Universität Regensburg

A fully traceable Supply Chain

based on Smart Contracts

Bachelorarbeit im Fach Informationswissenschaft am
Institut für Information und Medien, Sprache und Kultur (I:IMSK)

Vorgelegt von: Korbinian Kasberger
Adresse: Arnulfsplatz 9, 93047 Regensburg
E-Mail (Universität): korbinian1.kasberger@stud.ur.de
E-Mail (privat): korbiniank@outlook.de
Matrikelnummer: 1600695
Erstgutachter: Prof. Dr. Christian Wolff
Zweitgutachter: Prof. Dr. Niels Henze
Betreuer: Dipl. Math. (FH) Ottmar Amann (Krones AG)
Laufendes Semester: SS 2018
Abgegeben am: 20.07.2018

1 Content

2 Glossary

API.....*Application Programming Interface*
CLI..... *Command Line Interface*
ĐApp .....*Decentralized Application*
EAN.....*European Article Numbe*r
ERC20.....*Ethereum Request for Comment #20*
ERC721.....*Ethereum Request for Comment #721*
EVM .....*Ethereum Virtual Machine*
GLN..... *Global Location Number*
IDE.....*Integrated Development Environment*
IoT .....*Internet of Things, Internet of Things*
JS ..... *JavaScript*
JSON..... *JavaScript Object Notation*
NFC .....*Near Field Communication*
QR..... *Quick Response*
RFID ..... *Radio Frequency Identification*
SC.....*Supply Chain*
SCM.....*Supply Chain Management*
SDK .....*Software Development Kit*
WSC.....*Wine Supply Chain*
WTG.....*Wine Traceability Group*

3 Abstract

Blockchains and their addition of smart contracts are a new emerging technology, still searching for their use cases. Several corporations as well as the Fraunhofer Institute established the supply chain as a potential benefactor of this technology. This thesis aims to test this, as, while several startups and companies claim to use the technology already, little to no details are available on how requirements can be fulfilled with smart contracts. In an interactive web application, with smart contracts on the basis of Ethereum, an exemplary supply chain of the wine industry can be simulated. The smart contracts have further been tested in an interactive show booth at an internal Krones exhibition, showing the capabilities of performing in a semi-real scenario.
The results of this project show that - while the technology still has its limitations - supply chains can benefit greatly from the added trust and decentralization.

4 Motivation

Blockchains were all over the news in late 2017. While new technologies emerge on a regular basis, none previously had such a potential to disrupt entire industries. A blockchain is a decentralized peer-to-peer database of historic transactions. Every participant in this network can verify the transactions and once they are stored, it cannot be altered unless every participant in the network changes his record at the same time. With several hundreds to thousand so called *"nodes"*, this will likely never happen and since its introduction in 2008, the Bitcoin blockchain transactions have never been hacked.

The blockchain concept creates trust, as every transaction can be taken as fact. If a certain data-point was entered into the network via such a transaction, the timestamp, data and the sender of the transaction are forever connected to one another.

This enormous potential was recognized quickly by the industry and a study by Juniper Research in July 2017 revealed, that more than half of the questioned corporations larger than 20.000 employees are either actively researching blockchain technology or are already in the process of deploying it. (Juniper, 2017a,

2017b) While the technology was - and still is - hyped by many, the *"Gartner Hype Cycle for Emerging Technologies"* puts Blockchain-Technology already past its peak of *"inflated expectations"*. In this phase the generated ideas are sorted, many may turn out to be overhyped and impossible to implement and a few useful areas remain (Greenspan, 2016).

In April of this year, the Fraunhofer Institute released a positional paper, outlining several possible application areas for blockchains. Next to the most common one mentioned in this context, the financial sector, the second most promising one is the *Supply Chain Management* (SCM) and the ability to trace products. (Schütte, Fridgen, Prinz, & Rose, 2018).

With several interacting parties, that don't necessarily trust one another but still have to share data with each other, makes it an interesting research subject.

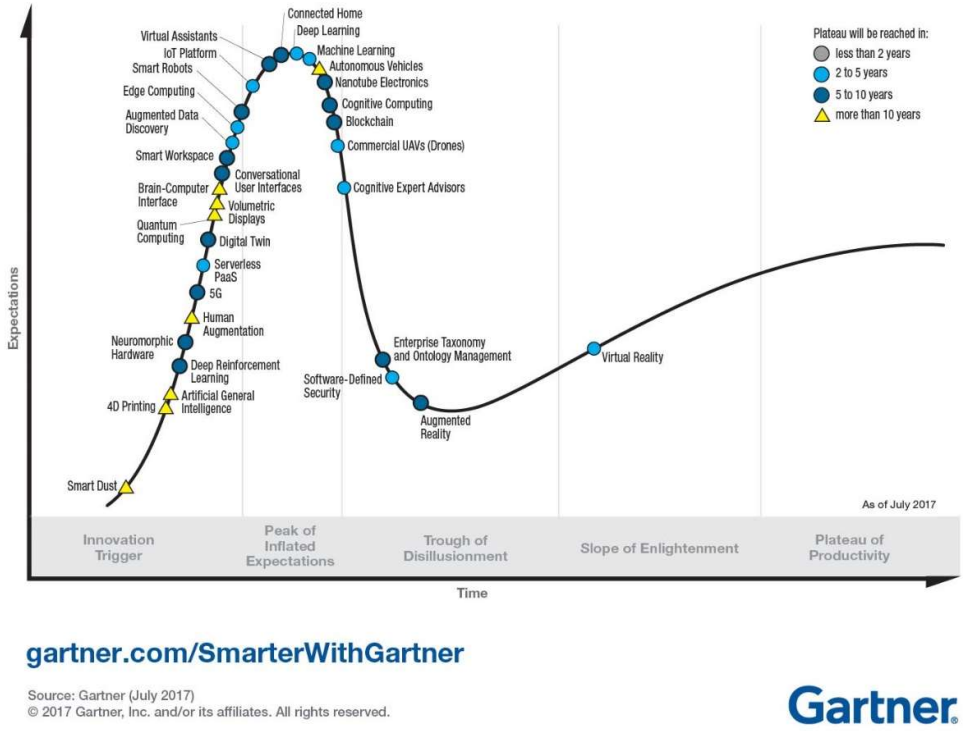

Figure 1: Gartner Hype Cycle for Emerging Technologies 2017[1]

[1] Source: https://blogs.gartner.com/smarterwithgartner/files/2017/08/Emerging-Technology-Hype-Cycle-for-2017_Infographic_R6A.jpg

Several big companies seemed to have reached a similar conclusion. DHL released their own report on the possible benefits of blockchains for logistics (DHL & Accenture, 2018).

IBM and Walmart teamed up in 2017 to eventually have their entire food supply chain traceable and using the trust a blockchain can create to ensure food safety regulations (Galvin, 2017). In early 2018 Starbucks announced, they are working on tracking their coffee beans from the farm to the cup - another ambitious project using blockchain technology (Starbucks Corporation, 2018) .

But despite these announcements, information on how these supply chains can be brought to life with blockchain technology is very scarce. Some startups offer a variety of case studies but little to no technical details. This thesis aims to find answers to a few questions on the topic and whether or not a supply chain can be extended with smart contracts.

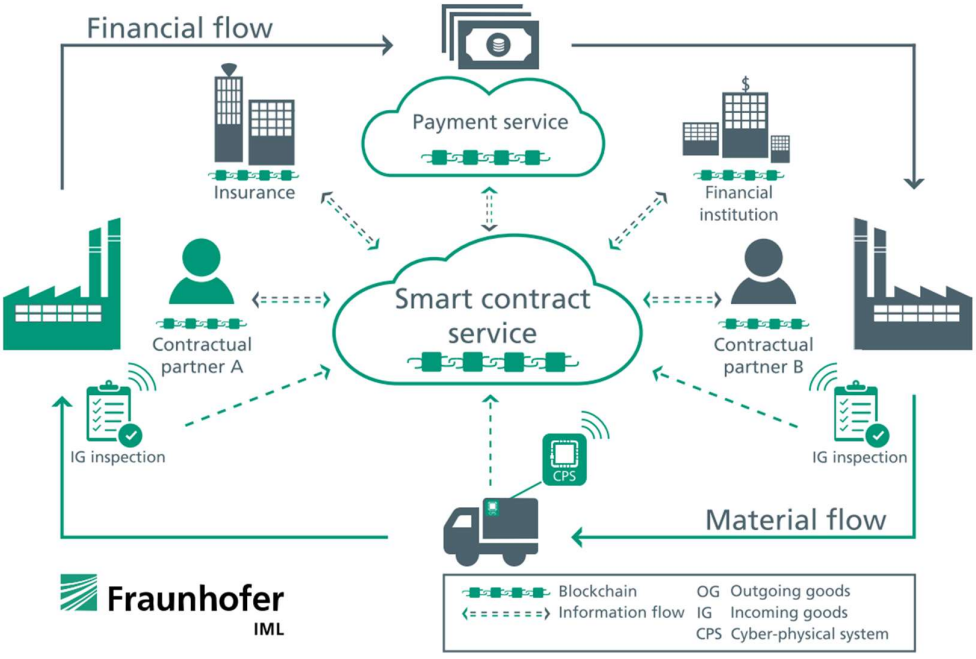

Figure 2: Blockchain-based supply chain network (Fraunhofer IML)

4.1 Relevance

Krones as one of the world leaders in manufacturing machines for the beverage industries has to investigate this technology and its requirements.

> *"In the context of Digitization, which started roughly 5 years ago with building a network of vertical and horizontal connections between Partners, Customers and Supplier, the market demands increasingly faster reaction times and implementations as well as more flexible and future-proof solutions. Simply based on the revolutionary and incredibly disruptive potential of a blockchain which, by design, includes topics such as Safety, Immutability of Information and Decentralization, it is our duty at R&D to take on this technology as early as possible. We have to be able to lay important groundwork to keep the dominant technological position, strengthen our competitiveness and solidify our innovative strength"*

Dipl.-Math.(FH) Ottmar Amann – CRD Project Manager at Krones AG

4.2 Goal

Despite the announcement of several big international companies to have their supply chains extended or supported by blockchain technology, no detailed information as to how these companies aim to realize these goals is available. Similarly, with startups in this sector[2]. In some cases only small parts of a supply chain are represented (Modum, 2018a), others only claim to be able to track everything but offer no publicly available product to verify this claim (DNV GL, 2018).

In this thesis I want to set up some basic structures for smart contracts that could be used and extended upon for a realistic use case. Supply Chains can be very complex and often hard to grasp. Therefore - as a first concept - I chose the Wine Supply Chain as an example to work with. While consisting of multiple actors[3] that interact with each other, it is still simple to understand and, being a product of expected high quality in most cases, could benefit greatly from traceability and added trust.

[2] See Section 8: Existing Block-/ Supply-Chain Solutions, p.18

[3] See Section 6: The Wine Supply Chain, p.13

4.3 Research Questions

The hype around blockchains and cryptocurrency has recently calmed down, also reflected in the previously mentioned *Gartner Hype Cycle*, Glaser calls blockchain *"an innovative technology in search of use cases"*(Glaser, 2017). While this is true, in the past 10 years since the launch of the first public blockchain in the form of Bitcoin, no real products have reached the market. However, the functionality to run decentralized code only appeared in 2015 with Ethereum. The technology still needs to mature and the ideas and possible use cases need to be evaluated.

One of these use cases, as previously mentioned, are supply chains. It is now time to test the technology for its viability and possibility of being implemented, thus leading to several research questions, this thesis aims to find answers to:

1. What is the current state of tracing in supply chains with blockchain technology?
2. What are the requirements for smart contracts to support supply chains?
3. What are the limitations that have to be faced in order to implement these smart contracts?
4. Are blockchains the right technology for supply chains?

5 Introduction Supply Chain Traceability

A supply chain consists of several manufacturers or service providers linked together by a network of physical flow of material or goods. In the last few decades globalization increased rapidly, leading to complex supply chains all around the globe and making traceability difficult.

In this section I will go over the available definitions of traceability as well as the current solution and problems.

5.1 Definition

There are several definitions for supply chain traceability (Table 1), the DIN and GS1 definition are rather unspecific while the EU regulation and the often quoted definition of T. Moe goes more into detail. For this project, I chose the latter, trying to have a product as traceable as possible.

Table 1: Definitions Traceability

Source	Definition
DIN EN ISO 9001:2015-1 (DIN Deutsches Institut für Normung e. V., 2015)	*The ability to identify and trace the history, distribution, location, and application of products, parts, materials, and services.*
GS1 (based on DIN 9001) (GS1, 2012, p. 13)	*Traceability is the ability to track forward the movement through specified stage(s) of the extended supply chain and trace backward the history, application or location of that which is under consideration*
EU Regulation (EC) N°178/2002 III Article 18, Article 3 Point 15 (European Parliament and Council, 2002)	*'Traceability' means the ability to trace and follow a food, feed, food-producing animal or substance intended to be, or expected to be incorporated into a food or feed, through all stages of production, processing and distribution*
Danish Institute for Fisheries (Moe, 1998, p. 1)	*Traceability is the ability to track a product batch and its history through the whole, or part, of a production chain from harvest through transport, storage, processing, distribution and sales […] or internally in one of the steps in the chain for example the production step*
Food Trak plc (Wilson & Clarke, 2007, p. 2)	*Food traceability can be defined as that information necessary to describe the production history of a food crop, and any subsequent transformations or processes that the crop might be subject to on its journey from the grower to the consumer's plate.*

5.2 Motivation for traceability

Having a traceable supply chain has various benefits. The primary drivers for using a traceability systems can be categorized into:

- **Improvement of Supply Chain Management** (Aung & Chang, 2014)
- **Quality Assurance** (Aung & Chang, 2014; Wilson & Clarke, 2007)
- **Identification of non-compliant items** (Opara, Vol, Opara, & Vol, 2003; Wilson & Clarke, 2007)
- **Proof of following regulations and standards** (Saak, 2016; Wilson & Clarke, 2007)
- **Proof of authenticity** (Aung & Chang, 2014; Palade & Popa, 2014)

Identifying products, especially if a problem has been detected, can greatly reduce losses. Certain products prone to counterfeiting can be verified by an unique identifier and can be traced back to its origin. Especially in the food industry it is required to have every process documented, so that when a contamination is detected, or a global recall required, the affected products can be pulled off the market quickly and the source identified.

5.3 Current solutions

To trace a product, some form of tag has to get attached to it. The standards are simple barcodes and batch numbers like EAN, that are tracked by scanning and storing the data in a database.(GS1, 2012)

Other common and more recent methods are RFID tags, that contain bits of relevant information that can either be read and re-written, or only read. (Costa et al., 2013; Farooq, Tao, Alfian, Kang, & Rhee, 2016)

A model from 2012 suggests combining the two with IoT devices and tracking the information with different database-units. (Cimino & Marcelloni, 2012, Figure 3)

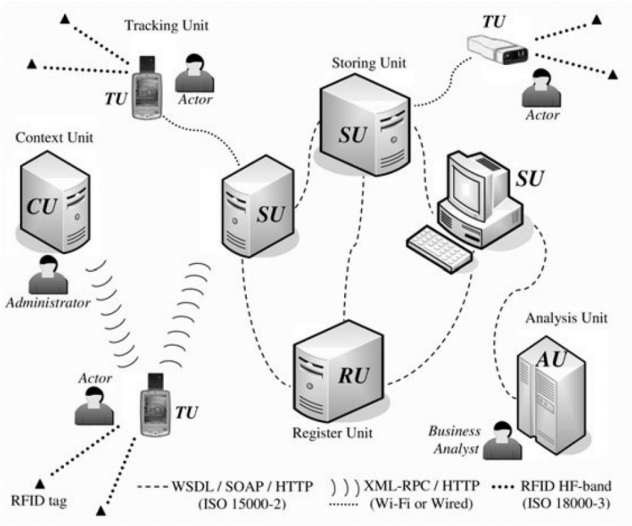

Figure 3: RFID Traceability System by Cimino & Marcelloni (2012)

Geerts & O'Leary continued the idea and developed an ontology driven model for this type of supply chain, which they called a *"highly visible supply chain"* and a *"supply chain of things"*(Geerts & O'Leary, 2014)

5.4 Problems

Every participant within a supply chain has their own interests. Especially in a global supply chain, the lack of trust can be big and the trust in received data and information is low. Other parties simply cannot verify if the data they received has not been tempered with, as a single centralized database can be manipulated rather easily. Another issue is the incompatibility of the individual SCM systems, creating *data-silos* that cannot share data between each other without great effort.

While it is difficult to copy or manipulate RFID tags, their validity is not guaranteed. If it has to be verified by a trusted third party it again opens possibilities of manipulation. Systems like the addition of video surveillance (Mao, He, Cao, Bigger, & Vasiljevic, 2015) were suggested but would greatly increase the cost and defeat the simplicity of the RFID technology. RFID tags are also limited in data-storage capacity, even though tags with several kilobytes exist, most only have a few bytes available. Only IDs, pointer to databases or timestamps could be stored, complex datasets would have to be stored somewhere else. A consumer

then has generally no way of verifying the authenticity or origins of the product as the access to companies' database would create a big security risk.

To create trust, industries have come up with a plethora of labels for consumers. The German newspaper *"Welt"* estimated over 1600 different labels in Germany alone (Dierig, 2016); A list of 463 Eco-labels[4] used worldwide, paints a similar picture. The overwhelming amount of different labels does not create trust and can cause confusion or even distrust (Bazhan, Mirghotbi, & Amiri, 2015) . There is usually no information whether or not the requirements for the label are met. In most cases consumers don't even know what the label stands for. (Label Insight, 2016)

Currently the only way to have the SC completely traceable, all the data needs to be collected in one central place. The *Efficient Consumer Response* Team (ECR), even recommended it in their 2004 report: *"To facilitate the product traceability, it is recommended to consolidate internally in **one place** (internal centralised log-file, internal **centralised database**) all the history of movements for each pallet."* (ECR Europe, 2004, p. 44 & Figure 4)

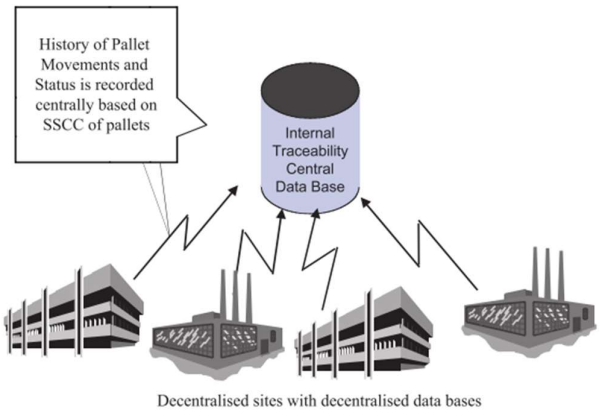

Figure 4: Centralized Database (ECR Europe, 2004)

The previously cited ontology model (Geerts & O'Leary, 2014) also relies on a central database, however in 2017 O'Leary said, that this database layer could be represented with the ledger technology of a blockchain. (O'Leary, 2017, p. 7)

[4] http://www.ecolabelindex.com/ecolabels/

Having all data in one place not only creates mistrust due to lack of control from other parties, it is also vulnerable to data-loss if those servers have technical issues or are attacked.

Collected, the problems can be categorized as follows:

- Technological issues (security, software incompatibility)
- Centralized Databases
- Lack of Trust

6 The Wine Supply Chain

6.1 Traceability in the Wine SC

In 2003 Global Standard One (GS1) "co-established the Wine Traceability Group (WTG) together with the British Wine and Spirit Association". In 2008 they published a 28 pages guideline for *"Wine Supply Chain Traceability"* (Armstrong et al., 2008) outlining the different actors within the supply chain as well as their requirements in order to achieve traceability. GS1 has also established several standards, such as the EAN numbers as well as a multitude of internationally adopted barcode systems resulting in a presence in over 100 countries worldwide. ("How we got here | GS1," n.d.). Their findings and requirements will be a basis for the development of this prototype.

The traceability in the wine industry has always been very important, due to very strict regulations such as EU 178/2002 (European Parliament and Council, 2002) and the ongoing problem of wine often being faked (Holmberg, 2010).

A project from 2009 proposed a flexible data-logger attached to bottles, that measures temperature, humidity and light. (Mattoli, Mazzolai, Mondini, Zampolli, & Dario, 2009) The data however is only stored locally on the circuit board.

6.2 Actors

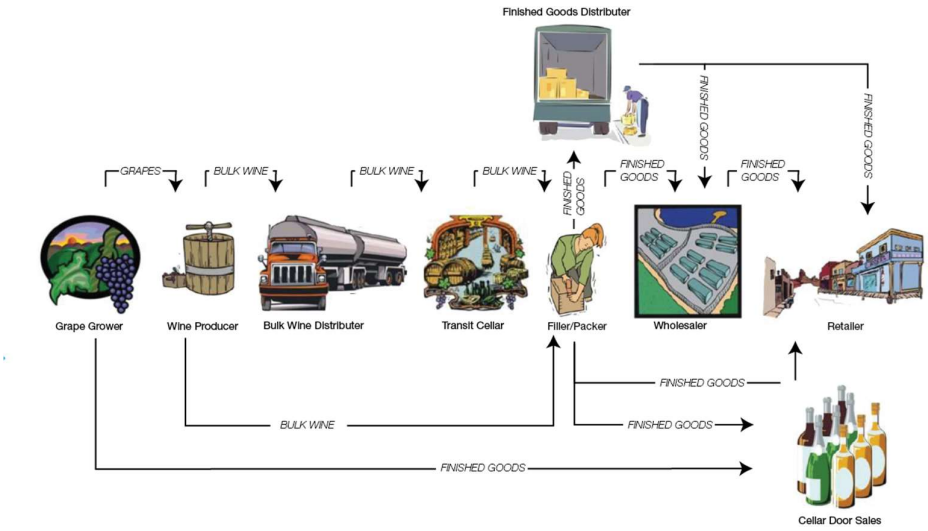

Figure 5: Wine Supply Chain (WSC) Actors
(based on: Armstrong et al., 2008)

The WTG determined several actors within the supply chain (Figure 5). At first glance there are many different steps and actors, however they can be summarized into the following categories, matching the results of the Fraunhofer paper (Schütte et al., 2018, p. 13):

Table 2: WSC actors categorized

Supplier	Distribution	Processing	Sales
Grape Grower	Bulk Wine Distributer	Wine Producer	Cellar Door Sales
	Finished Goods Distributer	Transit Cellar	Wholesaler
		Filler/Packer	Retailer

With these categories, the WSC can be compressed into fewer actors (Figure 7), to help to stay within the scale of a proof of concept implementation.

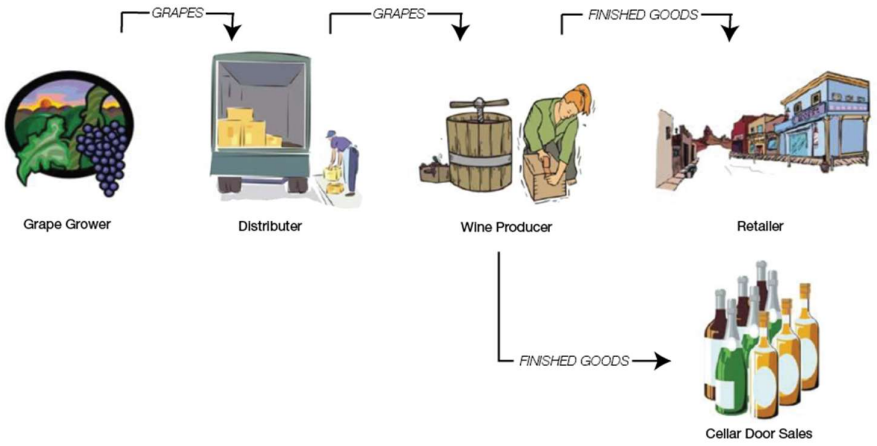

Figure 6: WSC actors compressed
(based on: Armstrong et al., 2008)

6.2.1 Grape Grower

The Grape Grower (GG) is responsible for the cultivation of one or several vineyards, using pesticides, fertilizer, as well as fulfilling regulatory paperwork.

6.2.2 Distribution

A distributor receives goods and transports them to their next destination. This can be a logistics company transporting grapes from a vineyard to a storage facility, a processing plant or delivering the finished wine bottles to retail stores.

6.2.3 Wine Producer & Filler/Packer

In the simplified supply chain the production and filling as well as packaging is combined into one *"processing"* step. The grapes are pressed, fermented and turned into wine. Afterwards the wine is filled into bottles, labelled, packed and shipped to retail via a distributer.

6.2.4 Retail

Retail is responsible for selling the product in supermarkets, outlets or directly to customers. This part of the supply chain does not generally create additional data that needs to be attached to a product. While the date of sale could be of potential interest for some products like electronics to have a proof of purchase for warranty, it is not as important for the purchase of wine. Therefore the retail section will not be included in the prototype, but may be an area for future work.

6.3 Requirements

Every step in the SC has some different requirements to reach transparency and being traceable.

6.3.1 Grape Grower

The GS1 guidelines (Armstrong et al., 2008) call for the tracking of following data-points:

- Name and address of the vineyard
- Plot map reference/cadastral reference/block identifier
- Size of plot/number of vines
- Vine variety
- Contact details

At harvest time, the GG is also required to keep track of additional data such as the number of harvested grapes, the date of the harvests and its including vine-yards. GS1 uses its Global Location Number (GLN) to include this data. With an uprooting or change of variety this number must get replaced.

6.3.2 Distributer

A distributer is responsible for receiving and transporting either supplies or fin-ished goods from one location to another.

- Tracing of shipments
- Samples / Quality check
- Possible invoices

6.3.3 Wine Producer & Filler/Packer

During the processing of the grapes and production of wine several data points have to be tracked. The producer is also responsible for:

- Data such as information on water used, chemicals for cleaning etc.
- Filling into bottles / barrels
- Lot number during filling process

With the individual actors determined, a first concept was setup how the entire prototypical supply chain would look like (Figure 7). This concept was also the basis for the final design of the exhibition art, painted by an art student. (Figure 29,p.61)

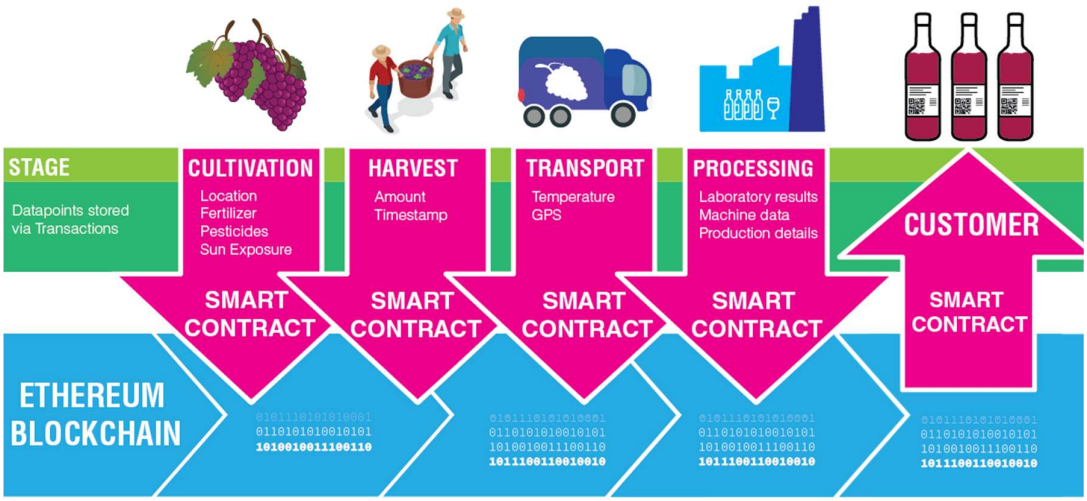

Figure 7: Proposed Supply-Chain representation

7 Short history of blockchains

Before going into depth with the technological details, I want to give a very quick overview over the history of blockchains, also showing how young the area of research still is.

The idea of having a digital currency has existed for many years. In 2005 Nick Szabo proposed *"Bit Gold"* in a blogpost. (Szabo, 2005) While never actually having been implemented, it is considered the predecessor to Bitcoin.

Bitcoin

Bitcoin was the first realization of a blockchain with a "Proof of Work" consensus algorithm and created the first peer to peer currency. Under a pseudonym, a paper titled "Bitcoin: A peer-to-peer electronic cash system" was released to the public on 31 October 2008 (Nakamoto, 2008). In January of the following year the first "block" was mined and the Bitcoin-Chain officially started to exist.

The Bitcoin blockchain does not offer any way to run functions "on-chain" by default.

Ethereum

Vitalik Buterin proposed Ethereum in 2013 in order to add an application layer to the blockchain. (Ethereum Foundation, n.d.) With these, so called smart contracts, it should not only be possible to have a currency but also any variety of digital assets. Ethereum has a built in, touring complete, language to write state transition functions.

Hyperledger

The Linux foundation started the Hyperledger project in 2015 (The Linux Foundation, n.d.) and consists of several individual frameworks. The most commonly associated one being *"Hyperledger Fabric"* , a permissioned blockchain network contributed by IBM. (Ferris Christopher & Blummer Tamas, 2016; "projects:fabric [Hyperledger Wiki]," n.d.). The goal is to have a modular platform for enterprises to build their projects on.

Like Ethereum, on Fabric self-executing logic can be ca deployed. Instead of calling them *"smart contracts"* those code pieces are called *"chaincode"* ("Chaincode — hyperledger-fabricdocs master documentation," n.d.). These snippets can also be written in modern languages, for instance Go or node.js, without having to rely on new languages such as Solidity.

8 Existing Block-/ Supply-Chain Solutions

The online platform "State of the ĐApps" lists projects and their Apps, most of them running on the Ethereum blockchain[5]. The tags "supply-chain" and "supply chain" reveal a total of three projects, of which two have the status "*'live*".[6] Further online research reveals several more companies or startups trying to bring the SC to the blockchain. In this next section I want to have a closer look at some

[5] https://www.stateofthedapps.com/
[6] Provenance & Devoleum

of the existing Đapps as well as the most prominent Startups in order to see if any of these projects offer a basis to build on or if they elaborate their strategies.

8.1 Devoleum[7]

***Description**:*

Devoleum is a Project that was created as part of the French startup contest *"StartHer Awards"* in October 2017. It aims to make the supply chain of Italian Olive oil traceable with an App and QR codes.

***Product**:*

There is a web interface[8], that retrieves data from the *Rinkeby* testnet (Ethereum). It displays a timeline of different "products", when the olives were harvested, transported, milled and stored.

***Conclusion**:*

The project does not reveal any technical details in a whitepaper or on their web-site. While there is a smart contract - or multiple ones - deployed to the *Rinkeby* net, the source code is not available. The "simulation" only retrieves some timestamps from the chain, how those were created is not explained. In general, the information about the product is very scarce and presumably, due to it being project from within a startup contest, may not be available. There has not been an update or news about the product since late 2017.

8.2 Provenance[9]

***Description**:*

The British tech-startup Provenance has created a platform to enable businesses to trace the origins of their products and connect "stories" to them.

***Product**:*

Several case studies are made available on their website[10], showing off an App that displays a timeline of a product.(Figure 8) The connection of physical goods

[7] https://www.devoleum.com/
[8] https://simulation.devoleum.com/
[9] https://www.provenance.org/
[10] https://www.provenance.org/case-studies

to the digital is again done via QR-Codes. Initially the date is not visible, can only be accessed via request.

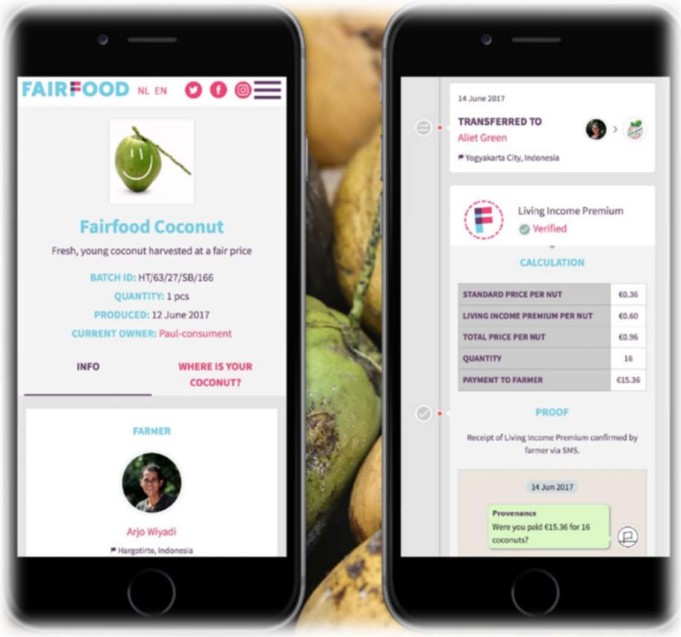

Figure 8: Provenance App[11]

Conclusion:

While having a whitepaper, outlining their mission, the paper does not include any technical information for any of the shown case studies.

8.3 Walimai[12]

Description:

Walimai is a Chinese startup trying to fight counterfeits in food and other consumer goods in china.

Product:

At the point of origin, a NFC label with QR Code that is linked to a blockchain is put on the product.(WaBi, 2017) If this label is tempered with, the scan fails to verify. The startup also claims that copying the data from one label to another

[11] https://assets.provenance.org/web-assets/modules/provenance-case-study-fairfood-1080x720- 8d9b029e04924d711c25f3416dee25c31ee04894fa02495914c9624de90674cd .jpg
[12] https://www.walimai.com/en/

would cause the entire batch to be "offsync" and the consumer would get warned, that he product is not safe anymore.

***Conclusion**:*

The whitepaper and websites are very limited on actual technical information. The Walimai system runs on a version of *Hyperledger* (Walimai, 2018), but at the time of this paper it is still running in a test environment. No information on how the labels are secured by the blockchain or how the data gets verified could be found. However, this might be due to the startup being from China and not releasing the information in English, something that cannot be verified at this moment. It also only seems to aim to verify products from manufacturer to the store, not the individual parts and contents.

8.4 Modum[13]

***Description**:*

Modum started in 2016 in Zürich, Switzerland and combines Internet of Things (IoT) with the blockchain technology. They aim to ensure that pharmaceuticals have been transported under EU regulations.(Modum, 2018b)

[13] https://modum.io/

***Product**:*

The company built a small IoT device that can be put in parcels and tracks temperature data during the transport. After setting certain alarm criteria, a smart contract for the shipment is setup, including that information as well as other transport specific data. During the transport the data gets logged and stored in a database. Upon receival of the parcel, the device can be scanned and the data then either gets verified or declined by the smart contract.

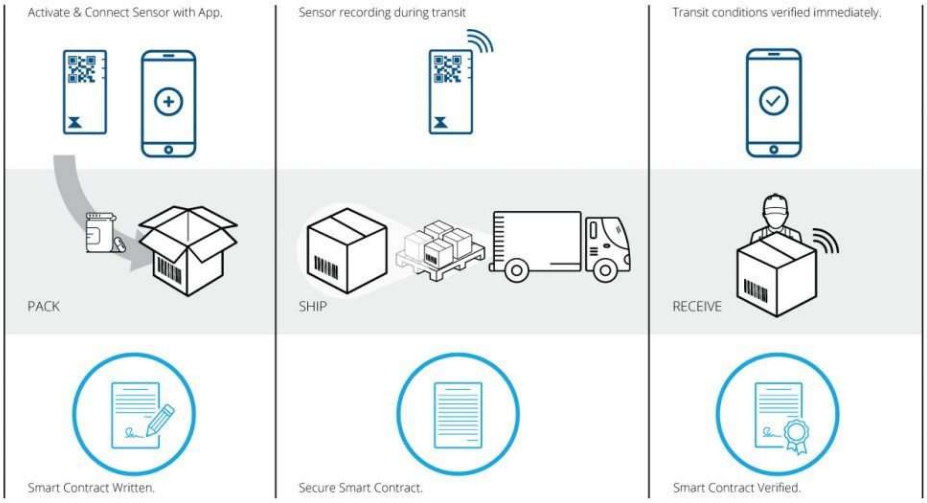

Figure 9: Modum: How it works (Modum.io, n.d.)

***Conclusion**:*

The actual temperature data is stored on the logger and/or a database, not the blockchain itself. However the data is encrypted and the device held to high security standards. (Modum, 2018b). The smart contracts are only used to verify an array of temperatures from a centralized database.

Like Walimai, it only aims to track the *"last mile"* and verify very specific data.

8.5 WaltonChain[14]

***Description**:*

WaltonChain, a Chinese and Korean Startup, also combines IoT with the blockchain technology and calls it VIoT, Value Internet of Things.(Waltonchain Team, 2018b)

***Product**:*

As the name suggests, WaltonChain offers their own blockchain solution aimed for enterprises to track their products with RFID. The underlaying technology is based on Ethereum. (Waltonchain_EN, 2018)

Conclusion:

Their main net launched in March 2018 (Waltonchain Team, 2018a) but no development tools, like an SDK is available. While everyone can buy and mine the Tokens required to interact with their network, no actual working use cases exist to use them. Again, this could be due to the language gap and lack of translation to English.

8.6 OriginTrail[15]

Unlike the previous named projects, OriginTrail wants to be blockchain agnostic and offers a data and network layer for businesses.

***Product**:*

OriginTrail only stores the fingerprint/hashes of product batches in a blockchain and thus offers some privacy. With their own network layer, they offer a *"zero-knowledge"* validation of Data.(Branimir Rakic MSc, Tomaz Levak, Ziga Drev, Sava Savic PhD(c)., & Aleksandar Veljkovic PhD, 2018)

Conclusion:

While claiming to be blockchain agnostic, the Trace Token is required to interact and make transactions. This token is an `ERC20` Token and thus Ethereum based. Various use cases are shown, for example authenticity of wine (OriginTrail, 2018b) and traceability of poultry(OriginTrail, 2018a). However, at least in their

[14] https://www.waltonchain.org/
[15] https://origintrail.io/

current implementation, the consumer has to enter the batch number and date in an app, together with scanning the barcode. Then relevant data of the product is shown.

8.7 VeChain[16]

Description:

VeChain is another Chinese startup that combines NFC, RFID chips, QR codes and proprietary IoT devices with blockchain technology. As such it seems the culmination of all previously mentioned technologies.

Product:

VeChain started out as ERC20 token but has recently made the switch to their own Proof of Authority network.(VeChain Foundation, 2018b)

Conclusion:

Currently there is no whitepaper available, only a document outlining their general development idea. (VeChain Foundation, 2018d) While various showcases and alliances are presented, no actual product can be found online. VeChain recently co-founded the *"Shanghai Wine and Liquor Blockchain Alliance"* (VeChain Foundation, 2018a). Their goal is to ensure authenticity and traceability in this sector.

Figure 10: MyStory QR-Code

[16] https://www.vechain.com

8.8 Specific solutions for the Wine industry

8.8.1 MyStory

In cooperation with DNV GL and some Italian wine producers, VeChain launched *"MyStory"* in early 2018.(DNV GL, 2018; VeChain Foundation, 2018c; Vestvik-Lunde, n.d.). But besides these articles, there is very little information about the actual product and how it handles the data. Also, the Application itself cannot be found. (Figure 10, VeChain Foundation, 2018c)

8.9 Ez Lab Wine Blockchain[17]

Ez Lab, an Italian Startup launched a project together with ERNST&YOUNG (Ez Lab, n.d.) to trace the wine supply chain from cultivation to bottling. In a case study a mobile application is shown, that scans a QR code on a bottle and retrieves all the information linked to this product. As with most other projects, no technical information is available on their website, the only details were found in a blog post not related to the company (Tomasicchio, 2017), claiming it uses Ethereum and smart contracts. This claim could not be verified.

8.10 Everledger & Chai Wine Vault[18]

Based on Hyperledger, the company Everledger created a solution to ensure wine authenticity in corporation with the Chai Wine Vault (Everledger, 2016). They use a set of 90 datapoints extracted with *The Chai Method* together with high resolution photography and create a digital "fingerprint" of a wine bottle. This "fingerprint" is then stored on a blockchain.

Originally announced in 2016, there has not been any news since then. The product also only serves as a form of database, but which technology in particular is used, is not explained.

[17] https://www.ezlab.it/case-studies/wine-blockchain/
[18] https://www.winefraud.com/chai-wine-vault/

8.11 Scientific research

Blockchain technology and smart contracts could disrupt many infrastructures and as such has been increasingly become topics of research.

Most studies however only cover a single aspect of the technology, such as the economic impact, security concerns, consensus and legal aspects of the technology. The technology should however be considered a multi discipline area of research, as it affects more than a single sector in industries, for example information technology and law (Mielke & Wolff, 2017), creating entirely new areas of research.

The possibility to disrupt the financial sector by bypassing conventional money transactions with cryptocurrencies, has been discussed since Bitcoin launched in 2008. (Accenture, 2017; Chiu & Koeppl, 2017)

Several studies have explored the cryptographic algorithms used for consensus (Bano et al., 2017; Bentov, Lee, Mizrahi, & Rosenfeld, 2014; Gramoli, 2017; Porat, Pratap, Shah, & Adkar, 2017) as well as the security of smart contracts. (Kalra, Goel, Dhawan, & Sharma, 2018; Luu, Chu, Olickel, Saxena, & Hobor, 2016)

Smart contracts have also recently sparked interest in legal departments and several papers have evaluated whether or not a "smart contract" is an actual contract. (Bacon, Brook, & Bazinas, 2016; Hansen, Rosini, & Reyes, 2018; Mielke & Wolff, 2017)

Research on blockchain technology for supply chains is still limited but a few are available. Extending on his previous work from 2016, O'Leary analyses the architecture and came to the conclusion that the technology mirrors basically all aspects of the transactional processes of supply chains, but existing SCM systems need to be "re-engineered" (O'Leary, 2017). A first ontology based model together with some basic smart contracts was proposed in 2016 (Kim & Laskowski, 2016). A single Ethereum smart contract handles objects[19] - called *TRU* - that can be created, used and consumed by processes. While this concept enables some simple traceability, no additional data about the product is added. Knowing the

[19] A struct object

limitations of Ethereum it is difficult to create complex `Struct` objects and Solidity makes it impossible to return these objects from a function[20], severely limiting the capabilities.

Another more recent paper explored the possibility of tracing wine with blockchain transactions, in particular the hash functionality. (Biswas, Muthukkumarasamy, & Tan, 2017) The underlaying blockchain is based on Multichain[21], which does not support smart contracts. Without the additional logic, only the transactions are stored, products or different objects in the SC have to be identified by other means.

8.12 Research Conclusions

While there are many companies and startups involved in bringing the supply chain to life with the blockchain technology, very little information of how it is done can be found. Only smaller parts for very specific cases have been successfully launched or are verifiable, such as Modum. Platforms such as VeChain and OriginTrail want to reach enterprises but don't offer development solutions to the public.

Therefore, it is not possible to build a Proof of Concept with any of those afore mentioned technologies. Some scientific research has been done previously on the topic of combining SC with blockchain technology, however not with complex smart contracts that enable to possibility to attach various kind of data to the product throughout the SC. For this reason, new patterns have to be developed and created on a platform. . Ethereum offers a wide range of development tools and is very well documented. Also, many existing projects started out on the Ethereum platform, thus making it the logical choice for this concept.

[20] See Section 9.5.3: Structs, p.36
[21] https://www.multichain.com/

9 The Ethereum Platform

Ethereum offers a wide variety of developer tools and has a very detailed documentation on all its interacting components.

The prototype will consist of several smart contracts, which are written in Solidity for Ethereum. To interact with them from the browser, a web library is required. Ethereum offers the Web3.js library that is already very stable and well adopted in the scene. MetaMask[22] offers a plugin for Firefox and Chrome to interact with Ethereum blockchains and helped the web application *Cryptokitties*[23], going viral, causing a severe network congestion (ConsenSys, 2018a). [24]

The next sections will highlight the functionality and structure of smart contracts on the Ethereum Blockchain as well as the languages and interaction methods available.

9.1 Smart Contracts

A blockchain is a ledger of transactions. While blockchains like Bitcoin or Litecoin are only storing currency transactions, blockchains of the second generation made it possible to add a layer of code in between. Ethereum's founder Vitalik Buterin chose the name "Smart Contracts" for the bits of code. This term was first mentioned by Nick Szabo in 1996, describing a digital contract in the form of code that only runs if certain conditions are met and are cryptographically secured (Szabo, 1996). In a later post, he outlined the concept with an example of a digital car lease (Szabo, 1997), which I will rephrase in the following part.

Before a transaction is added to the chain, the transaction is checked by pre-defined functions and requirements. This enables the potential for a lot of automatization. A carsharing company could put a smart contract in place, that only opens the car's door if a user has a) a valid driver's license, b) enough funds to pay for the service and c) enough funds to pay the deposit. There is no need for

[22] https://www.metamask.io
[23] https://www.cryptokitties.co
[24] CryptoKitties is an online collectible game, with underlying smart contracts, enabling players to collect, breed and sell their collected cats, each being unique.

an office to manually check the requirements, no bank involved, everything it is automated.

In its simplest form, smart contracts are nothing more than an `if… else…` check. So why is this regarded a new technology? Staying with the car example, the company has to trust, that the requirements cannot be altered in any way and that the code always runs, so that no person that shouldn't be able to get into the car, does. On the customer side, the user has to trust, that once he returned the car, he gets the deposit back. These rules are set in a smart contract, that can be publicly accessed and audited. Due to it being stored inside a block chain, these rules cannot be changed, both parties can trust the contract.

9.2 The EVM

Every account on the Ethereum blockchain consists of four parts:

- Nonce: a transaction counter, preventing executions of identical transactions multiple times
- Storage: key/value storage where both are 32byte long strings
- Bytecode:
- Balance: The amount of Ether the account owns

There are two types of accounts, externally owned accounts (EOA) and contracts[25]. EOAs have a unique private key, with it the account can be accessed and re-created from every node. Their storage and bytecode is always empty.

Contracts store their logic inside the bytecode area and a hash of the entire patricia-merkle state tree (Wood, 2014).

Every full node in the network runs an interpreter for the assembly-like bytecode language. This interpreter is called the Ethereum Virtual Machine, EVM in short. It is a quasi-Turing-complete machine (Wood, 2014, p. 10). *Quasi* only, because the computation is limited by the parameter *gas*. Every operation has a fixed *gas* value, listed in the Ethereum yellow paper Appendix G. The bytecode of a contract is a list of these op-codes and looks for example like this:

```
PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH1 0x3F […]
```

[25] https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial

9.3 Application Binary Interface

In order to be able to interact with the bytecode with other languages, like JavaScript, an additional interface is needed. One standard for this is the Application Binary Interface (ABI). The detailed specifications can be found in the official documentation[26] but an example output of a function in `JSON` format looks like this:

```
[{
        "constant": true,
        "inputs": [],               // no input
        "name": "getBalance",       // the function name
        "outputs": [{               // returns
            "name": "",
            "type": "uint256"
        }],
        "payable": false,           // can receive ether?
        "stateMutability": "view",  // does it change the state?
        "type": "function"
    },
[…]
```

Code 1: ABI example

This file, combined with the contracts address can then be used together to call available functions:

```
var contract = new eth.Contract([ABI], address, […]);
console.log(contract.getBalance.call());
```

Code 2: Using the ABI & address to call a contract in JS

9.4 Programming languages

It is in theory possible to write smart contracts with any language, as long as it can get compiled to the Ethereum bytecode. However a few languages are available, that have been specifically designed for smart contracts. In this section I want to give a quick overview over some of them.

9.4.1 Solidity

Solidity is the most commonly used language to write smart contracts for the Ethereum blockchain. Solidity code can be deployed to an Ethereum blockchain with Remix IDE, the official Ether Wallet as well as the Mist Browser, which is an extension of the wallet. Solidity is the official standard and has a very detailed documentation.[27]

[26] https://solidity.readthedocs.io/en/develop/abi-spec.html
[27] https://solidity.readthedocs.io/en/v0.4.24/index.html

The code running on the EVM however is bytecode and so any language has to get compiled to this bytecode before deploying. For Solidity, the *Solc* compiler[28] is used.

Besides the official documentation, big libraries, such as *OpenZeppelin*[29] and tutorials like *CryptoZombies*[30] are available making it the best choice to start developing smart contracts for Ethereum.

9.4.2 Lisp Like Language[31]

Lisp Like Language (LLL) is described as *"a low level language similar to Assembly. It is meant to be very simple and minimalistic; essentially just a tiny wrapper over coding in EVM directly."* [32] It is one of the original languages released to be compiled to the ETH bytecode. With LLL it is easier to keep track of the *costs* of a contract as the EVM op-codes can be used directly in LLL. The documentation is very limited and not many contracts have been written in LLL so far. If unfamiliar to Lisp or Assembler it is difficult to get started with LLL.

9.4.3 Vyper

Vyper is an experimental language still under development by the Ethereum developers.[33] It strips away a lot of functionality that Solidity offers, such as modifiers, class inheritance or overloading.[34] The reasons are listed as making the code more auditable and more secure. Vyper's syntax closely resembles Python code and the same is required in order to compile it.

It is not entirely clear if Vyper replaces Solidity at some point, some contracts of a future platform Update (Casper) are writing in Vyper[35] but currently it is still heavily under development.

---

[28] http://solidity.readthedocs.io/en/v0.4.24/using-the-compiler.html#
[29] https://openzeppelin.org/
[30] https://cryptozombies.io/
[31] https://lll-docs.readthedocs.io/en/latest/lll_introduction.html
[32] http://www.ethdocs.org/en/latest/contracts-and-transactions/contracts.html#id4
[33] https://github.com/ethereum/vyper
[34] https://vyper.readthedocs.io/en/latest/index.html
[35] https://github.com/ethereum/casper/blob/master/README.md

9.4.4 Other languages

There are a few more languages, either deprecated like Mutan[36] or not recommended to use, like Serpent[37]. Others, like Bamboo[38] are still in very early stages and still lack in support and documentation.

9.5 Solidity

As described in the previous section, Solidity[39] is simply the best choice currently to write smart contracts for Ethereum. In this section I want to highlight a few things that will be used in the implementation and are unique to this language.

9.5.1 Contracts

A contract's structure is like a `class` in OOP-like languages. A very simple example could look like this:

```
1.  pragma solidity ^ 0.4.23;
2.
3.  contract Calc {
4.
5.      function add(uint _a, uint _b) public pure returns(uint) {
6.          return _a + _b;
7.      }
8.
9.  }
```

Code 3: Basic Contract

The deployment in *Mist* costs a fee – called *gas* – depending on the complexity of the contract The exact amount is determined by the EVM op-codes (Wood, 2014). After deployment to the blockchain, the contract receives a unique 32bit address pointing to it, for example:

`0xb50ba468ea264db6255230bc7bb605af55a37256`

[36] https://github.com/zsfelfoldi/go-ethereum/wiki/Mutan-0.2/
[37] https://github.com/ethereum/serpent/
[38] https://github.com/pirapira/bamboo/
[39] Current version at time of writing: 0.4.24

With this address, everyone inside the same blockchain network can use the functions from this contract by calling the contract with its address:

```
1.  import “../calc.sol”
2.
3.  contract B {
4.      function calculate(address a) public view returns(uint) {
5.          return Calculator(a).add(5, 10);
6.      }
7.  }
```

Code 4: Calling another contract

One drawback is, that Contract B needs to know the sourcecode of the first contract in order to be able to access it. This can be done by using `imports.` (see line 1) It is also possible to store data inside contracts:

```
1.  contract Store {
2.      string s;
3.
4.      function setString(string _s) public {
5.          s = _s;
6.      }
7.
8.      function getString() public view returns(string) {
9.          return s;
10.     }
11. }
```

Code 5: Setter and Getter functions

Calling `getString()` is free, as it does not change the state of the contract and therefore the chain. To update the variable to a new value, a transaction is required (Figure 12) and since transactions are automatically recorded, so making changes can always be traced back.

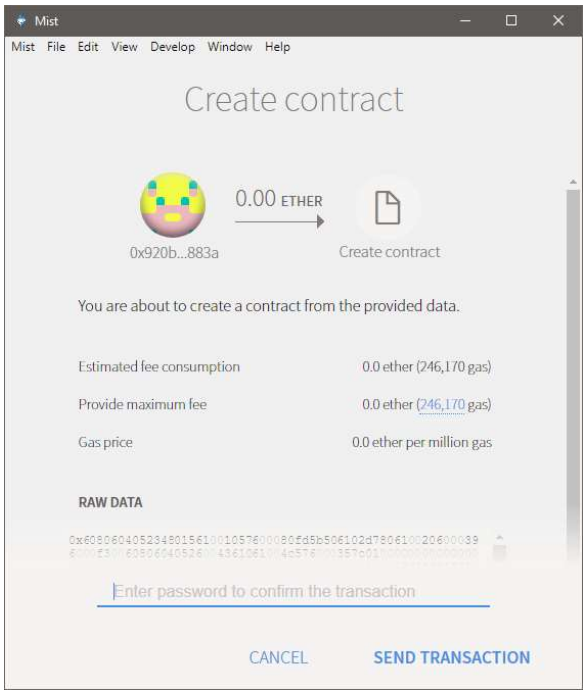

Figure 11: Deploying a contract

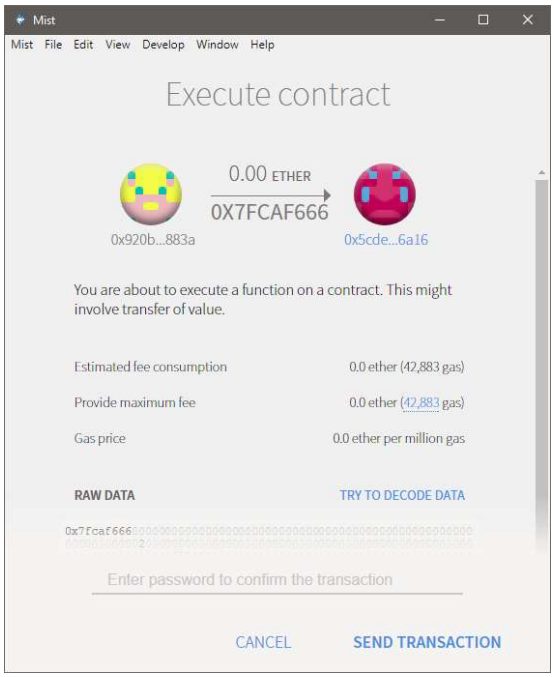

Figure 12: Sending a transaction

9.5.2 Mappings

Mappings are a unique feature of Solidity. While being somewhat similar to a hash table it doesn't store any actual data from the keys, only the keccak hash of it. (Ethereum Foundation, 2018b). But at first initialization, it already holds every possible key and maps it to the default value of solidity `0x`. This is a very useful key-value storage. One example, used in the *ERC20* standard: ("ConsenSys/Tokens,",2018; Fabian Vogelsteller & Vitalik Buterin, 2015)

```
mapping (address => uint256) public balances;
```

This maps an address to an integer value. The mapping is public, therefore solidity automatically creates a getter function, although one can be implemented manually as well.

```
balances[0x627306090abaB3A6e1400e9345bC60c78a8BEf57];
```

The request above will return the value that this address is mapped to, a concept that is used in tokens to keep track of how many tokens an account holds. As every possible address is already mapped, there is no need to add addresses manually. Any account can easily be updated by mapping it to a value:

```
balances[0x627306090abaB3A6e1400e9345bC60c78a8BEf57] = 100;
```

9.5.3 Structs

Structs are a form of Objects within Solidity. A *Struct* can consist of several types such as variables, mappings and other structs.

```
1.  contract UserContract {
2.      struct User {
3.          address userAddress;
4.          string userName;
5.          uint userAge;
6.      }
7.      User[] public users;

8.      function newUser(address _add, string _name, uint _age) public {
9.          User memory u = User(_add, _name, _age);
10.         users.push(u);
11.     }
12. }
```

Code 6: Struct example

This very basic setup can create a new User with some variables and stores it inside an array. In Remix you can test the interaction by accessing our array directly. (Figure 13)

Figure 13: Struct example in Remix

Despite it showing the correct output when accessing the array in Remix, a struct object cannot be returned from a function, which is still a big limitation of the language.

Structs in combination with mappings can be a very useful way to store a lot of information and map it to specific key, something I will later use to handle the storing of transaction information within contracts.

9.5.4 Modifier

Solidity Functions can be extended with modifiers, that check on runtime if a requirement is fulfilled or not. This makes the integration of certain checks very modular and easily modifiable. (Ethereum Community, 2018b)

```
contract ModExample {
    address internal owner;
    constructor() {
        owner = msg.sender;
    }
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function doSomething() public onlyOwner {
    // only the owner of the contract can access this function
    }
}
```

Code 7: Modifier example

The above example shows a very common pattern of restricting certain functions to a specific address, in this case the `owner` of this contract which is automatically set when the contract gets deployed and the `constructor` fires.

The modifier is added after the visibility of the function and is in itself a function and as such can also take variables. Before the main function gets executed, the modifier function runs. In the example it verifies, that the address that called the function is the owner of the contract with the `require()` check integrated in Solidity. If it fails it reverts the transaction and refunds it.(Ethereum Foundation, 2018a). After a successful check it jumps to `_;` a placeholder for the function that got called originally to continue. With this placeholder it is also possible for modifiers to execute code after the function ran, for example to automatically move the contract to the next stage in a state-transition machine like setup. (Ethereum Whitepaper, 2018)

9.6 ĐApps

Applications using the Ethereum network, are usually referred to as Decentralized Apps, ĐApps in short. The **Đ** is the Greek letter *Eth* and is used to show that it is an *Eth*ereum Application.

The website *State of the Dapps* (State of the Dapps, 2018) currently lists over 1600 of these, some of which are highly successful, like Cryptokitties (Cryptokitties, 2018).

9.7 Clients

To interact with a DApp, a way to connect to the blockchain itself and an account is required. The app can either offer a direct connection to a node via the RPC Interface[40] or use an injected Web3 provider through a browser plugin such as MetaMask[41] or a browser with Web3 support. For the desktop only the Mist Browser[42] is available as standalone browser (Ethereum Community, 2018a), for mobile devices Toshi[43] or Cipher[44] can be used. MetaMask is the easiest way and available as plugin for most modern browsers.

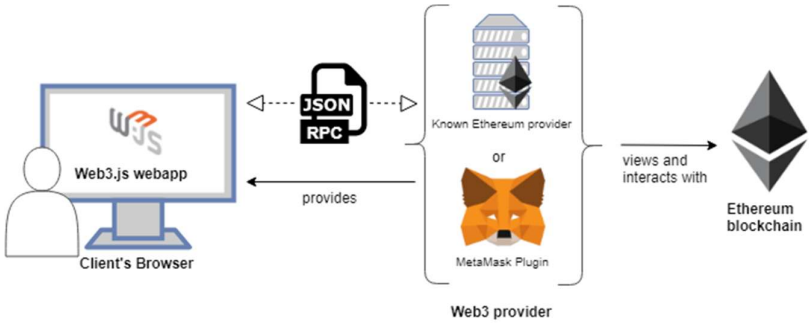

Figure 14: Web3 Interface (Corte, 2018)

With a provider available, requests and transactions can be done by using the web3 JavaScript API (Ethereum Community, 2018d) The current stable version is 0.20.6 but the 1.0 release is nearing its final beta steps and will be used for this proof of concept. The biggest change is the switch to asynchronous callbacks, making it easier and faster to interact with the blockchain from the web.

[40] https://github.com/ethereum/wiki/wiki/JSON-RPC/
[41] https://www.metamask.io/
[42] https://github.com/ethereum/mist/
[43] https://www.toshi.org/
[44] https://www.cipherbrowser.com/

10 Implementation

The implementation consists of two parts:

- The smart contracts as back-end
- A web-based User Interface

While the smart contracts can be extended to a real life test, the UI mainly focuses on testing and simulating the interactions within the supply chain.
The UI can be tested locally after meeting the requirements and installing according to the ReadMe-File[45].

10.1 Smart Contracts

In this section I will explain the structure of the smart contracts and how the data can be collected as well as retrieved. The full documentation is available online[46] and on the DVD[47].

10.1.1 Tools

The easiest way to develop smart contracts in Solidity is with *Mist*[48] (Wood, 2014) as it includes the *Remix IDE*[49] (ConsenSys, 2018b) as well as a way to observe contracts and handle multiple accounts. Mist can be connected to a "live" blockchain by starting it with the corresponding RPC[50] address of a node or used without connecting it to one, thus creating a local development environment. The latter has been used for the majority of the development processes, as it also makes it possible to have accounts without passwords, speeding up the deployment and testing considerably.

[45] ReadMe: "./Code/README.md"
[46] https://korbiniank.github.io/SupplyChain/
[47] Folder: "./Code /docs"
[48] Version 0.10.0 was used
[49] https://remix.ethereum.org
[50] https://github.com/ethereum/wiki/wiki/JSON-RPC/

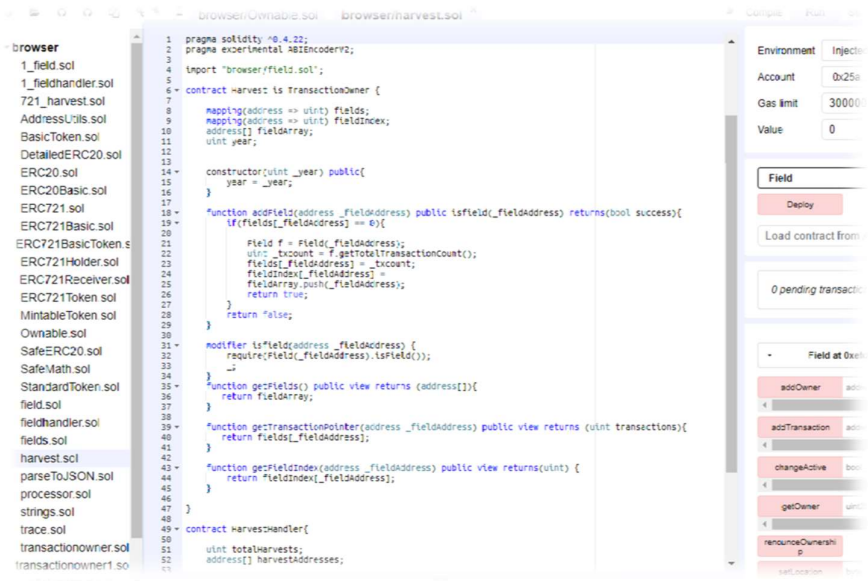

Figure 15: Missing folder support in Remix IDE

Remix handles the debugging, compiling and deploying and creates an effortless way to make changes to contracts. It however also has some drawbacks. It lacks a way of exporting the individual `.sol` files as well as a support for folders. Having multiple files to work with, it's getting increasingly difficult to keep them organized (Figure 15).

10.1.2 Requirements

In addition to the requirements in Section 6.3[51] the smart contracts also have to fulfill a few more, specific to this use-case:

- Each step of our supply chain needs to be able to receive and store transactions that include data
- Every Interaction, that changes the state should be stored inside the contracts
- Each step also needs to include some specific attributes unique to these steps
- Each step needs to be linked with the previous one, so that a traversable path exists from start to end
- The entire dataset must be query-able

[51] See p. 27

10.1.3 Transactions

In the supply chain, there are several individual *steps*, such as a Vineyards, Transports and Bottles which will have data attached to them. This could be done by representing each step as a Struct (9.5.3) inside a single contract

but every step of the supply chain needs to have its own attributes and details. Storing them together with the specific logic of that step, makes the contract(s) bigger than they need to be. It makes sense to keep the individual parts, such as a vineyard, a transport or a harvest separate from the others and only link them with transactions and addresses. If a new instance of one of these steps is created, they all have their unique address and not just a pointer inside an array. This makes it easier to associate data to a specific step.

Also if Tokens are used and should get transferred from one *step* to another, the implementation gets difficult. The ERC20 token, for example, keeps track of addresses and their balance in a mapping. (1) Transfers happen between separate accounts, with `transfer(address _from, address _to)`. It would create a lot of overhead, if every struct needs their own custom balance mapping and a lot of adjustments to the token interface. It would defeat the purpose of the standard.

The transactions alone could be done by the default transaction functionality included in Ethereum and Solidity. A default transaction contains `msg.sender`, the address of the account that made this transaction, `msg.value`, the amount of ether that has been sent within the transaction and finally, `msg.data`. The data part of a transaction can be *"an unlimited size byte array specifying the input data of the message call"* (Guideline, Beckett, Klein, & Corbet-milward, n.d.). It is then possible to access a transaction by querying the blockchain with the transaction hash and retrieve the data that was sent with it.

However, it is impossible to get those transaction hashes from within the contract, as they are generated outside the contract when the transaction is *'mined'* and processed. At the time of creating the transaction, this data is simply not available. For the same reason, a contract itself cannot give any information which transactions happened, at least not by default.

Due to this limitation by design, transactions must be stored manually. For this I created a contract that can be then inherited by other contracts within the system: the contract `TransactionOwner`.

The TransactionOwner Contract

The contract contains two `Structs`[52] which will store the details of the transactions:

```
1.  struct TransactionSender {
2.      address sender;
3.      mapping(uint => TransactionStruct) transactions;
4.      uint transactionCount;
5.  }
6.
7.  struct TransactionStruct {
8.      address sender;
9.      bytes data;
10.     uint256 time;
11. }
```

Code 8: Storing Transaction details in structs

The `TransactionSender` object describes the sender of a Transaction, including the address of the sender, as well as a map of `TransactionStruct` and a counter of how many transactions this particular sender has sent to the contract.

The address is mapped to this object, so you can get more detailed information about this sender with this mapping:

```
mapping (address => TransactionSender) public transactionSender;
```

The `TransactionStruct` describes the individual Transactions itself and contains the sender as well as the data sent with it. For this first concept, only one *data* input is implemented, but this pattern opens the possibility to store specific variables separately by simply adding it to the struct. A transaction does not return a timestamp itself (s. Figure 16). In order to get the timestamp, one would have to retrieve the block number and then query the block with that number to get information when this block was mined.

[52] See 9.5.3: Structs, p.17

```
  blockHash: "0xe62178e407c16bc5d27fbe0ebe48e5b0eac5ef8e823cc83d9e805f0da6679f80"
  blockNumber: 13
  contractAddress: null
  cumulativeGasUsed: 158252
  gasUsed: 158252
▶ logs: []
  status: 1
  transactionHash: "0x61933df45af8afcd213a4af0841fd30874f9d862ac9a7496a7607c142213d35f"
  transactionIndex: 0
▶ __proto__: Object
tx: "0x61933df45af8afcd213a4af0841fd30874f9d862ac9a7496a7607c142213d35f"
```

Figure 16: Callback of a transaction

Inside the block is the information about which transactions have been added to this block and the timestamp of when the entire block was mined. While possible to determine the timestamp this way, it requires multiple calls to the network, to simplify this process, the variable `uint256 time` will be set at creation time with the timestamp. Solidity offers this functionality by adding the block timestamp directly (Ethereum Community, 2018).

```
  difficulty: "0"
  extraData: "0x0"
  gasLimit: 6721975
  gasUsed: 158252
  hash: "0xe62178e407c16bc5d27fbe0ebe48e5b0eac5ef8e823cc83d9e805f0da6679f80"
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000
  miner: "0x0000000000000000000000000000000000000000"
  nonce: "0x0"
  number: 12
  parentHash: "0x0a27cb8a10d379e06310eb0c9d546e04a97b08270b1db513f76e862e09b24e20"
  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347"
  size: 1000
  stateRoot: "0xfccb98ef5aec555ba56e0054cde628d74c391903b2b0a4dbec575702308033f1"
  timestamp: 1529239606
  totalDifficulty: "0"
▶ transactions: ["0x61933df45af8afcd213a4af0841fd30874f9d862ac9a7496a7607c142213d35f"]
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421"
▶ uncles: []
```

Figure 17: Block content

To be able to access these transactions easily at a later point, a mapping to an index is created:

```
mapping (uint => TransactionStruct) public transactions;
```

Additionally, to keep track of the previous sender accounts, another mapping to a Boolean value is created. It is then possible to check, whether it is a new account or one that has sent data before.

```
mapping (address => bool) internal isSender;
```

The addition of transactions to the contract are then handled by the following function:

```
1.  function addTransaction(address _sender, bytes _data) public onlyVerified {
2.      if (!inArray(_sender)) {
3.          sender.push(_sender);
4.          isSender[_sender] = true;
5.      }
6.      TransactionSender storage ts = transactionSender[_sender];
7.      ts.transactions[ts.transactionCount] = Transaction-
    Struct(_sender, _data,now);
8.      transactions[totalTransactions] = TransactionStruct(_sender, _data);
9.      tsender.transactionCount++;
10.     totalTransactions++;
11. }
```

Code 9: Storing a Transaction

In order to be able to add a transaction, the sender needs to be a verified address and is checked by the modifier `onlyVerified`.[53]

The contract then checks, if the sender has previously sent a transaction to this contract. If it has not, that address is added to the array `sender[]`. To remember that this address has sent a transaction, the mapping of the address is set to *true*.

From the mapping of `transactionSender` you get the struct object `TransactionSender tsender`. Here the mapping functionality of Solidity shows its usefulness.

If the sender has not sent any Transactions before, you simply get an empty object back, that then can be filled. If the sender exists, the object with all the relevant information is sent back can then extended.

The transaction information is then added as `TransactionStruct` to the mapping within the object itself. Lastly, the transaction counter of the object is increased as well as the overall transaction counter of the contract instance.

[53] See Section 8.4.3: Modifier, p. 2

The remaining functions will only be explained in short, as most of them are straight forward:

```
function getTransactionDataAtIndex(uint _index) public view returns(bytes) {
    return transactions[_index].data;
}
```

Code 10: Getting data from a Transaction I

This function takes a pointer and returns the data from our transaction mapping. Again, the mapping is very useful as even when the index does not point to an existing transaction it will then simply return `0x` instead of throwing an error.

```
function getTotalTransactionCount() public view returns(uint) {
    return totalTransactions;
}
```

Code 11: Getting the number of total Transactions

The overall amount of transactions the contract has handled can be queried by:

```
function getTransactionSenderAtIndex(uint _index) public view returns(address) {
    return transactions[_index].sender;
}
```

Code 12: Getting a specific sender

With a pointer it is possible to get the sender from a transaction from the mapping, again if there is no sender, it will simply return the default *0x* value.

```
function getTransactionCountFromSender(address _sender) public view returns(uint) {
    TransactionSender storage s = transactionSender[_sender];
    return s.transactionCount;
}
```

Code 13: Getting the number of transactions sent by a specific address

To check how many transactions come from a particular address, the mapping `transactionSender` can be called:

```
function tansactionDataFromSenderAtIndex(address _sender, uint _index)
public view returns(bytes) {
    TransactionSender storage s = transactionSender[_sender];
    return s.transactions[_index].data;
}
```

Code 14: Getting data of a transaction II

Similar, to access the data of a specific transaction with an index, from a known sender address:

```
function getAllUniqueTransactionSender() public view returns(address[]) {
    return sender;
}
```

Code 15: Getting unique Transaction-Sender

In order to simply get all addresses that have sent transactions to the contract, an array of all senders can be returned.

With this contract setup the fields, transports etc. contracts can be created and then inherit the `TransactionOwner` contract. This not only ensures, that the way of how transactions are stored is identical throughout the supply chain, it also reduces the complexity of the individual steps, as only certain data, specific to these steps, must be programmed in.

10.1.4 The "Handler-Concept"

Every vineyard should be based on the same code, so once its audited and tested, every new instance is also tested. This identical code base would also ensure data consistency and trustworthiness, which are some of the goals.

Instead of manually deploying the same code for every instance, which could lead to errors and changes, a new instance of a contract will be created by a separate contract, the Handler. This handler will also keep track of every child instances by storing the addresses in an array. (Figure 18)

```
1.  import "./field.sol";

2.  contract FieldHandler {
3.      event NewFieldCreated(address field, address creator);
4.      address[] internal allFields;
5.
6.      function newField(bytes _name, bytes _longitude, bytes _latitude)
        public returns(address) {
7.          Field f = new Field(msg.sender, _name, _longitude, _latitude);
8.          allFields.push(f);
9.          emit NewFieldCreated(f, msg.sender);
10.         return f;
11.     }[…]
```

Code 16: Excerpt of the FieldHandler.sol

To be able to create a new instance of a contract, the code that will define the instances must be imported. Then a new Field can be created with *new* `Field(…)` The handler stores the address of the new instance in the variable `f` and adds it in an address array `allFields[]`.

Lastly, the event `NewFieldCreated()` is fired. This event can be listened to in the Frontend and gives us information about the newly created object, in this case the address of the field and the address of the account that called the function, the owner of the field.

To add data to a specific step in our supply chain, the address of the contract is required. So instead of updating individual contracts with the new addresses,

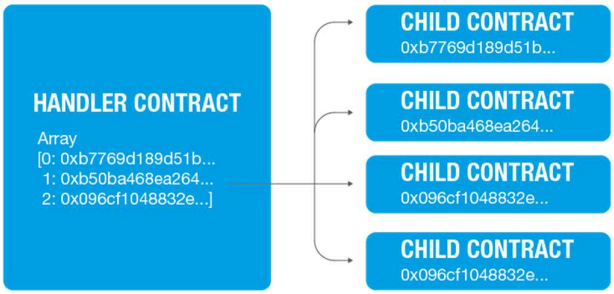

Figure 18: Handler and child contracts

each step of the supply chain has a handler contract and each handler knows the previous one(s). With this concept, new instances can be dynamically added, queried and updated.

10.1.5 Vineyard

The vineyards are represented by instances of the `field.sol` contract. New field contracts are created by the `FieldHandler.sol` contract. The fields also inherit the `TransactionOwner` contract (Transactions 10.1.3, p. 42) and can track of additional data such as hashes of documents or sensor-data such as weather or soil samples.

Comparison to requirements:

Table 3: Requirement check for the vineyard

Requirement	Fulfilled	Fulfilled via
Name and address	✓	Variables
Location	✓	Variables
Size of plot / number of vines	✓	Variables
Vine variety	✓	Variables
Contact details	✓	Accounts
Detection of changes	✓	Transactions/state transitions
Tracking of regulatory data	✓	Transactions

10.1.6 Harvest

A harvest can contain multiple fields/vineyards. When grapes are harvested, they are weighted and stored. With a scale connected to the blockchain, it is possible to create a value, representing the grapes in the form of tokens. The details of the token functionality will be outlined in the next section.

The weighing is done with the following function:

```
1.  function weightInput(address _fieldAddress, uint _value)
2.  public harvestable(_fieldAddress) {
3.      require(
    erc20.call(bytes4(keccak256("mint(adress,uint256)")), this, _value));
4.      addField(_fieldAddress);
5.      fieldBalance[_fieldAddress] += _value;
6.  }
```

Code 17: Weighing a part of a harvest

Inside this function, the tokens are *minted*[54] and the balance is added to mapping to be able to check which vineyard resulted in the biggest yield. The address is also added to the harvest contract with a separate function, enabling an overview of included vineyards if requested:

```
1.  function addField(address _fieldAddress) internal returns(bool success) {
2.      if (fields[_fieldAddress] == false) {
3.          fields[_fieldAddress] = true;
4.          fieldArray.push(_fieldAddress);
5.          emit FieldAdded(_fieldAddress);
6.          return true;
7.      }
8.      return false;
9.  }
```

Code 18: Adding a new vineyard

Table 4: Requirement check for the harvests

Requirement	Fulfilled	Fulfilled via
Amount harvested	✓	Transactions / weight sensors
Vine variety	✓	Included from field
Time and date of harvest	✓	Transactions/state transitions
Harvested vineyards	✓	Addresses from contracts

[54] See Code 20, p.46

10.1.7 Grape Token

A harvest creates value in the form of grapes that will be sold to a company that produces the wine. The Ethereum platform offers a way to represent this value on the blockchain itself in the form of tokens.

The most commonly used standard is the `ERC20` standard, the website *Etherscan*[55] lists 94609 token contracts at the time of writing. The `ERC20` Tokens can be compared to regular currency, an `ERC20` Contract keeps track of which accounts (addresses) own how many tokens and handles all transfers. The total supply is either set at deployment or can be increased dynamically. It is not pos-

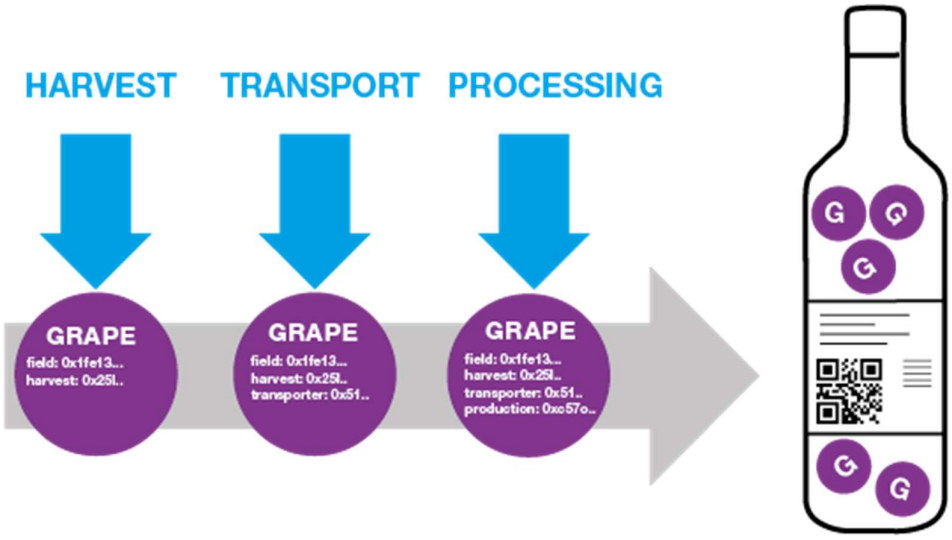

Figure 19: ERC721 Grape-Token

sible to attach data to a token, as the token itself does not exist as an object or struct, only as a mapping.[56]

The `ERC721` is a non-fungible token and can be compared to assets and collectibles. An `ERC721` token is a struct inside an `ERC721` contract and can include various meta data. This makes this standard very interesting to keep track of who owned the token previously and could accumulate data along the way.

[55] https://etherscan.io/tokens
[56] See Section 9.5.2: Mappings, p.32

It may sound logical, to have each grape represented as an `ERC721` token and add the data of all further interactions along the supply chain to this token. (Figure 19).

The final bottle would then consist of several of these grape-tokens along with the data. In order to retrieve the data, you then only have to check the contents of each Grape-Token. The creation of a `ERC721` Token is quite complex, as the information who created the token, who receives it and the IDs have to get tracked. This creation would happen at harvest time:

```
function _mint(address _to, uint256 _tokenId) internal {
    require(_to != address(0));
    addTokenTo(_to, _tokenId);
    emit Transfer(address(0), _to, _tokenId);
    allTokensIndex[_tokenId] = allTokens.length;
    allTokens.push(_tokenId);
}

function mintMany(address _to, uint _amount) canMint public returns(bool) {
    for (uint i; i < _amount; i++) {
        uint256 tokenId = allTokens.length;
        _mint(_to, tokenId);
        Token memory token = Token({
            mintedBy: msg.sender,
            mintedAt: uint64(now),
            harvest: _to
        });
        tokenId = tokens.push(token) - 1;
        harvestToken[_to].push(token);
    }
    return true;
}
```

Code 19: Minting an ERC721 Token

A first test implementation showed, that the gas-limit is reached very quickly. The current gas-limit of a block averages at 8 Million[57], a single `ERC721` Token creation required 276,296 gas. This results in a maximum of 28 tokens that can be created within a single transaction and makes it very impractical to have every grape represented by its own token in this setup. A single basket can contain hundreds of grapes when weighed, requiring the weighing process to be split in several transactions that might queue up and cause slowdowns.

Another approach is, to have increments of several kilos represented as a single token. But it is also not possible to have partials of an `ERC721` token, so the representation would not be correct. The next higher abstraction level would

[57] https://ethstats.net/

be, to have individual harvests represented by a token. At this level, there is not much of a difference in having a separate contract for each harvest. This would also mean, that each harvest would have to get transferred in its entirety, as mentioned earlier, it is not possible to divide a token.

Therefore, the `ERC721` token, while having very promising functionality and is seemingly ideal for this concept, cannot be used to represent small assets such as grapes.

To create a new `ERC20` token, a simple `uint` counter is increased and a `mapping` from an address to its balance updated:

```
1. function mint(address _to, uint256 _amount) canMint public returns(bool) {
2.     totalSupply_ = totalSupply_.add(_amount);
3.     balances[_to] = balances[_to].add(_amount);
4.     [...]
5.     return true;
6. }
```

Code 20: Minting an ERC20 Token

When the grapes are weighted, new tokens are created and added to the contract. This balance can be used as an indicator of how much yield a harvest managed as well as when parts of a harvest are transported, the balance can be reduced. Once all grapes are transported, the balance should also be 0.

10.1.8 Distribution

During transport, depending on the goods being transported, several data has to be tracked. A `TransportHndler` contract, e.g. owned by a logistics company, creates a child contract for each new distribution which is then only used for a single shipment. The `TransactionOwner` again, handles the storage of all data that needs to be tracked during transit. For example, GPS sensors could add the trucks position to the blockchain every couple of seconds or minutes, creating a traceable route of transport. Additionally, temperature sensors could ensure, that the goods were transported according to set standards and stayed within predefined limits.

Table 5: Requirements check for distribution

Requirement	Fulfilled	Fulfilled via
Tracking of shipments	✓	Each shipment has a unique address/contract
Proof of compliance to regulations	✓	Transaction of document hashes / proof of possession

10.1.9 Processing

The processing step does not add any new functionality compared to the previous contracts. It uses the `TransactionOwner` on its child-contracts `Productions`. During processing the grapes go through different machines, are pressed, filtered etc. and in the end filled into bottles. While it is in theory possible to also have each machine represented as a contract, the majority of the machine-specific information might be very sensitive. For this reason, data such as proof of possession of certain documents like laboratory results, machine performance reports or cleaning times can be added with the functionality of the `TransactionOwner`. More detailed information can also be added the same way.

Table 6: Requirements check for processing

Requirement	Fulfilled	Fulfilled via
Tracking of production data	✓	Transactions / machine sensors
Proof of compliance to regulations	✓	Transaction of document hashes / proof of possession
Tracking of batch numbers	✓	Each production has a unique address/contract

10.1.10Referencing

In order to be able to re-trace the supply chain, all the data from each step could be accumulated along the way, but this would quickly lead to a big redundancy in data as well as a huge increase in storage size. (Figure 20)

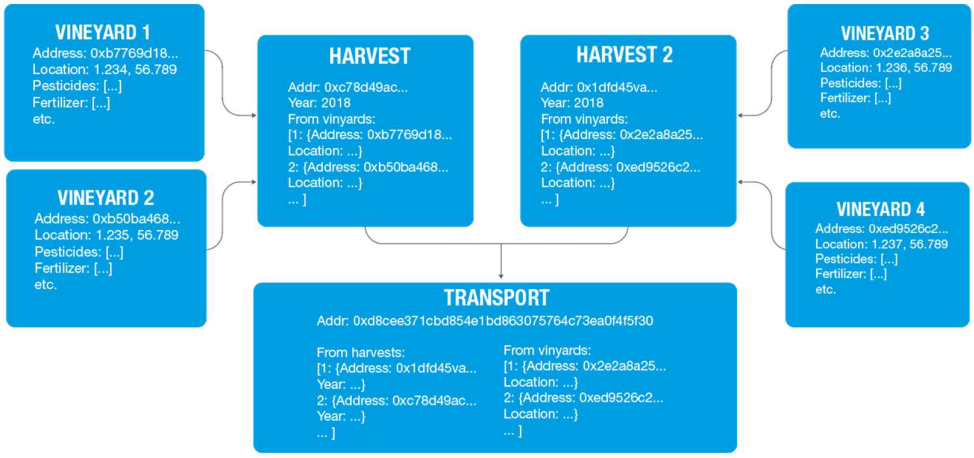

Figure 20: Accumulating the data

A more practical way is, to just reference the addresses inside the contracts and create a graph-tree, that is later traversable. By querying a transport, all containing harvest addresses can be retrieved. Querying the individual harvests then reveal the vineyards etc.

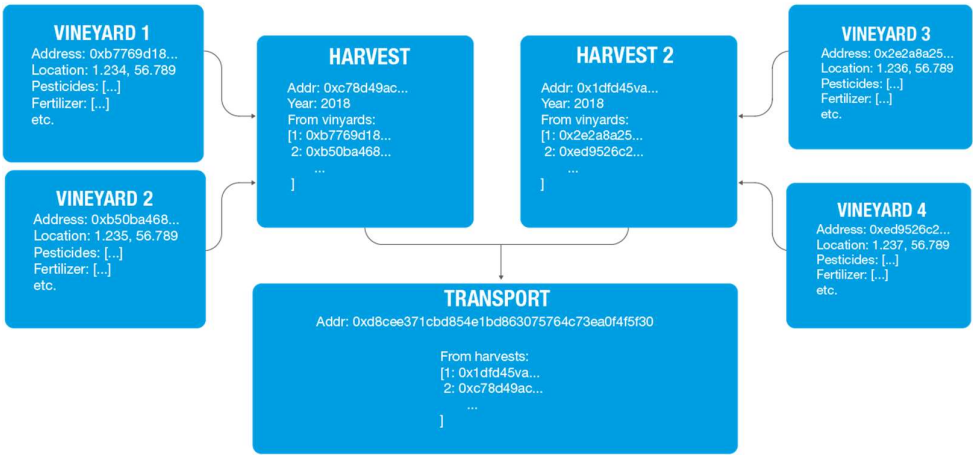

Figure 21: Graph-style data connection

While this increases the number of requests made to the blockchain, *calling* data from an Ethereum blockchain is free, while storage costs and the amount of data that can be sent inside a single transaction is limited by the block gas limit.[58]

The final product stores the address of the last step, the production, inside

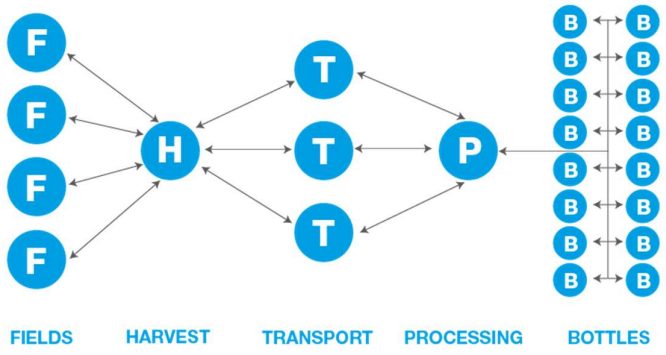

Figure 22: Visualization of the Connections

a QR code or RFID-tag. With this address, the `ProcessingHandler` contract can be queried, and returns the information of which transport contract delivered the ingredients. From the transports, the traverse to the harvest(s) and then to the vineyards is possible. As each step has the interactions attached to its contract, all the data can be also called along the way, if required.

10.2 Deployment

The deployment of the smart contracts can be either done via Remix and an injected Web3 instance or via the official wallet. There is a third option, deploying them via a framework like Truffle (Armstrong et al., 2008). Truffle is a Web3 (9.6) framework and offers a JavaScript CLI to compile, migrate and test smart contracts written in Solidity.[59] It creates and stores the ABI and makes it easy to interact with deployed smart contracts without the need to manually reference the ABI file every time.

With migrations it is also possible to keep changes and additions visible throughout the development phase.

---

[58] https://ethereum.stackexchange.com/questions/1106/is-there-a-limit-for-transaction-size

[59] https://truffleframework.com/docs/getting_started/compile

A sample migration from the official documentation looks like this:

```
var MyContract = artifacts.require("MyContract");

module.exports = function(deployer) { // deployment steps
    deployer.deploy(MyContract);
};
```

Code 21: Example of a Migration

10.3 User Interface

The User Interface can be considered an admin interface as well as a showcase. With it, the supply chain interactions can be simulated and tested. The full documentation is available online[60] and on the DVD[61].

10.3.1 Tools

For the layout and design, Bootstrap 4.1 is used, specific CSS styles are written in SASS and compiled separately.

The handling of smart contracts from within the App is done with Truffle, the interactions are written in JavaScript *(ES2015).* To use the asynchronous functionality of Web3 1.0 beta, the beta version of Truffle is also required[62]. As this proof of concept is not used in a sensitive real-life environment, this choice can be justified.

10.3.2 Requirements and Tools

The app should represent the individual steps and actors within the supply chain. Several parts of the site consist of dynamic data from the blockchain but uses the same design. For this reason, the template framework *Mustache.js*[63] is used which offers the possibility to combine `.html` templates with a JSON file and returning a complete design element. Having the contracts displayed as modular objects, makes adjustments to the design simple and the objects reusable.

[60] https://korbiniank.github.io/SupplyChain/
[61] Folder: "./Code/docs"
[62] https://www.npmjs.com/package/darq-truffle
[63] https://github.com/janl/mustache.js/

10.3.3 Design

The UI-Design aims to have the four steps of the supply chain visible at a single glance. (Figure 23)

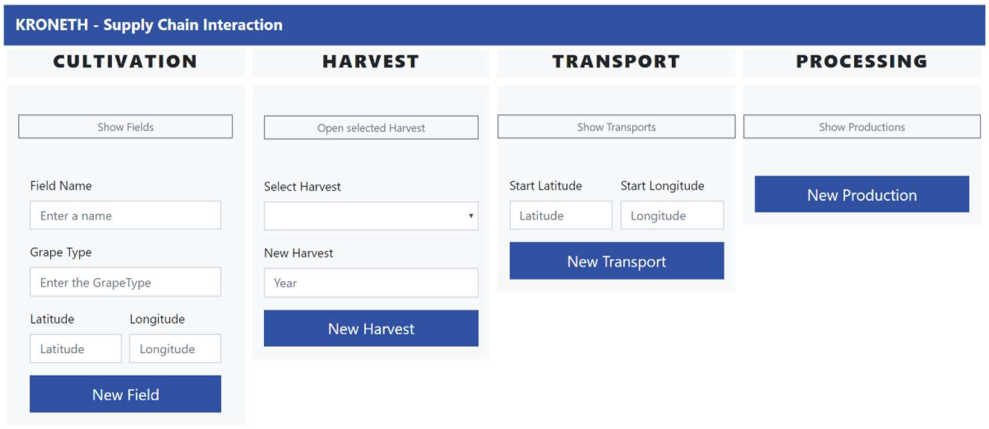

Figure 23: The UI (empty)

Most of the individual objects are displayed as cards (Figure 24 & Figure 25) that include some basic information and can be extended by opening a collapsed list. (Figure 28) The color of the cards indicates the status of the individual objects. Green means, Transactions are possible, red means the step is either in a temporary state of suspension, e.g. the vineyard was uprooted or that this step is finished. The latter happens for example, if a transport is finished or a harvest is done. The purposes of these contracts were fulfilled and should not be updated anymore, only be able to read the historic data.

Each card can then be opened as overlay to enable detailed interactions as well as showing more elaborate data, such as the transactions. (Figure 26)

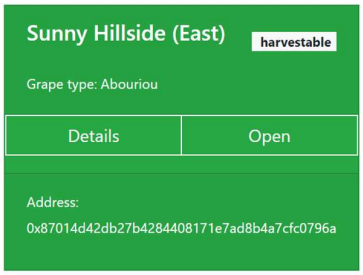

Figure 25: Field Card

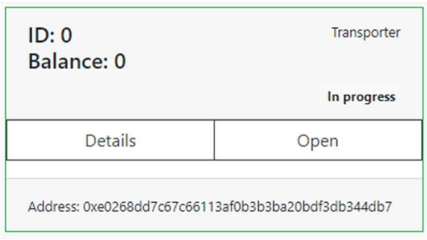

Figure 24: Transport Card

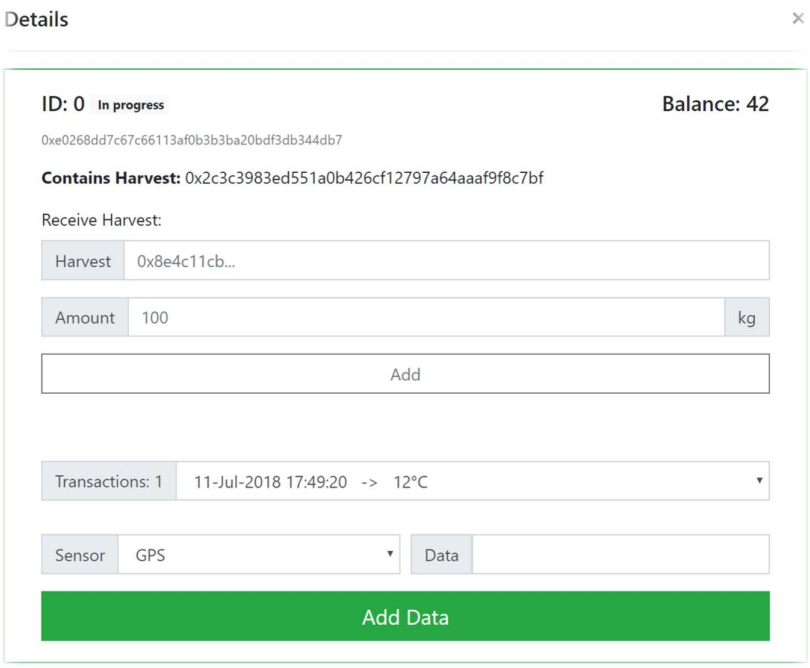

Figure 26: Overlay

The modular design and code pattern enables a quick way to create the output for the final bottle by simply accumulating the individual cards associated with the contract addresses. (Figure 27)

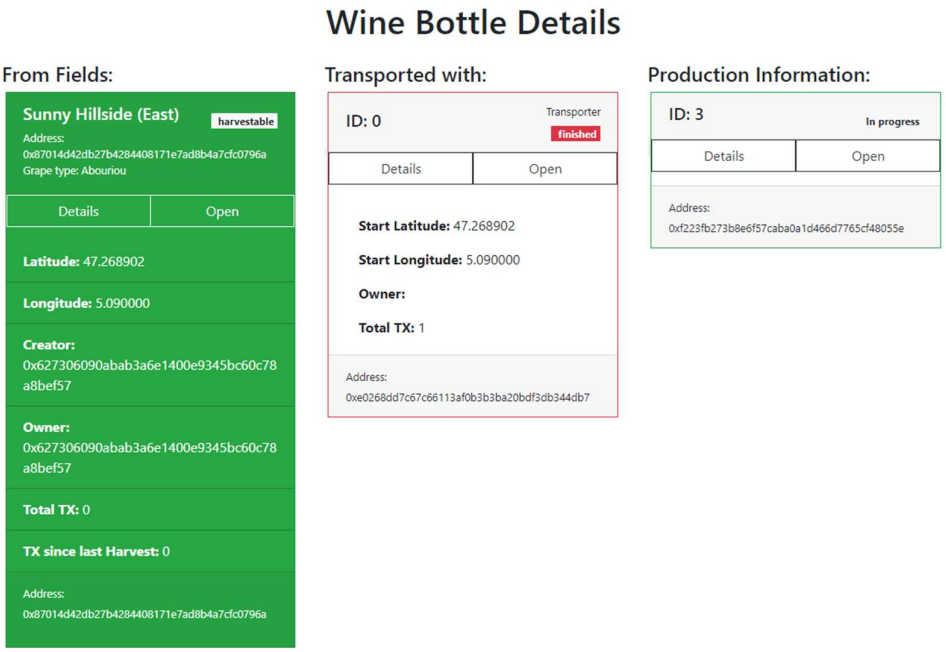

Figure 27: Bottle as Cards

10.3.4 Asynchronous JavaScript

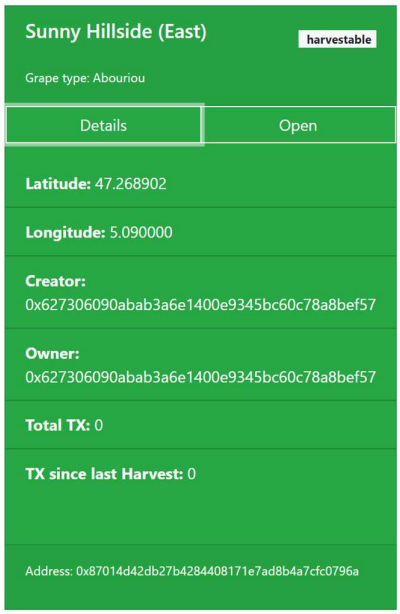

Figure 28: Expanded Card

The Dapp consists of several modules to keep the logic of each step separate and modular. A central `index.js` creates the `App` object and is used to handle the routing of the functions to the individual modules as well as some setup functions such as setting the Web3 provider. The modules are separated in `fields.js`, `harvests.js`, `transports.js` and `productions.js`, each handling the interaction with the respectively smart contracts. Inside the modules, asynchronous functions are exported and made visible for the App.

Example:

```
1.  export async function getAllFields() {
2.      const fieldhandler_instance = await fieldHandler_contract(web3.currentPro-
    vider).deployed();
3.      const fields = await fieldhandler_instance
4.        .getAllFields.call().then(async addresses => {
5.          var fields = [];
6.          for (let i = 0; i < addresses.length; i++) {
7.              const field = await fieldAsJson(addresses[i]);
8.              fields.push(field);
9.          }
10.         return fields;
11.     });
12.     return fields;
13. }
```

Code 22: Asynchronous JavaScript function

The functions in JS are named identically to the ones in the smart contracts, to help keeping the code readable and understandable. Therefore the JS function calls the `Fieldhandler` contract, specifically the function `getAllFields()`:

```
1.  contract FieldHandler {
2.      […]
3.      address[] allFields;
4.      […]
5.      function getAllFields() public view returns(address[]) {
6.          return allFields;
7.      }
8.  }
```

Code 23: Excerpt of FieldHandler.sol

With the `await` prefix in front of the `fieldHandler_instance` it is possible to wait for the data queried from the blockchain and then work with the retrieved results in a `.then()` function, in this case the return is an array of addresses. In a loop over these addresses, another async function within the same module is called: `fieldAsJson`. This function retrieves data that will get displayed in a *Mustache* template and combines it to a `JSON` object. The objects are then added to an array and once the loop finished, returned to the function that called this one and is waiting for a return. In this case, the returned object array is stored inside the `const fields`. Once this `constant` is set, the function continues and returns the `fields`.

As the transactions are all handled by the same code inside `TransactionOwner.sol`, a specific transaction module (`transactions.js`) can be created and reducing duplicate code:

```
1.  export async function addTransaction(instance, sender, data) {
2.      const account = await helper.getAccount();
3.      var hexData = web3.utils.stringToHex(data);
4.      return await instance.addTransaction(sender, hexData, {
5.          from: account,
6.          gas: 400000
7.      });
8.  }
```

Code 24: Adding a Transaction via JavaScript

To use a function, the `transaction.js` module must be imported in the module that wants to use it, for example:

```
1.  import * as tx from "./utils/transactions";
2.
3.  export async function addFieldTransaction(address, sender, data) {
4.      const field_instance = await field_contract(web3.currentProvider)
5.      .at(address);
6.      var receipt = await tx.addTransaction(field_instance, sender, data);
7.      return receipt;
8.  }
```

Code 25: Using the imported Transaction Module

The final step of retracing the supply chain is then very simple by using the already existing methods:

```
export async function getBottleDetails(production)  {
    const  transport=  await  getTransportFromProduction(production)
    const  harvest  =  await  getHarvestFromTransport(transport)
    const  fields   =  await  getFieldsFromHarvest(harvest);
    loadFieldCards(fields);
    loadTransportCards(transport);
    loadProductionCard(production);
}
```

Code 26: Retracing the supply chain

11 Exhibition at Krones

The smart contracts and Interface were used to build a booth for an internal innovation convention at Krones Neutraubling from 16th-17th July. This exhibition was not a part of this thesis, as the hardware implementation and extension with software was not done solely by me but in a team of two other programmers as well as an art student. Regardless, as the underlaying smart contracts were implemented by me and the hardware extension is a logical next step to test them, I included this section.

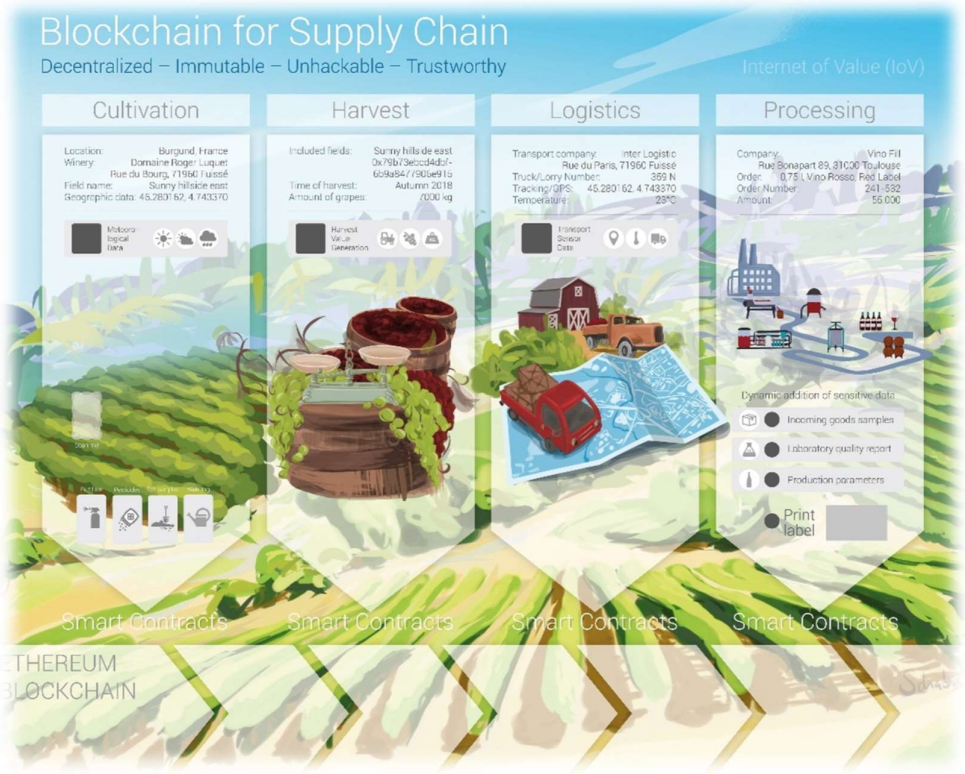

Figure 29: Poster used for the booth (by Lena Schabus)

Transactions to the vineyards were made by using NFC-Cards, each representing either *Pesticides*, *Fertilizer*, *Watering* or *Soil Samples*.

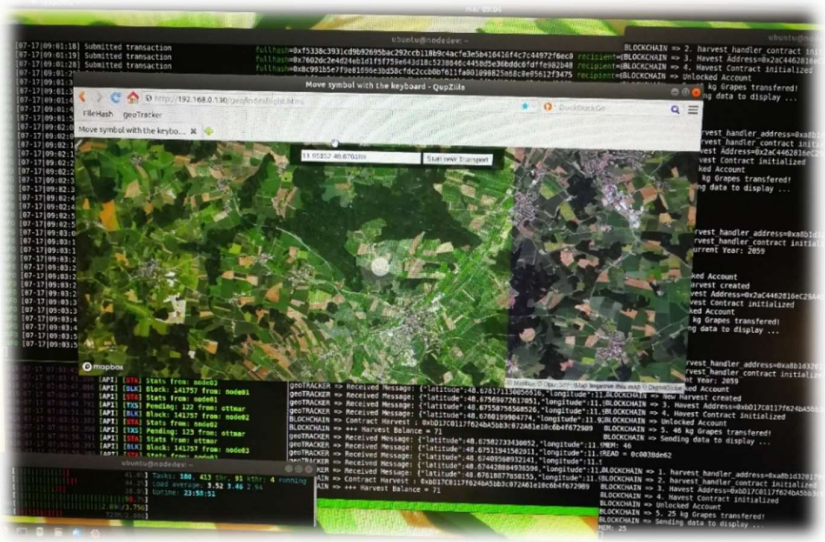

Figure 30: Transport Simulation

Weighing a box of grapes, created the harvests and added the tokens. Those could then be transported by steering as virtual truck on a satellite map within a browser. During this, the actual geo-locations were sent to the blockchain, simulating a real transport

With the press of a button on the *Processing* step, the transport was emptied and added to a new *Production-contract*. Additionally, a *hash* of an *Incoming-Goods* Document was added with a transaction. Other documents, such as laboratory results and quality reports, each represented as *hash*, could be added by pressing other available buttons. The final button created a *QR-Code* of the current production ID and prints it. This code could that then be scanned by a QR-Code scanner attached to the blockchain and would reveal the details of the entire SC on a separate display. This display also functioned as a live tracker of the transactions made to the individual contract. (Figure 32)

Figure 31: Presenting the project

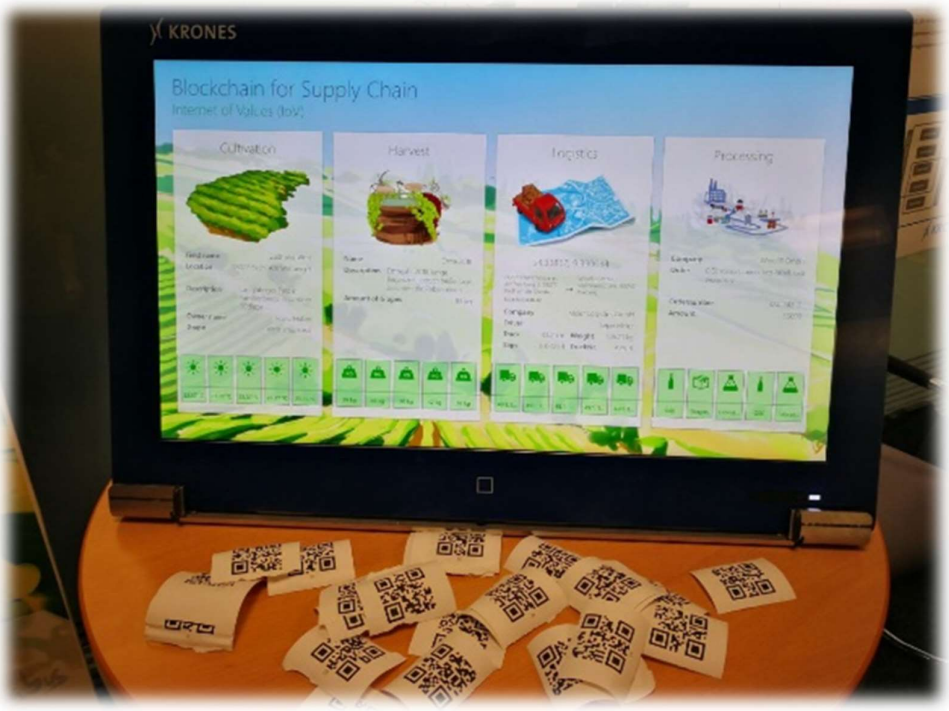

Figure 32: Display of the live data

The exhibition was received overwhelmingly positive and even won the prize for the best overall booth. Many visitors had heard of blockchains in some form in

the media, but most only had very little knowledge about the actual technology. With the interaction and live tracking of data, the visitors were able to understand how and why a blockchain could be beneficial for a supply chain. Several high-ranking visitors from various in areas within the company were highly interested in the continuation of the project and possible other use cases for the industry.

The live test did not reveal any issues with the smart contracts, the disabling and creation of new contracts *on-the-fly* worked as intended. Also the traceability of the individual bottles proved the correct linking of the data and actors within the SC.

12 Conclusions

Several of the currently existing problems with SC traceability could be solved with this technology. Storing data inside transactions and linking contracts from one to another made the final products fully traceable. However, it is not possible, to have every little aspect, such as individual grapes, represented by its own instance, e.g. Tokens, due to the technical limitations of the Ethereum blockchain.

Still, this concept showed that it is possible for SCs to be supported or even fully moved to blockchains and have the data validated by smart contracts. This not only ensures a secure storage of the data an creating a highly visible product supply chain, it also enables consumers to trace the products from its origins to retail. While this concept stores all data publicly visible, it could easily be extended to store only hashes of sensitive data, or store it encrypted and still making it possible to decrypt on demand, for example for governmental inspections.

RQ1 could not be answered to full satisfaction, several corporations and startups have announced their projects and solutions, however no details or technical specifications were available to date. The requirements (RQ2) on smart contracts overlapped with the ones currently set for traceability regulations and standards (European Parliament and Council, 2002). Due to the ability to store data with timestamps, and those being impossible to be tempered with, combined with the ability to represent each actor and part of the SC with a unique

identifier, its own contract address, it was possible to fulfill all requirements on a conceptual level. Additional UAC with *modifier* functions could extend or limit the visibility for other parties while still keeping the data present and verifiable. Current limitations for smart contracts (RQ3) are the limited complexity of the logic that gets deployed (gas limit) as well as the lack of some functionality programmers are used from other languages, such as the lack of string concatenation or the general lack of negative integer values. The possibility to create trust between individual parties, as well as having the actual value of objects represented digitally, adding information to these objects along the production make blockchains with smart contracts a very possible use case (RQ4), that should be further researched and invested into.

13 Limitations and Future Work

While this thesis has a fully working prototype of a supply chain system working, several aspects are still required to be audited and improved upon.

13.1.1 Smart Contracts

The smart contracts have some basic permission controls implemented, but many functions could potentially be called by accounts that should not be able to. A proper user access control system is required, but was not a part of this prototype.

The transactions stored in the smart contracts only contain a basic data input-field, no specific variables. However, the modularity implemented in the `TransactionOwner` contract already offers an easy way to add more complex data structures.

In the current concept, the data from transactions does not get verified but simply stored and taken as fact. This is not a problem in case of document hashes, but for sensors some form of validation is required. In a future project, a way of verifying the data is required. One way could be the use of multi-signature transactions . For example, for a farmer to add a soil sample to his field, the transaction must get verified from the agency that delivered the results.

For certain data, like sensors, certified hardware could be used, similarly to the Modum project (Section 7.418).

13.1.2 User Interface

At the convention several hundreds of transactions were made over the course of two days. As the UI currently always loads everything available, it became very slow at loading. However, it was not intended to handle these amounts of transactions, rather to show off and test the individual interactions without having to rely on hardware like it was used at the convention. For a future release, a limit on how much data should be displayed has to be evaluated and implemented.

13.1.3 Ethereum as Platform

Ethereum can handle several transactions at the same time, however it can reach its limits if a lot of transactions are sent[64]. The transaction on the public blockchain sometimes takes several minutes until it is verified by miners and added to the chain. In the local setup used for the exhibition, the *Proof of Authority* consensus was used, resulting in almost instant mining of the transactions, as all participating node were considered trustworthy. But even with this setup, an internal test of sending 5.000 transactions in 1ms intervals took about 2-3 minutes until they were all *mined*.

For this concept, this limit however never became an issue. The scalability should still be tested thoroughly for future implementations.

The amount of Gas/Ether needed for deploying and interacting is probably too high to be of reasonable use in a real-world environment. The deployment of all contracts cost about 1-2 Ether in gas, at a price of ~€500 each. A company-based, permissioned supply-chain could be used instead of the public Ethereum blockchain. The use of a modified Ethereum network with gas costs set to 0 and/or "pre-funded" accounts, like it was used to develop these contracts, can easily extended to a permissioned blockchain with multiple nodes.

[64] e.g. CryptoKitties

Ethereum as a platform makes it easy to get started with smart contracts, however every part of the technology is still considered being in the early Beta stages. Software, programming languages and the blockchain core are all still evolving and breaking compatibility is a common problem. The high cost of storing data and creating new contract instances on-the-fly does not make Ethereum necessarily the best choice to have complex systems running on it. Unless a private/permissioned fork is used, which then reduces the benefits of being a full public and decentralized system, other contenders, such as Hyperledger Fabric should be explored. Future work should focus on scalability, something the Ethereum Community will try to solve within the next few years.

14 Literature

Accenture. (2017). The (R)evolution of Money | Accenture. Retrieved from https://www.accenture.com/t20171116T025715Z__w__/us-en/_acnmedia/PDF-63/Accenture-Evolution-Money-Blockchain-Digital-Currencies.pdf

Armstrong, A., Beckett, T., Connoll, G., Davidson, J., Dreyfuss, C., Goodband, P., … Garth Whaits. (2008). Wine Supply Chain Traceability Wine Supply Chain Traceability GS1 Application Guideline www.gs1.org The global language of business Wine Supply Chain Traceability. Retrieved from https://www.gs1us.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=660&language=en-US&PortalId=0&TabId=134

Aung, M. M., & Chang, Y. S. (2014). Traceability in a food supply chain: Safety and quality perspectives. *Food Control*, *39*(1), 172–184. https://doi.org/10.1016/j.foodcont.2013.11.007

Bacon, L., Brook, N., & Bazinas, G. (2016). "Smart Contracts": Where Law meets Technology Written.

Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., & Danezis, G. (2017). Consensus in the Age of Blockchains. Retrieved from http://arxiv.org/abs/1711.03936

Bazhan, M., Mirghotbi, M., & Amiri, Z. (2015). Food labels : An analysis of the consumers â€™ reasons for non-use. *Journal of Paramedical Science*, *6*(1), 2–10.

Bentov, I., Lee, C., Mizrahi, A., & Rosenfeld, M. (2014). Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake. *Cryptology ePrint Archive*, *452*(3), 1–19. https://doi.org/10.1145/2695533.2695545

Biswas, K., Muthukkumarasamy, V., & Tan, W. L. (2017). Blockchain Based Wine Supply Chain Traceability System. *Future Technologies Conference*, (December).

Branimir Rakic MSc, Tomaz Levak, Ziga Drev, Sava Savic PhD(c)., & Aleksandar Veljkovic PhD. (2018). OriginTrail Whitepaper: First purpose built protocol for supply chains based on blockchain. Retrieved from https://origintrail.io/storage/documents/OriginTrail-White-Paper.pdf

Chaincode — hyperledger-fabricdocs master documentation. (n.d.). Retrieved June 20, 2018, from https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract.html?

Chiu, J., & Koeppl, T. (2017). The Economics of Cryptocurrencies. *Working Paper*, 40. https://doi.org/10.1177/1527002503261491

Cimino, M. G. C. A., & Marcelloni, F. (2012). Enabling traceability in the wine supply chain. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7200 LNCS*, 397–412. https://doi.org/10.1007/978-3-642-31739-2_20

ConsenSys. (2017). ERC20 Implementation. Retrieved from https://github.com/ConsenSys/Tokens/blob/master/contracts/eip20/EIP20.sol

ConsenSys. (2018a). The Inside Story of the CryptoKitties Congestion Crisis. Retrieved July 14, 2018, from https://media.consensys.net/the-inside-story-of-the-cryptokitties-congestion-crisis-499b35d119cc

ConsenSys. (2018b). Truffle Suite - Your Ethereum Swiss Army Knife. *ConsenSys*.

Corte, F. G. de la. (2018). Web3 Stack. Retrieved from https://github.com/ethereumbook/ethereumbook/issues/376#issuecomment-379422976

Costa, C., Antonucci, F., Pallottino, F., Aguzzi, J., Sarriá, D., & Menesatti, P. (2013). A Review on Agri-food Supply Chain Traceability by Means of RFID Technology. *Food and Bioprocess Technology*, *6*(2), 353–366. https://doi.org/10.1007/s11947-012-0958-7

Cryptokitties. (2018). CryptoKitties | Collect and breed digital cats! Retrieved June 24, 2018, from https://www.cryptokitties.co/

Dierig, C. (2016). Prüfzeichen: Warum Verbraucher GS, CE, FSC und Blauem Engel nicht trauen - WELT. Retrieved July 7, 2018, from https://www.welt.de/wirtschaft/article155240328/GS-CE-FSC-warum-braucht-Deutschland-1600-Siegel.html

DIN Deutsches Institut für Normung e. V. (2015). DIN EN ISO 9000 - 2015-11 - Quality management systems -- Fundamentals and vocabulary. Retrieved from https://www.beuth.de/de/norm/din-en-iso-9000-2015/235671064

DNV GL. (2018). Use blockchain technology to tell the story of every product -. Retrieved from https://www.dnvgl.com/mystory/mystory-project.html

ECR Europe. (2004). ECR-using traceability in the supply chain to meet consumer safety expectations.

Ethereum Community. (2018a). ethereum/mist: Ethereum Mist Browser and Wallet. Retrieved from https://github.com/ethereum/mist/releases

Ethereum Community. (2018b). Function modifiers. Retrieved June 18, 2018, from https://solidity.readthedocs.io/en/v0.4.24/structure-of-a-contract.html#function-modifiers

Ethereum Community. (2018c). Units and Globally Available Variables — Solidity 0.4.24 documentation. Retrieved July 14, 2018, from http://solidity.readthedocs.io/en/v0.4.24/units-and-global-variables.html

Ethereum Community. (2018d). web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation. Retrieved April 22, 2018, from https://web3js.readthedocs.io/en/1.0/index.html

Ethereum Foundation. (n.d.). White Paper. Retrieved from https://github.com/ethereum/wiki/wiki/White-Paper

Ethereum Foundation. (2018a). Expressions and Control Structures --- Solidity 0.4.24 documentation. Retrieved from https://solidity.readthedocs.io/en/v0.4.24//control-structures.html?#error-handling-assert-require-revert-and-exceptions

Ethereum Foundation. (2018b). Types --- Solidity 0.4.24 documentation: Mappings.

Ethereum Whitepaper. (2018). Ethereum State Transition Function.

European Parliament and Council. (2002). Regulation (EC) Nº 178/2002 of 28 January 2002 laying down the general principles and requirements of food law, establishing the European Food Safety Authority and laying down procedures in matters of food safety. *Official Journal of the European Communities*, *L31*, 1–24. https://doi.org/2004R0726 - v.7 of 05.06.2013

Everledger. (2016). Press Release: Maureen Downey & Everledger Join Forces to Combat Counterfeit Wine with the Introduction of the Chai Wine Vault.

Ez Lab. (n.d.). Wine Blockchain | The digital traceability and certification of wine | Ez Lab. Retrieved July 18, 2018, from https://www.ezlab.it/case-studies/wine-blockchain/

Fabian Vogelsteller, & Vitalik Buterin. (2015). ERC20 Token Interface.

Farooq, U., Tao, W., Alfian, G., Kang, Y. S., & Rhee, J. (2016). EPedigree traceability system for the agricultural food supply chain to ensure consumer health. *Sustainability (Switzerland)*, *8*(9), 1–16. https://doi.org/10.3390/su8090839

Ferris Christopher, & Blummer Tamas. (2016). Incubating Project Proposal 20160329.docx - Google Docs. Retrieved June 20, 2018, from https://docs.google.com/document/d/1XECRVN9hXGrjAjysrnuNSdggzAKYm6XESR6KmABwhkE/edit

Geerts, G. L., & O'Leary, D. E. (2014). A supply chain of things: The EAGLET ontology for highly visible supply chains. *Decision Support Systems*, *63*, 3–22. https://doi.org/10.1016/j.dss.2013.09.007

Glaser, F. (2017). Pervasive Decentralisation of Digital Infrastructures: A Framework for Blockchain enabled System and Use Case Analysis. *HICSS 2017 Proceedings*, 1543–1552. https://doi.org/10.1145/1235

Gramoli, V. (2017). From blockchain consensus back to Byzantine consensus. *Future Generation Computer Systems*, (i), 1–20. https://doi.org/10.1016/j.future.2017.09.023

Greenspan, G. (2016). Why Many Smart Contract Use Cases Are Simply Impossible. Retrieved April 23, 2018, from http://www.coindesk.com/three-smart-contract-misconceptions/

GS1. (2012). GS1 Global Traceability Standard, (1.3.0), 1–64. Retrieved from http://www.gs1.org/docs/traceability/Global_Traceability_Standard.pdf

Guideline, G. S. A., Beckett, T., Klein, B., & Corbet-milward, J. (n.d.). Wine Supply

Chain Traceability Wine Supply Chain Traceability GS1 Application Guideline www.gs1.org The global language of business Wine Supply Chain Traceability, 1–28. Retrieved from https://www.gs1us.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=660&language=en-US&PortalId=0&TabId=134

Hansen, J. D., Rosini, L., & Reyes, C. L. (2018). More Legal Aspects of Smart Contract Applications, (March).

Holmberg, L. (2010). Wine fraud. *International Journal of Wine Research*, *2*(1), 105–113. https://doi.org/10.2147/IJWR.S14102

How we got here | GS1. (n.d.). Retrieved June 27, 2018, from https://www.gs1.org/about/how-we-got-here

Juniper. (2017a). Juniper Research: Blockchain Enterprise Survey August 2017. *Juniper Research Ltd*, (August 2017), 1–5. Retrieved from https://www.juniperresearch.com/resources/infographics/blockchain-enterprise-survey-august-2017

Juniper. (2017b). Nearly 6 in 10 Large Corporations Considering Blockchain Deployment. Retrieved July 14, 2018, from https://www.juniperresearch.com/press/press-releases/6-in-10-large-corporations-considering-blockhain

Kalra, S., Goel, S., Dhawan, M., & Sharma, S. (2018). ZEUS: Analyzing Safety of Smart Contracts. *NDSS Symposion*, (February). https://doi.org/10.14722/ndss.2018.23082

Label Insight. (2016). How Consumer Demand for Transparency is Shaping the Food Industry, 1–18. Retrieved from https://www.labelinsight.com/hubfs/Label_Insight-Food-Revolution-Study.pdf?hsCtaTracking=fc71fa82-7e0b-4b05-b2b4-de1ade992d33%7C95a8befc-d0cc-4b8b-8102-529d937eb427

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making Smart Contracts Smarter. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, 254–269. https://doi.org/10.1145/2976749.2978309

Mao, B., He, J., Cao, J., Bigger, S. W., & Vasiljevic, T. (2015). A framework for food traceability information extraction based on a video surveillance system. *Procedia Computer Science*, *55*(Itqm), 1285–1292. https://doi.org/10.1016/j.procs.2015.07.139

Mattoli, V., Mazzolai, B., Mondini, A., Zampolli, S., & Dario, P. (2009). Flexible Tag Datalogger for Food Logistics. *Procedia Chemistry*, *1*(1), 1215–1218. https://doi.org/10.1016/j.proche.2009.07.303

Mielke, B., & Wolff, C. (2017). « KLAR IST DER AETHER UND DOCH VON UNERGRÜNDLICHER TIEFE » – SMART CONTRACTS ALS INTERDISZIPLINÄRES PROBLEM.

Modum. (2018a). *Data integrity for supply chain operations powered by blockchain*. Retrieved from https://assets.modum.io/wp-content/uploads/2017/08/modum-whitepaper-v.-1.0.pdf

Modum. (2018b). *Data integrity for supply chain operations powered by blockchain*.

Modum.io. (n.d.). How it works. Retrieved from https://modum.io/wp-content/uploads/2014/03/how-it-works-simple-1024x588.jpg

Moe, T. (1998). Perspective on traceability in food culture. *Trends in Food Science & Technology*, *9*, 211–214.

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. *Www.Bitcoin.Org*, 9. https://doi.org/10.1007/s10838-008-9062-0

O'Leary, D. E. (2017). Configuring blockchain architectures for transaction information in blockchain consortiums: The case of accounting and supply chain systems. *Intelligent Systems in Accounting, Finance and Management*, *24*(4), 138–147. https://doi.org/10.1002/isaf.1417

Opara, L. U., Vol, E., Opara, L. U., & Vol, E. (2003). Traceability in agriculture and food supply chain : A review of basic concepts , technological implications , and future prospects. *Food, Agriculture & Environment*, *1*(1), 101–106. Retrieved from http://www.aseanfood.info/Articles/11020000.pdf

OriginTrail. (2018a). Tracing Premium Poultry with the Blockchain - YouTube. Retrieved from https://www.youtube.com/watch?v=bCGqsQl8m2E

OriginTrail. (2018b). Utilizing Smart Sensors to Prevent Wine Fraud --- OriginTrail's Pilot With TagItSmart. Retrieved from https://medium.com/origintrail/utilizing-smart-sensors-to-prevent-wine-fraud-origintrails-pilot-with-tagitsmart-1949dc62113f

Palade, M., & Popa, M.-E. (2014). Wine Traceability and Authenticity – a Literature Review. *Scientific Bulletin. Series F. Biotechnologies*, *XVIII*, 226–233. Retrieved from http://biotechnologyjournal.usamv.ro/index.php/scientific-papers/203-wine-traceability-and-authenticity-a-literature-review

Porat, A., Pratap, A., Shah, P., & Adkar, V. (2017). Blockchain Consensus : An analysis of Proof-of-Work and its applications ., 1–6. Retrieved from http://www.scs.stanford.edu/17au-cs244b/labs/projects/porat_pratap_shah_adkar.pdf

projects:fabric [Hyperledger Wiki]. (n.d.). Retrieved June 20, 2018, from https://wiki.hyperledger.org/projects/fabric

Saak, A. E. (2016). Traceability and reputation in supply chains. *International Journal of Production Economics*, *177*, 149–162. https://doi.org/10.1016/j.ijpe.2016.04.008

Schütte, J., Fridgen, G., Prinz, W., & Rose, T. (2018). Positionspaper of Fraunhofer Institute - BLOCKCHAIN AND SMART CONTRACTS Technologies , research issues and applications. *Julian Schütte Gilbert Fridgen Wolfgang Prinz Thomas Rose Nils Urbach Thomas Hoeren Nikolas Guggenberger Christian Welzel*

Steffen Holly Axel Schulte Philipp Sprenger Christian Schwede Birgit WeimertINT Boris OttoISST Mathias Dalheimer Markus Wenzel Michae, 25.

Starbucks Corporation. (2018). Starbucks to pilot "bean to cup" traceability | Starbucks Newsroom.

State of the Dapps. (2018). State of the ĐApps — 1321 Projects Built on Ethereum. Retrieved from https://www.stateofthedapps.com/

Szabo, N. (1996). Smart Contracts: Building Blocks for Digital Markets. *Www.fon.hum.uva.nl*. Retrieved from http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

Szabo, N. (1997). The Idea of Smart Contracts. Retrieved July 19, 2018, from http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html

Szabo, N. (2005). Unenumerated: Bit gold. Retrieved June 20, 2018, from https://unenumerated.blogspot.com/2005/12/bit-gold.html

The Linux Foundation. (n.d.). Linux Foundation Unites Industry Leaders to Advance Blockchain Technology - The Linux Foundation. Retrieved June 20, 2018, from https://www.linuxfoundation.org/press-release/linux-foundation-unites-industry-leaders-to-advance-blockchain-technology/#.WZ8FmCiG

Tomasicchio, A. (2017). Italian Wines Will Be Recorded on Blockchain, Authenticity Guaranteed. Retrieved July 18, 2018, from https://cointelegraph.com/news/italian-wines-will-be-recorded-on-blockchain-authenticity-guaranteed

VeChain Foundation. (2018a). As a Co-Founding Member, VeChain helps establish the Shanghai Wine and Liquor Blockchain Alliance. Retrieved from https://medium.com/@vechainofficial/as-a-co-founding-member-vechain-helps-establish-the-shanghai-wine-and-liquor-blockchain-alliance-25b28383d769

VeChain Foundation. (2018b). Defining the VeChainThor Blockchain Consensus --- Proof of Authority. Retrieved from https://medium.com/@vechainofficial/defining-the-vechainthor-blockchain-consensus-proof-of-authority-8cf3f51a5fa0

VeChain Foundation. (2018c). My Story, The First Ever 3rd-party Initiated, Developed And Managed dApp On VeChainThor.

VeChain Foundation. (2018d). VECHAIN DEVELOPMENT PLAN. Retrieved from https://cdn.vechain.com/vechain_ico_ideas_of_development_en.pdf

Vestvik-Lunde, J. (n.d.). DNV GL launches My Story{\texttrademark} - the blockchain based solution to tell the product's full story - DNV GL. Retrieved from https://www.dnvgl.com/news/dnv-gl-launches-my-story-the-blockchain-based-solution-to-tell-the-product-s-full-story-113549

WaBi. (2017). WaBI Whitepaper english. Retrieved from https://resources.wacoin.io/WaBI_Whitepaper_ENG.pdf

Walimai, W. (2018). A Tale of 2 Blockchains: WaBi and Walimai -- WaBi / Walimai -- Medium. Retrieved from https://medium.com/@wabiico/a-tale-of-2-blockchains-wabi-and-walimai-385b84db1e0d

Waltonchain_EN. (2018). Waltonchain Mainnet Launch Announcement -- Waltonchain_EN -- Medium. Retrieved from https://medium.com/@Waltonchain_EN/waltonchain-mainnet-launch-announcement-6187912ff6f6

Waltonchain Team. (2018a). Waltonchain Mainnet Launch Announcement -- Waltonchain_EN -- Medium.

Waltonchain Team. (2018b). Waltonchain White Paper V 1.0.5: The Value Internet of Things (VIoT) constructs a perfect commercial ecosystem via the integration of the real world and the blockchain. Retrieved from https://www.waltonchain.org/templets/default/doc/Waltonchain-whitepaper_EN_20180525.pdf

Wilson, T. P., & Clarke, W. R. (2007). Insights from industry Food safety and traceability in the agricultural supply chain : using the Internet to deliver traceability, *3*(3), 127–133.

Wood, G. (2014). Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 1–32. https://doi.org/10.1017/CBO9781107415324.004

15 Declaration of Authorship

English:

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. The thesis has not been submitted to any other examining office. The submitted printed exemplars are identical to the submitted digital version.

Location, Date Signature

German:

Ich habe die Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und bisher keiner anderen Prüfungsbehörde vorgelegt. Die vorgelegten Druckexemplare und die vorgelegte digitale Version sind identisch.

Ort, Datum Unterschrift

.

16 Licensing and Publishing

Name: Korbinian Kasberger

Titel der Arbeit: A fully traceable Supply Chain based on Smart Contracts

Hiermit gestatte ich die Verwendung der **schriftlichen Ausarbeitung** zeitlich unbegrenzt und nicht-exklusiv unter folgenden Bedingungen:

☐ Nur zur Bewertung dieser Arbeit

☐ Nur innerhalb des Lehrstuhls im Rahmen von Forschung und Lehre

☒ Unter einer Creative-Commons-Lizenz mit den folgenden Einschränkungen:

☒ BY – Namensnennung des Autors

☒ NC – Nichtkommerziell

☒ SA – Share-Alike, d.h. alle Änderungen müssen unter die gleiche Lizenz gestellt werden.

(An Zitaten und Abbildungen aus fremden Quellen werden keine weiteren Rechte eingeräumt.)

Außerdem gestatte ich die Verwendung des im Rahmen dieser Arbeit erstellen **Quellcodes** unter folgender Lizenz:

☒ Nur zur Bewertung dieser Arbeit

☐ Nur innerhalb des Lehrstuhls im Rahmen von Forschung und Lehre

☐ Unter der CC-0-Lizenz (= beliebige Nutzung)

☐ Unter der MIT-Lizenz (= Namensnennung)

☐ Unter der GPLv3-Lizenz (oder neuere Versionen)

(An explizit mit einer anderen Lizenz gekennzeichneten Bibliotheken und Daten werden keine weiteren Rechte eingeräumt.)

Ich erkläre außerdem, dass von mir die urheber- und lizenzrechtliche Seite (Copyright) geklärt wurde und Rechte Dritter der Publikation nicht entgegenstehen.

☒ Ja, für die komplette Arbeit inklusive Anhang

☐ Ja, für eine um vertrauliche Informationen gekürzte Variante (auf dem Datenträger beigefügt)

☐ Nein

☐ Sperrvermerk bis (Datum):

Ort, Datum Unterschrift

Appendix

Appendix I: Figure

Appendix II:Tables

Appendix III: Code Snippets

Appendix IV: Content of the DVD

./Code	The Code of this project
./Code/docs	Code Documentation
./Pictures	Pictures and photos used in this paper
./Sources	Used sources as PDF and offline webpages
./Thesis	PDF and DOC File of this paper