plugin system is also provided by the browser with features like web bluetooth or webRTC.

For this functionality PWA make use of a design called “application shell” which is the minimal statically structure (HTML, CSS nad JavaScript) which is needed to show something meaningful which is aggressively cached locally. It can be compared with what is shipped in an app store or with the minimum of a single-page application where afterwards only data is sent and rendered. This allows the “application shell” to also work when no internet connection is available and no new data can be fetched. By also introducing a new JavaScript concept “service worker” which can manage background tasks like push notifications and background sync, PWA can be used for most applications.

Unfortunately the development of those concepts is not mature yet, but a lot of those are currently supported by Android devices and Apple has announced the start of bringing them to iOS.

5.2.5 Evaluation

Cross platform application frameworks have the potential to enable access to multiple target platforms efficiently and exploit reusability. Despite those advantages there are reasons to avoid cross platform applications. Research has found results that indicate that hybrid cross platform apps tend to be more prone to user complaints than native cross platform apps [16]. Also native applications seem to have generally less complaints about performance and reliability than cross platform applications [16]. The following lists showcase some of the positive and negative aspects.

Positive aspects of cross platform applications include:

- Code can be reused on different platforms allowing to develop faster and reduce costs.
- It can effectively reduce the amount of maintenance necessary, *e.g.* when a bug is found and this can be resolved with one change for multiple platforms.
- Tests have to be only written once for common code, reducing the work for testing.

- Existing programming experience can be used in a language, rather than invest time in learning a new language and environment.

Negative aspects consist of:

- Performance issues for many mid level and entry level phones that don't have enough hardware power to perform smooth HTML5 animations (this factor is slowly diminishing with technological advance).
- Higher energy consumption. Modern HTML and CSS features like gradients require a lot of CPU and GPU resources. A native apllications has generally lower battery consumption than a hybrid application.
- Achieving native look and experience can be challenging to achieve. This depends to the capabilities of the used framework too.
- Although there is only a single code base, the complexity for that code base can increase.
- Development of native mobile operating systems is fast paced. Resulting in features which have to be included in a cross platform framework which can require time. Users might request newer features immediately.

For the case of the modum two features are extremely important: (1) Support for bluetooth low energy and (2) background service to manage tasks at a later point. The reason for their importance is that the mobile client is the data bridge from the sensor device to the backend and therefore needs to connect and read data from the sensor device via BLE. In the operating environment of the application, mainly storage houses, the internet connection has been proven to be relatively unreliable. The mobile application has to handle connectivity issues and upload the data as soon as possible. Table 5.2 shows a comparison of the introduced platforms according to those requirements.

5.3 Public / Private Blockchain

Blockchain technology can be separated, among other criteria, on who is allowed to participate in the network. A public blockchain network is completely open and anybody who wants to participate can do so. It usually contains an incentivizing mechanism to encourage more participants to join the network. However this open nature comes at a price of substantial amount of necessary computational power to maintain a distributed ledger at large scale. Another drawback for some applications, especially for enterprise use, can the public nature of the blockchain itself be. Although it might seem counter intuitive to trade the essential public part of a blockchain, which is a the fundament for it being reliable and trusted, to restricted access and permissions, this is exactly what a private blockchain does. There are considerable discussions in the community about the value of private blockchain and how (or even if) it differentiates from a shared databases.

Table 5.2: Comparison of cross platform solutions for modum

Name	**Type**	**Language**	**BLE**	**Background Service**
Xamarin	native	C#	yes (iOS 7.0+ and Android 4.3+)	yes
Appcelerator Titanium	native	JavaScript	yes (iOS 8.1+ and Android 4.0+)	yes (separate implementation for each platform)
NativeScript	native	JavaScript, Angular and TypeScript	yes (iOS and Android 4.3+)	Android only
QT	native	C++ and QML	yes (iOS and Android)	Android only (Qt 5.7+)
RubyMotion	native	Ruby	yes (experimental and community maintained)	no (can be implemented for oneself)
Cordova	hybrid	JavaScript, HTML and CSS	yes (iOS and Android 4.3+)	yes (community maintained)
Progressive Web Apps	hybrid	JavaScript, HTML and CSS	yes (only experimental yet)	yes (not useable for complex tasks yet)

Public and private blockchain still share many commonalities like constituting of a decentralized distributed peer-to-peer network, where each participant maintains a replica of all the transaction of the shared ledger and provide measures to handle faulty or malicious participants. Such features can also be seen as advantage for a private blockchain over a shared database.

As introduced in Section 2.2 the modum system stores two values in the public blockchain: (1) the outcome (yes or no) if any temperature excursions from the defined range occurred and (2) the hash of the measurements. The information inserted is limited for privacy concern as well as the economic cost of inserting huge data into the public blockchain. However there is still the need to provide customers access to the recorded data regularly apart from the dashboard. Private blockchain technology allows to share this information with a permission system, efficiently ensuring that only authenticated users can access the data. Although private blockchain technology does meet the requirements for the modum system, it is not clear if it does meet the requirements of GDP. Section 2.1 introduced the GDP guidelines in which chapter 4 mentions the requirement to delete or anonymise personal data after 5 years. A blockchain system however is a immutable append-only data structure where it is impossible to remove data.

Table 5.3 compares several distributed storage options which could, as an alternative to private blockchain, be used to share the relevant information with customers and while also serve as replication of datasets. However they have their own issues, which makes them impractical or unusable in compliance with GDP. The requirements are authentication and authorization to access data as well as later mutability to comply with privacy concerns mentioned in GDP. However no solution from Table 5.3 seems practical, because they either don't allow for file deletion or have no sharing capabilities which would make it impossible to revoke access when keys are shared. Also adhering to the privacy concerns can prove to be difficult in distributed storage, because it is extremely difficult to ensure that no node still has a copy. The best option with decentralized distributed storage seems to be Sia as soon as they have implemented the sharing capabilities, which is planned for November 2017. Although distributed storage is an interesting topic in itself it might not be suitable for modum to build a completely compliant product with GDP.

5.4 System Usability Scale of the Workshop

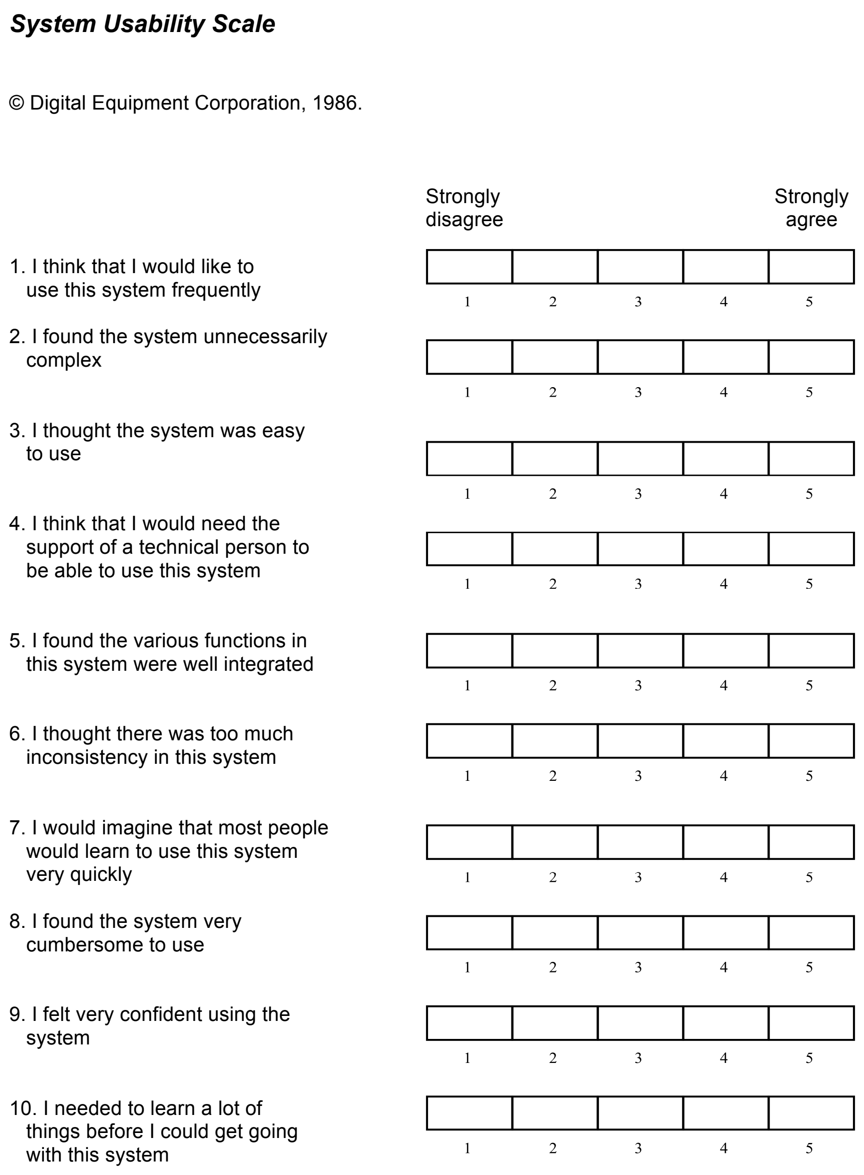
The System Usability Scale (SUS) was designed by John Brooke in 1986 as a "quick and dirty" measure for the perception of usability and has become a defacto standard in the industry since then [17]. SUS consists of a 10 item questionnaire with 5 positively and 5 negatively phrased questions. Figure 5.1 shows the questions which can be answered on a Lickert scale from 1 to 5 which represents from "strongly disagree" to "strongly agree".

The result of SUS is a single number representing the measure of the overall usability of the system. The values of the questions do not hold any meaning on their own, only the composite of them does. In order to calculate the SUS score, the contribution of each item has to be summed. The contribution of each item can be between 0 and 4 and is obtained for the questions 1, 3, 5, 7 and 9 through subtracting one from the value of the questionnaire. For the questions 2, 4, 6, 8 and 10 the contribution is five minus the value of the questionnaire. The so obtained sum is then multiplied by 2.5 to obtain the overall SUS value which can be in the range between 0 and 100. The calculation for participant 1 in Table 5.4 is given as an example: $((3-1)+(5-4)+(2-1)+(5-4)+(3-1)+(5-3)+(2-1)+(5-4)+(2-1)+(5-4))*2.5 = 32.5$

The interpretation of the total value by SUS is not contained in the original paper. However experience from the last 30 years has shown that the average is around 68, where a value above means a usability above the average of general software usability and a value below means a worse usability than generally in software. When interpreting the results in Table 5.4 it shows that for two users the usability of the dashboard was bad and for three users it was acceptable. This indicates that the dashboard has to be improved to make it easier understandable and usable.

Table 5.3: Comparison distributed storage June 2017

Name	**Type**	**Access Control**	**Files can be deleted**	**Sharing**	**Blockchain**	**Maturity**	**Cost**	**Remarks**
Storj.io	Decentralized distributed storage	End-to-end encrypted	Yes	only possible by sharing secret keys	Used to incentivize sharing storage (Counterparty)	Operational	0.015/GB/month, 0.05/GB download (earn money by sharing storage)	
Sia	Decentralized distributed storage	End-to-end encrypted	Yes	only possible by sharing secret keys (simple sharing feature planned for November)	Used to incentivize sharing storage (Siacoin –>own blockchain)	Operational	No information found (earn money by sharing storage)	
Swarm	Decentralized distributed storage	Provides integrity protection by content addressing	No	knowing the hash allows anyone to access the files	Used to incentivize sharing storage (Ethereum)	Alpha	No information found	Must run on own node. Part of Ethereum ecosystem.
IPFS	Decentralized distributed storage	Provides integrity protection by content addressing	No	knowing the hash allows anyone to access the files	No	Operational	No information found	Private IPFS possible. Own node necessary.
MaidSafe	Decentralized distributed storage	End-to-end encrypted	Yes	only possible by sharing secret keys	Used to incentivize sharing storage (Safecoin)	Alpha	No information found (earn money by sharing storage)	
Filecoin	Decentralized distributed storage	End-to-end encrypted	No information	only possible by sharing secret keys	Used to incentivize sharing storage (Filecoin)	Not started	No information found (earn money by sharing storage)	Sister protocol of IPFS
BigChainDB	Decentralized distributed storage	Public or Private	No	On an account level	Blockchain like from feature set (immutable, decentralized, ...)	Alpha	No information	Has different permissions



System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Figure 5.1: Questionnaire of the System Usability Scale [17]

Table 5.4: SUS ratings for the participants of the workshop

Question	**Participant 1**	**Participant 2**	**Participant 3**	**Participant 4**	**Participant 5**
1	3	4	4	3	4
2	4	3	2	3	2
3	2	3	4	3	3
4	4	3	2	2	1
5	3	4	4	4	3
6	3	2	2	2	2
7	2	3	5	3	3
8	4	3	2	2	2
9	2	3	4	3	3
10	4	2	2	2	1
Result	32.5	60	70	47.5	70

Chapter 6

Summary and Conclusions

Blockchain technology has not yet achieved broad public acceptance. Some still perceive it as technical playground. Like with many fundamental changes it happens slow and has to face a lot of resistance. Blockchain technology does not make sense for everything, however it could solve problems like digital identity or digital ownership faster and more secure.

modum has found a specific use case where a real problem exists and their system can help to solve that more efficiently and cheaper, while also contributing to making the usage of medicinal products safer for the end consumer.

This thesis has made the following contributions to the modum system:

- Consolidated lists of features to implement, usability issues and minor bugs which were discovered in the client workshop. They have been added to the internal feature and bug tracking system of modum and were rated based on the importance for the final product.
- Computing and analyzing the system usability scale (SUS) of the web client.
- Analyzing web frameworks in Go on their suitability for the backend of modum.
- A complete rewrite of the backend code in Go with prioritizing software quality and long-term maintainability.
- A flexible permission system where permissions are predefined and users or admins can create roles to which permissions can be assigned.
- The respective frontend adaptions for the backend improvements as well as new features like the permission.
- Implementing a minimal information blockchain visualization.
- Theoretical evaluation of which cross platform platforms are available and if they fullfil the current requirements of modum.

- Theoretical evaluation if private blockchain technology would be suitable to share information with clients.

Several aspects of thesis could and should be refined in a further work. The results and findings of the workshop have been extremely important to the development of the whole modum system. Therefore another workshop should be planned for when the system is nearing its completion. The blockchain visualization in its current minimal form does not convey any help to better understand the blockchain technology. The user needs additionally to have knowledge on what Etherscan exactly shows them to make full use of the information. A workshop to find what the users want and expect to see with respect to the blockchain technology might prove beneficial. Several possible technologies were presented and evaluated on their suitability for a cross platform mobile application for the modum system. A prototype should be built with one of those. Finally Sia was based on the results best suited for the use case of modum. A prototype of information sharing based on Sia should be implemented as soon as it contains the sharing capabilities.

Bibliography

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. May 2009.

[2] Roger Wattenhofer. *The Science of the Blockchain.* CreateSpace Independent Publishing Platform, USA, 1st edition, 2016.

[3] Sarah Underwood. Blockchain Beyond Bitcoin. *Commun. ACM*, 59(11):15–17, October 2016.

[4] Karan Bharadwaj. Blockchain 2.0: Smart Contracts. 2016.

[5] Modum Whitepaper. `https://modum.io/wp-content/uploads/2017/08/modum-whitepaper-v.-1.0.pdf`, 2017.

[6] European Commission. Guidelines of 5 March 2013 on Good Distribution Practice of medicinal products for human use (2013/C 343/01). `http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX%3A52013XC1123(01)`, 2013.

[7] Good Distribution Practice of Medicinal Products (GDP Guidelines). `https://www.bag.admin.ch/bag/en/home/themen/mensch-gesundheit/biomedizin-forschung/heilmittel/aktuelle-rechtsetzungsprojekte/gdp-leitlinien.html`, 2015. Online; last visited 10. November 2017.

[8] Chronicled Webpage. `https://chronicled.com`, 2017. Online; last visited 14. November 2017.

[9] Ubirch Webpage. `https://www.ubirch.de`, 2017. Online; last visited 14. November 2017.

[10] Ambrosus Webpage. `https://ambrosus.com`, 2017. Online; last visited 14. November 2017.

[11] F. Paetsch, A. Eberlein, and F. Maurer. Requirements Engineering and Agile Software Development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pages 308–313, June 2003.

[12] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton, Jr. N Degrees of Separation: Multi-dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 107–119, New York, NY, USA, 1999. ACM.

[13] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces.* " O'Reilly Media, Inc.", 2011.

[14] T. C. Lethbridge, J. Singer, and A. Forward. How Software Engineers use Documentation: the State of the Practice. *IEEE Software*, 20(6):35–39, Nov 2003.

[15] Ehud Reiter, Chris Mellish, and John Levine. Automatic Generation of Technical Documentation. *Applied Artificial Intelligence*, 9(3):259–287, 1995.

[16] Iván Tactuk Mercado, Nuthan Munaiah, and Andrew Meneely. The Impact of Cross-platform Development Approaches for Mobile Applications from the User's Perspective. In *Proceedings of the International Workshop on App Market Analytics*, WAMA 2016, pages 43–49, New York, NY, USA, 2016. ACM.

[17] John Brooke. SUS: A Quick and Dirty Usability Scale. 189, 11 1995.

[18] Progressive Web Applications. `https://developers.google.com/web/progressive-web-apps/`, 2017. Online; last visited 14. November 2017.

List of Figures

List of Tables

Appendix A

Installation Guidelines

Note that the interaction with the system will be very limited and data can not be generated because of the missing modum sensor device and mobile application. Both, running the backend and frontend, are required to operate on the dashboard.

A.1 Sources

- The backend sources can be found on the CD in the folder “modum-backend” or online at `https://github.com/arddor/modum-backend`
- The frontend sources can be found on the CD in the folder “modum-web” or online at `https://github.com/arddor/modum-web`

A.2 Installation Backend

Postgres and Golang have to be installed. Note that Go is very restrictive for the project environment. For information on how to install Go please refer to `https://golang.org/doc/install`. The modum-backend folder has to be placed exactly in the folder `$GOPATH/src/github.com/arddor/modum-backend` or the files can not be compiled (the easiest way to correclty install them is via `go get github.com/arddor/modum-backend`).

1. Set Postgres information in the conf/app.conf file of the modum-backend folder (They start with pg, *e.g.* pguser for the user of the postgres database).
2. Install all the Go dependencies by executing `go get ./...` in the modum-backend folder
3. Run `go run main.go --help` to see the modum specific commands
4. First the database migrations have to be executed by `go run main.go migrate`

5. A company and super user should be created for being able to login by `go run main.go addCompany` respectively `go run main.go addAdmin`

6. Finally the server can be run with just executing `go run main.go`

7. The swagger documentation can be found at `http://localhost:8080/swagger`. (Note the backend and frontend are completely separated. Run both to interact with the backend over the dashboard)

A.3 Installation Frontend

Node with NPM has to be installed.

1. Execute `npm install` in the modum-web folder

2. In order to a server which serves the static files run `npm run dev`

Appendix B

Contents of the CD

1. Source files of the thesis in the folder “thesis”
2. The thesis in PDF format as “Masterarbeit.pdf”
3. Abstract in german as “Zusfsg.txt” and in english as “Abstract.txt”
4. Source files of the modum backend system in the folder “modum-backend”
5. Source files of the modum frontend system in the folder “modum-web”

Appendix C

Workshop

The following findings have been identified in the client workshop described in Chapter 3. Table C.1 shows a categorized and rated list of feature requests which were made by the clients. Likewise Table C.2 shows usability issues which were either observed or mentioned directly by the clients and Table C.3 shows bugs which were found during the workshop session. All Tables contain ratings of the set of ‘Low‘, ‘Medium‘, ‘High‘ and ‘Critical‘ which indicates the importance and impact it has on the final product.

Table C.1: Feature requests identified in the workshop

Topic	Description	Rating
PDF report selection	In Elpro, one can choose what should be in the PDF report (Picture, Data). There should be a way to choose what the user wants in the report.	Low
Multiple sensors per shipment	Like Elpro. Possible functionality (compare function, show graphs single or combined)	High
Meta information about the sensor	Meta information about the sensor should be accessible (how long is the sensor valid, calibration certificates, battery performance). This information should also be in the report	Critical
Adapt calculations in graph	The calculations in the graph should be adapted to the Zooming Level, like Elpro does.	Medium
Sensor management and planning	A tool for planning and calculating how long a sensor can be in transit under certain circumstances.	Medium
Non critical mode (statistics mode)	Additional mode for making measurements (without blockchain)	Low

Continued on next page

Table C.1 – continued from previous page

Topic	Description	Rating
Approving shipments	All shipments must be approved by the responsible person. The reasons for approved must be available and be stored if the measurements were out of the boundaries. Also a disapprove is possible with comment	Critical
Filter directly under column	The filter should be directly under the title of the column and would only apply to this column.	High
Jump to TOP	A button which allows to move back to the top quickly.	Low
Data table header and footer should be fixed	Data table header and footer should be fixed and only the data scrollable	High
Role / permission management	Different roles must be definable: • Not all roles may see temperature curves • Phones should have role management too, permission to see data or result can be time limited • Admin and Admin IT should be seperated • Web Access should be only available from certain roles upwards • Operator may set predefined temperature and alarm criteria • There should be an Audit Role (read only)	Critical
Audit trail	Everything must be logged, resulting in a complete audit trail for any shipment.	Critical
Login log	Every login has to be registered	Low
Custom alarm criteria	Criteria should be definable under which an alarm (user notification) is issued, e.g. more than 5 minutes +/- 3°C	Medium
Trends	Trends have to be deductible. It should be apparent how the general behavior of shipments on the same route has changed	Medium
Realtime alerts	Alerts are send in realtime, while the shipment is still in transit	Medium

Continued on next page

Table C.1 – continued from previous page

Topic	Description	Rating
Realtime temperatures	The temperature is sent in realtime to the backend, allowing to see the temperature graph and all the information while the shipment is in transit	Medium
Fraud detection	Detect fraud attempts, e.g. sensor is not actually sent with shipment	Medium
GPS tracking	Always know the current position of the shipment	Low
GPS history	GPS location of single checkpoints	Medium
Comparison shipment temperature with environment temperature	Gaining more insight in which environment the shipment was.	Medium
Audit trail in the report	The change information have to be in the pdf report and a new generation of pdf report has to be versioned and the information of the changes have to be there	Medium
Timed role assignment	A role can be changed and would switch back after a certain amount of time	Low
Configurable dashboard	The dashboard should be configurable so that the user can rearrange widgets to his own likings	Low
Data import	Users want to import existing data and analyze it with the Modum system	Low
Status ”processed”	Status which shows that shipment has been processed by a responsible person (when, who)	Critical
Print Functionality	The user wants a separate print functionality (**Not the same as PDF report**)	Low
Manual input	Possibility to enter data manually, e.g. import data on drugs	Low
Fuzzy area of the sensor	The inaccuracy of the sensor should be displayed in a ”fuzzy area”: The sensor should send his inaccuracy area by itself	High
Spatial mapping of multiple sensors	Best practice with multiple sensors is to have 9 points in a cuboid. A visual representation on how it was during the shipment	Low
Sensor delay	Start sensor with a definable delay so that only relevant data is in the set or possibly the sensor could adjust to drastic temperature changes	Medium
More specific filters	Additional filters, e.g. too hot or too cold, time out of spec	Medium

Continued on next page

Table C.1 – continued from previous page

Topic	Description	Rating
Analysis functionality	For example comparison by receiver, transport company etc.	Low
Planned delivery time	Show an expectation of when the shipment should be delivered	Low
Planned shipment destination	Currently the destination location is shown retrospectively after the shipment was received. Show where the shipment is planned to go	Low

Table C.2: Usability issues identified in the workshop

Topic	Description	Rating
Double click	Windows users are accustomed to double click which interfered on some components on the web application (e.g. row expands and collapses again).	High
Right click	Windows users expect an options menu on right click (not the options menu of the browser)	High
Date format and timezone	The date format should be displayed without ambiguity and clearly, timezone and local times also.	Critical
Measurement units	Measurement Units should be displayed everywhere without ambiguity.	Critical
Temperature limits	In the temperature graph, the temperature limits are not displayed clearly. In the information, the temperature limits are not contained	Critical
Round measurement units to 2 or 3 digits	User want a sane default like 2 digits after comma, but still be able to change it under settings.	Critical
Graph library is not intuitive enough	For example: Zooming was not understandable for some people. The features in the rop right row of the graph library have not been seen	High
Unclear filter behavior of substrings	It is unclear whether the filter represents ”includes” or ”equals”	High
Expectation of state persistency	It is expected that the state of everything is persistent when tabs are changed	Medium
Detail Page not found	The detail page was not found by a user	Low
Activation temperature	The input field and update behaviour of the activation energy confused users	Medium

Continued on next page

Table C.2 – continued from previous page

Topic	Description	Rating
Filter behavior not intuitive	AND, OR, Substrings, Equals, not equals, greater than... how to defined them	High
Data table columns adapt themselves	It was confusing for users that when they sorted the data differently, the width of columns adapt themselves	High
Status Received Success and Mined Success	Should be displayed together for the end user	Critical
Expanding of row not intuitive	It is not intuitive that a click on a row reveals more information about a shipment	High
Temperature Boundaries not visible	The written description, e.g. 25°C, of the temperature boundaries is not always visible	Critical
Detail Page: Not clear how to get back	It was for some users not intuitive to use the browser back button to go back	High
More information in web app for user	Users expressed the wish to have more information available, e.g. sensor information (battery, condition, ...) on hover	Critical
Kelvin Minutes	Kelvin Minutes should be displayed in minutes and seconds, not fraction	Critical
Customizability (settings)	For example: change which columns are shown or language switching	Low
Handle temperature deviation similar to Elpro	Deviations have to be labeled; - User can / have to set a comment for deviations; - the table of deviations as well as each individual deviation has to be approvable	Critical
Improve temperature string	A temperature string which indicates deviations and by how much was added to the row, however this has to be improved to make it more intuitive	High
Dashboard has too much redundant information	Some information is repeated in different places	Low

Table C.3: Bugs identified in the workshop

Topic	Description	Rating
Date is not visible in the time column of the exported Excel File	If the temperature measurements are exported as an excel file, there is a time column but the date is not visible. (excel formatting issue)	Critical
Company is not shown in detail page of shipment		Critical
Filter by ”approved”	The filter does not work	Critical