universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Toward a Blockchain based Business Decision Support System using a Cloud Blockchain Eco-System“

verfasst von / submitted by

Gerald Honegger, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr.Wolfgang Klas

Contents

List of Tables

List of Figures

Code Listings

Zusammenfassung

Für Systeme zur Unterstützung von Geschäftsentscheidungen ist es von Vorteil, die Implementierung eines manipulationssicheren, transparenten, konsistenten, deterministischen und datenschutzorientierten Online-Abstimmungs-Systems zu ermöglichen. In der Vergangenheit zögerten Regierungen, Institutionen und ihre Mitglieder selbst, Benutzern die Wahl über wichtige Entscheidungen in elektronischen Online-Abstimmungen zu ermöglichen.
In meiner Arbeit werde ich eine Organisationssoftware vorschlagen, die den Benutzern die Stimmenabgabe bei Entscheidungen sowie das Anmelden zu Veranstaltungen ermöglicht. Die verteilte Applikation verwaltet verschiedene Rollen für ihre Benutzer: Die Mitglieder des Systems können an Veranstaltungen teilnehmen und bei Abstimmungen für eine Entscheidung wählen. Organisatoren haben darüber hinaus zusätzlich die Fähigkeit, Abstimmungen und Veranstaltungen zu erstellen.
In der privaten Blockchain wird ein Mechanismus zum Zählen einer Stimme pro Ethereum-Brieftaschenadresse verwendet, wobei nur zuvor registrierte Benutzer Zugriff auf das Netzwerk erhalten. Auf diese Weise wird die Richtigkeit der Ergebnisse garantiert.
Cloud-Dienste werden unter anderem eingesetzt, um die Privatsphäre der Nutzer und deren Abstimmungsentscheidungen gegenüber Dritten zu gewährleisten. Die Kosten der Cloud-Service-Preise, die beim Ausführen der Blockchain in der Cloud anfallen, werden erfasst und analysiert. Mithilfe einer DApp-Implementierung in der Cloud kann ein sicheres und benutzerfreundliches Entscheidungssystem implementiert werden.

Abstract

Implementing a tamper-proof, transparent, consistent, deterministic and privacy-focused eVoting system is vital for Business Decision Support Systems. Historically, governments, institutions as well as their members themselves have been reluctant to allow users to take part in electronic, online decisions to vote for important decisions. In my thesis, we will propose an organisation software which will not only allow its users to cast their vote, but to also sign up for events. There are different roles of users handled by the distributed application. Organizers are able to create votings and events, as well as having the same functionalities of the members to vote and sign up to events. A mechanism of counting one vote per Ethereum wallet address will be used in the private blockchain, with only previously registered users receiving access to the network. This way, the correctness of the results will be guaranteed. Among other technical measures, cloud services are used to ensure the privacy of users and their voting decisions from third parties. Costs of the cloud service prices, which arise while running the blockchain in the cloud, are recorded and analyzed. Using a DApp implementation in the cloud, a secure and user-friendly decision-making system can be implemented.

1 Topic of the Master Thesis

Implementing and releasing a tamper-resistant electronic decision making system has always been difficult, and its results are often questioned. Organisations and governments have so far been reluctant on implementing eVoting systems and trusting them to handle important decisions. A promising technology that could ensure tamper-proofness and traceability of results is the distributed ledger technology of the blockchain. With a decentralized Ethereum application running in a cloud computing environment, users in an organisation can vote in elections and make other decisions. In order to correctly authenticate users, only credentialed participants do get access to an Ethereum wallet in the blockchain, which will enable them to cast one vote per election. Once approved, users can vote for administrators (and other positions which need to be filled in the organisation) or sign up for events. This way, the legitimacy of the results can be guaranteed by the consensus mechanism of the blockchain.
In my proposed Ethereum application, there are roles for organizers and members in a private Ethereum blockchain. Organizers can create elections or events. Members can take part in elections for candidates to take key positions in their organization, or sign up for an event. The organizers do have the full functionality of the members. This system must ensure a transparent environment where electronic votes are counted accurately, while at the same time guaranteeing the privacy of the user data, such as voting decisions or personal information, from third parties.
The proposed system will be a private blockchain running on the Azure cloud using the Azure Blockchain Workbench (ABW). The ABW runs on a private blockchain provided by Parity's Ethereum PoA template. The two most popular Ethereum clients are GetH and Parity. GetH is the more widely used client. However, according to [Rouhani and Deters (2017), p. 1], the Parity client does process transactions on average 89.8 percent faster than the GetH client. The ABW is used for the implementation of the organisation software described in this thesis. The functionality of the distributed application (DApp) runs on Ethereum Smart Contracts written in Solidity.

1.1 Comparison of Blockchain Platforms

A number of attributes can be compared for the different blockchain technology platforms (cf. table 1, [Rouhani and Deters (2019), p. 3]), such as consensus, whether the blockchain is public or not, if it offers Smart Contract (SC) support, if there is good documentation on the cloud technology support (brackets indicate that there are code examples, but no official documentation), the Smart Contract language(s) used, and the type of built-in cryptocurrency, if available. All of the platforms listed here do offer Smart Contract support, with Bitcoin only supporting it by attaching a sidechain to the main blockchain.

Ethereum uses the mechanisms Proof of Work as well as Proof of Stake, as can be seen in this table. It allows the development of both public and permis-

Platforms	Consensus	Public/Permissioned	Cloud Technology Support	Smart Contract Language	Built-in Cryptocurrency
Bitcoin	PoW	Public	Azure, AWS	Ivy, RSK, BitML	Yes (BTC)
Ethereum	PoW and PoS	Both	Azure, AWS, Google Cloud	Solidity, Flint, SCILLA	Yes (ETH)
Hyperledger Fabric	PBFT	Permissioned	Azure, AWS, Google Cloud	Go, Node.js, Java	No
Neo	dBFT	Both	(Azure, AWS)	C#, VB.Net, F#, Java, Kotlin, Python	Yes (Neo)
Nem	Proof of Importance	Both	-	Provide template	Yes (XEM)
Quorum	Raft-based and Istanbul BFT	Permissioned	Azure, (AWS), Google Cloud	Solidity	No
Cardano	PoS	Public	-	Plutus (Functional Language)	Yes (ADA)
EOS	DPoS	Public	-	C++	Yes(EOS)
R3 Corda	Flexible plugin feature for consensus	Permissioned	Azure, AWS, Google Cloud	Kotlin	No
Tendermint	BFT	Permissioned	Azure, AWS	Any Language	No
Waves	LPoS	Public	Azure, AWS	RIDE	Yes (Waves)

Table 1: Blockchain Platform Comparison

sioned blockchain applications. One thing that is not evident from this table, is that Ethereum is likely the most widely used[1] Smart Contract blockchain technology. The cloud service providers Azure, Amazon Web Services (AWS) and Google Cloud do support Ethereum and offer documentation on it.
Bitcoin was not made to run decentralized applications on it, and the other platforms listed are not as well known and not as widely adopted as Ethereum, the de-facto standard Smart Contract platform. Because of these reasons, Ethereum is the blockchain technology used for the implementation of this thesis.

2 Terms and definitions

In this chapter, we are explaining important terms and we are giving definitions which are necessary to understand the approach taken to create this Business Decision Support System (BDSS) implementation. The focus is on concepts used in this thesis, as well as concepts related to them.

2.1 Smart Contracts

The first blockchain concepts, most famously Bitcoin, made a distributed cryptocurrency possible which is not controlled by a central authority and which can not be copied or replicated. In 2015[2], Vitalik Buterin has advanced this concept by introducing a blockchain network protocol which does not only provide a cryptocurrency, but which can run applications in a distributed way. With the open-source, distributed platform Ethereum and its Smart Contract language

[1] This is partly subjective, and can change in only a few years

[2] He first described the concept in 2013 and it was presented in the North American Bitcoin Conference in January 2014

Solidity, multiple ledgers can validate whether a process has terminated correctly.
This technology can be used to automate processes which would otherwise be done by a human actor, thus reducing the operating costs of companies. For example, an airline could automatically refund ticket prices to customers after an oracle (an independent entity which provides information to Smart Contracts) detects that a plane flight was cancelled.

2.2 Blockchain Technology Platforms

2.2.1 Ethereum

Ethereum is a global, open sourced and decentralized platform that uses Ether as cryptocurrency and gas to make transactions. An advantage the Ethereum protocol has over other blockchain cryptocurrencies is that it can be used to run decentralized applications (DApps) on its blockchain. Apart from using a public blockchain, using the global Ethereum network, it is also possible to implement DApps on a private blockchain in the cloud, which we have done for the BDSS application described in this thesis.

2.2.2 Quorum

The Quorum blockchain is a soft fork of Ethereum which is being developed by JP Morgan and is designed for use in companies. The basic structure of Quorum was developed in cooperation with the Ethereum Enterprise Alliance and Microsoft on the basis of Go Ethereum, the basic code of the Ethereum platform. The aim of the blockchain infrastructure is to process payment transactions in a company in a transparent and tamper-proof manner. Quorum is the protocol used in the ABW.

2.3 Type of Blockchain

2.3.1 Private Blockchain

In contrast to public blockchains, which are decentralized and don't have a single authority over the network, a private blockchain is controlled by an organization which has the sole authority over read and write permissions. The disadvantage of this approach is that such a blockchain is not or less decentralized, while its advantages are that it is easier scalable and can offer data protection.

2.3.2 Permissioned or Consortium Blockchain

A permissioned (also called consortium blockchain) is a private blockchain which is or can be owned by multiple organisations. Users with a permission, which was granted by the consortium, can read information of and write blocks in the blockchain.

2.4 Blockchain in the Cloud / Blockchain as a Service (BaaS)

Multiple large tech companies have begun offering blockchain in the cloud services - also known as Blockchain as a Service (BaaS), including, but not limited to Microsoft, Amazon, Google, SAP, Oracle and IBM. Demand for such services comes mainly from businesses which want to experiment with blockchain technologies. Use cases for this technology are for example Supply-Chain-Tracing-Systems, where suppliers want to make supply chains more transparent and guarantee the authenticity of products with simple means.

2.4.1 Azure Cloud

Microsoft Azure is a highly scalable cloud computing platform from Microsoft which offers cloud services and is primarily aimed at companies and software developers. The cloud computing environment's performance and range of services have continuously expanded since its official release in 2010.
Cloud service competitors of and alternatives to Microsoft Azure are Amazon AWS, Google Cloud, Oracle Cloud, CloudVPS, DigitalOcean, SAP Cloud Platform, Open Telekom Cloud and more.

2.4.2 Azure Blockchain Workbench

The Azure Blockchain Workbench (ABW) is a cloud environment offered by Microsoft which can run DApps and uses the Quorum protocol. It allows developers to host Solidity contracts in the blockchain on the Azure cloud and makes versioning of Smart Contracts as well as handling user authorization of the DApps more practical. Being a permissioned blockchain, the ABW increases the level of privacy of user data since it can not be accessed by third parties outside of the system. Authentication of the system is managed by Administrators of the Azure Service, who are able to add or remove users to different Smart Contracts, as well as giving them roles in the system.

The ABW can be integrated with several other Azure components as can be seen on the description of the ABW architecture (see Fig. 1). Users are identified by their Azure Active Directory identity, and a new identity is automatically generated for on-chain usage when adding a new user to the ABW contracts. Data is stored and can be queried in the off-chain Azure SQL Database and visualized using Power BI. Smart Contracts and their associated metadata is stored in the Azure Storage. Documents or media can be stored there, while their hash gets stored on-chain. Monitoring of the application messages, such as warnings, errors and success messages for debugging purposes can be done with Application Insights and Azure Monitor. Besides the web app, there is also Xamarin code available to set up an Android or iOS app which can access

[4]https://docs.microsoft.com/en-us/azure/blockchain/workbench/architecture, retrieved on August 20th 2020

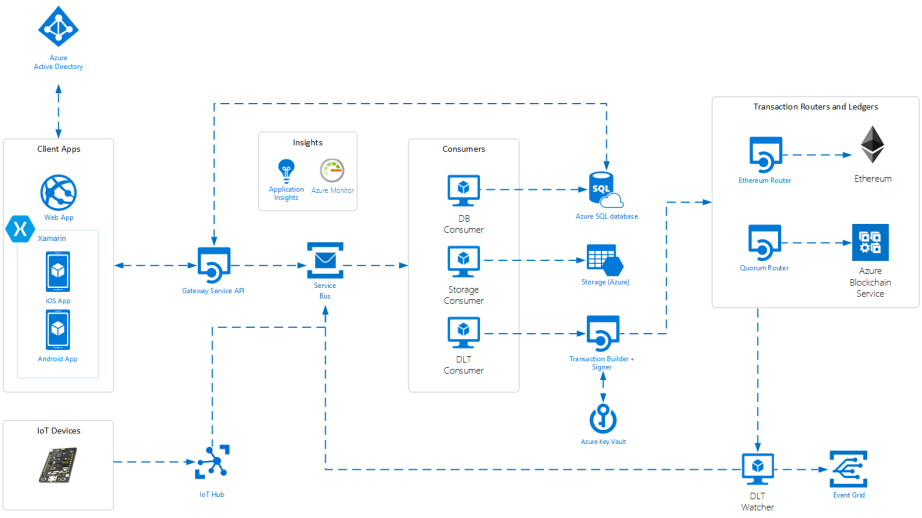

Figure 1: The architecture of the ABW as described in the official documentation [4]

the ABW website of the application [5].
Advantages of running a blockchain network in the Azure cloud include Azure Monitor logging and alerts of the node health, data encryption in transit and at rest, automatical node storage increase if the ledger size increases, ledger backup and node recovery, automatical updates and patch releases of Quorum, and high availability of the service with a 99.9% SLA with the standard tier of the Azure Blockchain Service, which is used for the ABW.
It should also be noted that it is planned that users can run their own nodes of an ABW network outside of the cloud. This way, not all nodes have to run in the cloud, and with more than 50% of all nodes running independently and managed by different parties outside of the cloud, the level of centralization could be reduced drastically.

2.5 Consensus Algorithms

2.5.1 Proof of Work (PoW)

Invented in 1993 as an algorithm to prevent DOS attacks or SPAM in networks and getting its name in 1997, the Proof of Work algorithm only got well known once it was implemented in the Bitcoin blockchain technology in 2009 to handle the mining of new blocks and validation of transactions. This is vital for keeping the Bitcoin network up, since new transactions constantly need to be validated,

[5] see https://github.com/Azure-Samples/blockchain-devkit/tree/master/connect/mobile/blockchain-workbench/workbench-client, retrieved on August 20th 2020

and miners are getting a reward (in the form of a new block, which is mined at random intervals) for doing so.
It allows for a decentralized blockchain network to exist without a central authority, but its disadvantages are that it is quite costly in that it needs a lot of electricity (to run miners) and processing power to keep the network running. To be more specific on the electricity usage of the Bitcoin network, it is "now on pace to use just over 42TWh of electricity in a year, placing it ahead of New Zealand and Hungary and just behind Peru, according to estimates from Digiconomist."[6]. Also, faster validation times may be required for business use cases, since a bitcoin transaction can take about ten minutes to be confirmed. Because of those reasons, developers and companies were looking for more efficient alternatives to this algorithm.

2.5.2 Proof of Stake (PoS)

The Proof of Stake algorithm does favor rich users who own a larger share of the cryptocurrency than other users. They do get a higher chance to be chosen to forge the next block in the network. In theory, this makes PoS more unequal than PoW; however with this algorithm, participants need no expensive mining equipment to take part in the blockchain, which makes it less expensive to set up new nodes. If users validate fraudulent transactions, then they are punished by losing a part of their stake.
A weak spot of blockchain technologies is that an attacker can take over the network by owning the majority of its computing power (the networks mining hash rate). This is called a 51% attack, and it allows a malicious user to prevent other user's transactions, stop miners from completing new blocks or revert transactions, allowing him to double-spend coins. Proof of Stake makes such a 51% attack less likely since a large amount of money is required for it to succeed - for example, if Bitcoin used Proof of Stake, then owning more than $111 billion USD in Bitcoin (half of the entire Market Cap as the time of writing) would be necessary to run a successful attack. [7]. In the actual Bitcoin network, it is sufficient to have the majority of the mining power to run such an attack successfully. That being said, concerning the Bitcoin network, this is only a hypothetical scenario since such an attack has not taken place.
The current version of Ethereum uses Casper, which is an implementation that has the objective of making Ethereum, which was started as a PoW blockchain in 2015, a PoS-protocol cryptocurrency. This will have the advantage of reducing the energy necessary for running the network by making mining unnecessary and increase its security by removing malicious validator nodes faster.
Other examples of Proof of Stake cryptocurrencies are peercoins, LSK and NXT, and Ouroboros by Cardano, which calls itself the "first provably secure proof-

[6] https://www.theguardian.com/technology/2018/jan/17/bitcoin-electricity-usage-huge-climate-cryptocurrency, published on January 17th 2018, last modified on July 3rd 2019, retrieved on August 18th, 2020

[7] https://www.coindesk.com/price/bitcoin: Market Cap $221.60B. Retrieved on August 18th 2020

of-stake protocol"[8].

2.5.3 Proof of Authority (PoA)

PoS does still leave a risk that richer users will control the network, which may not be avoidable using the algorithm for a blockchain network. Because of that, the reputation-based Proof of Authority consensus algorithm was proposed in 2017 by Ethereum co-founder Gavin Wood.
It does continue the basic idea of the PoS algorithm and runs without miners which are required in PoW. It is especially efficient and practical for private blockchain networks.
A group of validator nodes do maintain the network in this reputation based consensus algorithm. Here, the base of the stake a user gets in the network is not the amount of (crypto-)coins he/she has, but his/her own reputation. Blockchains using PoA are secured by arbitrarily selected validation nodes, which are the trusted entities of the network. It is a highly scalable mechanism since only a small number of block validators is required to keep the network up. Previously approved participants can verify transactions and blocks.
This algorithm is especially demanded by corporate customers, such as companies dealing with logistic applications. A typical example where a PoA blockchain solutions can efficiently be used are supply chain processes. It allows for companies to guarantee their privacy while using blockchain applications, which is important to prevent industry espionage, and is often times required by law when dealing with sensitive user data. In the ABW, PoA is used as the consensus mechanism to set up a private network. This way, mining is unnecessary and a cryptocurrency like the ether gas and its associated costs are avoided.

2.6 Byzantine Fault Tolerance (BFT)

The Byzantine Fault Tolerance is a universal solution to a specific problem of decentralized systems: A node in a network can give false information while claiming to be a trusted actor. BFT deals with the problem that some nodes can be malfunctioning or malicious and ensures that the blockchain network will continue to work with a certain level of byzantine fault tolerance. This algorithm makes faulty transactions impossible and avoids the double spending problem of cryptocurrencies. BFT works by deciding when a hash is being confirmed as finalized (finality rules) and ensuring that nodes which misbehave multiple times e.g. by approving malicious transactions will be removed from the network (slashing rules).
Bitcoins PoW was the first blockchain-based probabilistic solution to the Byzantine Fault problem. Apart from blockchain implementation, the BFA is being used in every system which deals with a lot of information from a lot of different sensors and acts based on that information, such as aircraft electronics, nuclear power plant systems, and more.

[8]https://cardano.org/ouroboros/ Retrieved on August 18th 2020

3 Related Literature

In this chapter, papers relevant to the research for the topic of my thesis are discussed. It covers challenges and issues related to privacy, security, user authentication, distributed ledger technology running on cloud computing services, cost analysis of Ethereum transactions and more.

3.1 Defining and Delimitating Distributed Ledger Technology

Lange *et al.* offers definitions and core characteristics of distributed ledger technology (DLT) and the blockchain technology. While both are seen as synonymous, the terms are analyzed and a differentiation and characterization of both technologies is derived. One key difference is the state of the synchronization - in some DLTs, not all nodes need the entire information of the network, and there are DLTs where "only nodes that are associated with a transaction see, verify and store the data of the transaction". [Lange et al. (2019), p. 46] Also, some Blockchain core characteristics, like blocked chains, are only optional features for DLTs. [Lange et al. (2019), p. 48]

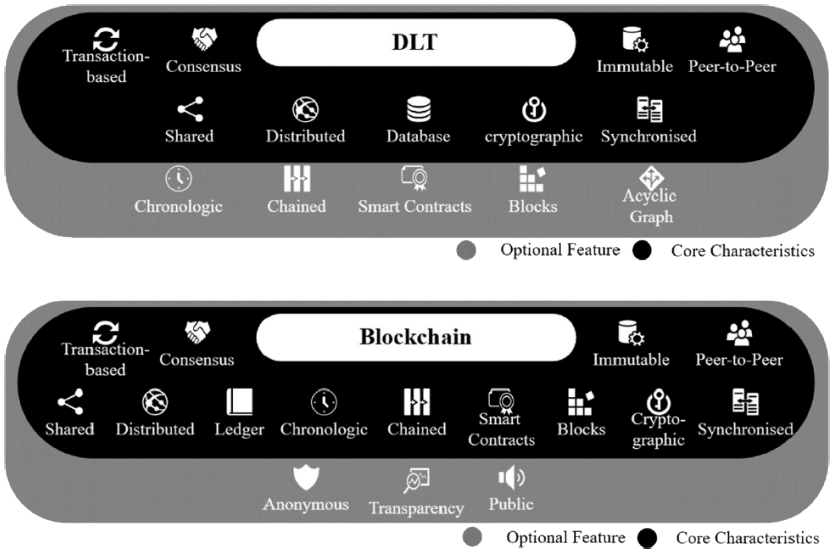

Figure 2: Characteristics of DLT and Blockchain based on the quantitative analysis

The core characteristics of both technologies are visualized in Fig. 2 [Lange et al. (2019), p. 48], which is based on a quantitative analysis. Terms which are more strongly associated with the technology and are more often mentioned

alongside of it are shown in the graphic. Nearly all DLT core characteristics are also blockchain core characteristics. [Lange et al. (2019), p. 48]

3.2 Mitigating Data Tampering Security Risks

Among the advantages of the blockchain technology are its traceability, integrity and tamper-proofness. The management and mitigation of tamper-risk in a proper way is the focus of this paper. A countermeasure architecture based on Ethereum is introduced (cf. Fig. 3, [Iqbal and Matulevičius (2019), p. 21]), which ensures that transaction data is validated and encrypted data is stored on an immutable ledger. Also, the information is obfuscated by splitting the data and storing it in random locations.[Iqbal and Matulevičius (2019)]

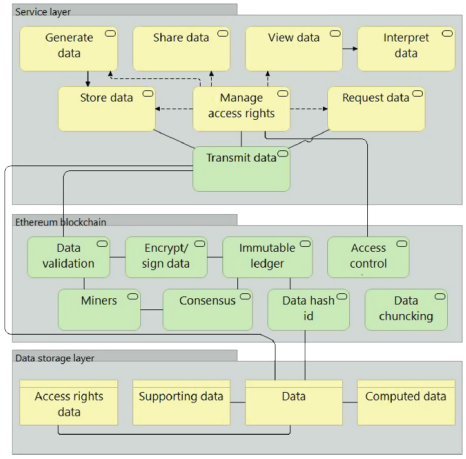

Figure 3: Ethereum-based countermeasure architecture

Iqbal *et al.* concludes that apart "from the tampering risk, blockchain-based applications could help mitigating other security risks, like DoS/DDoS attack, MitM attack, side-channel attack and etc."[Iqbal and Matulevičius (2019), p.26]. On the other hand, more blockchain-specific security risks such as a sybil attack, double spending attack or a 51% attack need to be considered when planning to build a blockchain-based application. [Iqbal and Matulevičius (2019), p.26]

3.3 An Identity-Protecting License Trading Platform

The organization software proposed in this thesis is a privacy-critical blockchain application. Privacy and Security issues will be a main concern of it. In [Kakarott et al. (2019)], an Intellectual Property License Trading Platform is presented, which handles sensitive user data. Identities and License information needs to be secured to ensure that an observer can not attain the information

without permission. Once a trade is completed, the information about the license must be shared with the trading partner. The proposed platform-concept also reduces transaction costs.

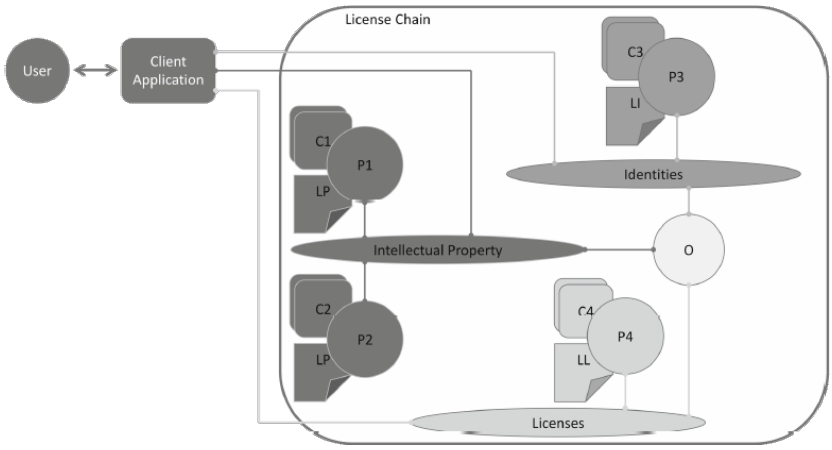

Figure 4: Hyperledger Fabric based model of the License Chain Concept. P: Peer, C: Chaincode, L: Ledger, O: Ordering Node

Fig. 4 [Kakarott et al. (2019), p.33] shows how a user in this Hyperledger-based system can order a licensing contract from an Intellectual Property owner. Due to the platform, a certain level of loss of privacy exists, but it is limited by introducing trusted partners, which will match potential buyers with owners of Intellectual Property. There remains a single point of attack for this system: Trusted partners need to behave correctly and need to be vetted before being accepted into this position.
A 51% attack can be avoided in this system by only allowing known partners to participate in the consensus process as network peers. This way, the users of this system know each other and have less of an incentive to manipulate it.[Kakarott et al. (2019)]

3.4 Blockchain-based Multi-party Business Process Monitoring

Trusted data exchange as well as minimizing the risk of a fraudulent organization altering monitoring data are an important security concern for Meroni *et al.* This can be achieved by combining blockchain technology with artifact-driven monitoring, as is explained in [Meroni et al. (2019)]. This monitoring system enables the organization to know how its processes are being performed. The monitoring information is stored in and forwarded by a blockchain.

Based on an artifact-driven monitoring platform architecture (cf. Fig. 5)

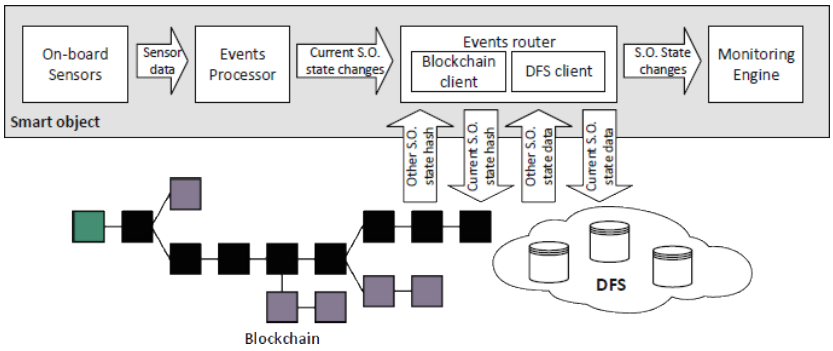

Figure 5: Architecture of the Distributed File System (DFS)-blockchain hybrid platform

[Meroni et al. (2019), p.60], in this system, on-board sensors collect information about smart objects, which is then handled in the Events Processor. The Monitoring Engine detects violations in the system and when activities are executed. The most interesting part of the system is the Events Router, which is integrated with a blockchain client. The client initiates a new transaction when it is called and sends notifications to the monitoring system whenever a block of information is added to the blockchain.

3.5 Balancing Privity and Enforceability

The amount of information shared in an organisational system should be limited to a minimum - this is the privity objective of the implementation. Only the parties of a contract should have access to the information. Both factors conflict with each other, and trade-offs between privity and online enforcability need to be considered in the development of a blockchain application that handles sensitive data. One possible security measure explained in Köpke *et al.* is to encrypt part of the data used in Business Process Management (BPM)-Based Smart Contracts. Encrypting the data does have the disadvantage of limiting enforcability. The same key exchanges mechanisms that enable the encryption and decryption of blockchain data does also work for off-chain data, with the keys being shared on-chain. [Köpke et al. (2019)]

3.6 Executing Collaborative Decisions Confidentially

Through the use of blockchain technology, data can be stored and decision logic can be executed in a transparent and tamper-proof manner. This way, collaborative decision making can be improved. In Haarmann *et al.*, an approach to hide sensitive data on the blockchain is proposed. In this system, a blockchain-based conflict can be solved by revealing the decision without requiring a third party. Malicious behaviour needs to be discouraged in the blockchain through the use of an incentive model for this mechanism to work. One drawback of this system

is that it slows down the automated decision making time.

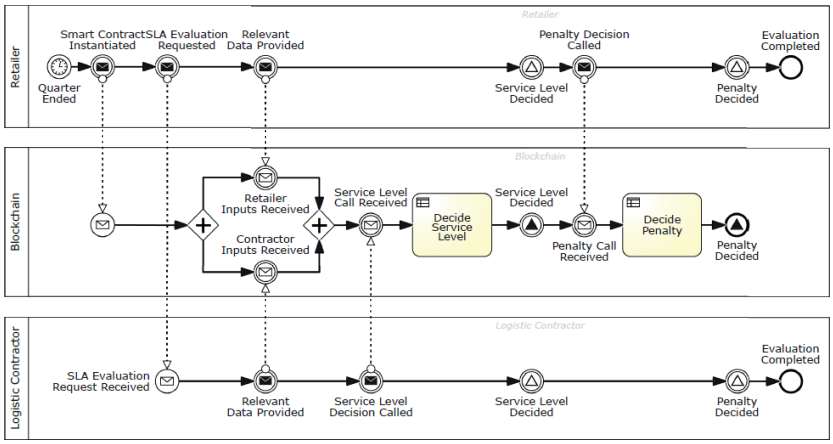

Figure 6: Collaborative decision making by storing and executing decisions (logic and instance data) on a blockchain

Fig. 6 [Haarmann et al. (2019), p. 123] shows how Retailer, Blockchain and Logistic Contractor interact with each other and how a participant can be penalized for revealing sensitive data. [Haarmann et al. (2019)]

3.7 Performance and Scalability

Smart Contract technologies such as Ethereum can combine the versatility of software with cryptographically secure blockchains. Parameters of private Ethereum blockchains can be altered to optimize the transaction speed of the system. Examples for such blockchain-specific parameters are the "time passing between two consecutive blocks, the size of blocks, the hardware of the nodes running the blockchain software, or simply the size of the network." [Schäffer et al. (2019), page 103]
According of an analysis of an amount of transactions (on a private blockchain) that equals the amount of eight days of transactions on the Ethereum main chain, changing one of these parameters does majorly impact the configuration of the other parameters since they are intertwined with each other. Block frequency and transaction signing were identified as the two biggest bottlenecks in the analysis. [Schäffer et al. (2019), page 113]

3.8 Extracting Process Mining Data from Blockchain Applications

Users of my proposed organisation software will need to have an overview over its data and processes, which will require functionality to achieve this goal.

Since extracting and analyzing blockchain data is difficult, it is rarely used for process mining.

A framework is presented in [Klinkmüller et al. (2019)], which consists of "a

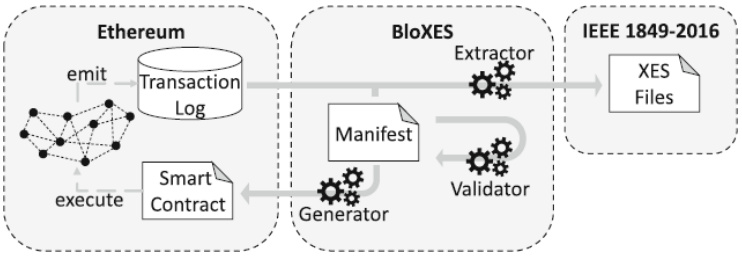

Figure 7: High-level overview of the components

manifest specifying how data is logged, an extractor for retrieving data (structured according to the XES standard), and a generator that produces logging code"[Klinkmüller et al. (2019), p. 71] The high-level overview of the framework in Fig. 7 [Klinkmüller et al. (2019), p. 75] shows how event-data from the log is extracted. The on-chain Ethereum transaction logging used in this paper could be useful in developing user-defined DApps. Cloud services supporting blockchain technologies and DApps provide a practical overview of its processed data, which is further explained in chapter 6.2.5.

3.9 Security, Performance, and Applications of Smart Contracts

Rouhani *et al.* covers a wide and general variety of key concepts of the Ethereum technology, such as security tools, performance improvement approaches and decentralized applications based on Smart Contracts. It presents a "systematic review on the Smart Contract history, supporting platforms, programming languages, security, performance, and decentralized applications." [Rouhani and Deters (2019), p. 18]. The primary focus of decentralized applications (in the field of healthcare and IoT) have been identified to be record keeping, access control management and sharing resources. Performance improvement mechanisms presented are the use of a lighter consensus mechanism and running transactions concurrently.

3.10 Performance Analysis of Ethereum Transactions

Ethereums clients run on different speeds and with a different performance. Choosing the better performing client will have an impact on the overall distributed ledger system. The transaction speed of the two most popular Ethereum clients, GetH and Parity, is compared in [Rouhani and Deters (2017)] in

a private blockchain.

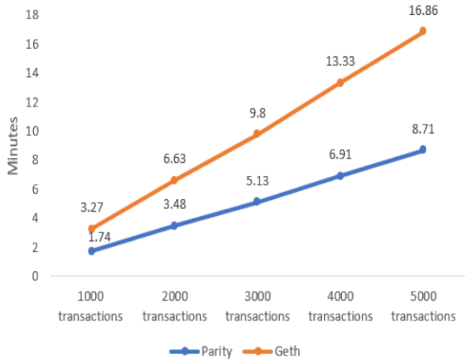

Figure 8: Total time in minutes for processing different amounts of transactions by different clients

In Fig. 8 [Rouhani and Deters (2017), p.73], we can see that the GetH client is slower than Parity when running 1000 to 5000 transactions. Judging from these results, it can generally be said that Parity is a faster client in processing Ethereum transactions.

3.11 Ethereum Transaction Fees

The paper by Pierro *et al.* discusses the various factors which determine the price of Gas that an Ethereum transaction costs. A transaction will only get approved by miners once the price is paid to them. This paper concludes that only a few factors do have a major impact on the Gas price, which are mainly the number of miners and pending transactions. [Pierro and Rocha (2019)]

3.12 Optimized Execution of Business Processes on Blockchain

García-Bañuelos *et al.* covers collaborative business processes running on Smart Contracts on the blockchain. More specifically, it proposes to optimize the cost per transaction. The areas targeted by this optimization strategy are the initialization cost, the task execution cost as well as improved runtime components. This way, gas consumption is minimized in a Solidity Smart Contract which is created by compiling a Business Process Model and Notation (BPMN) process model. [García-Bañuelos et al. (2017)]

3.13 A Cloud Data Movement Policy Architecture

Using Smart Contract technologies to improve the level of trust users have using cloud services is a benefit my proposed implementation aims to offer. This paper

explains how trust in the cloud is negatively affected by an "outdated Service Level Agreement (SLA) model, untrusted third parties with access to our data, unknown data location, and unwanted data movement" [Kirkman and Newman (2018), p. 1]. Similar to my implementation, it proposes to use the Ethereum blockchain to store data and concludes that decentralization can solve the trust issue users have with the application.

3.14 Using Smart Contracts for Cloud Tenant Management

An application to manage tenant and service accounts is presented by Nayak *et al.* The authors explain how private and public key pairs are generated by clients to handle authentication and identity management in an application based on the blockchain system Quorum. The paper argues that trust and transparency can be increased by offering "security, non-repudiation, tamper-resistance, and easy transaction history access" [Nayak et al. (2018), p. 1] to users through the use of blockchain technology, and mentions increased scalability and resilience as additional benefits of the technology.

3.15 Decentralized Voting Platform Based on Ethereum Blockchain

Khoury *et al.* propose a voting mechanism without the need of a trusted authority running a third party server. Using a decentralized, trustless voting platform, users can cast their vote using their mobile phone numbers as a way of authentication, while their privacy is guaranteed by the application running on the Ethereum Virtual Machine.

In Fig. 9 [Khoury et al. (2018), p.4], we can see how the voting contract checks if a user is eligible to vote and if he/she has already taken part in the election. Multiple votes per phone number are restricted in this blockchain-based system, making the presented platform a practical way of ensuring the authenticity of voting participants in the system. [Khoury et al. (2018)]

3.16 Towards Secure Ethereum-Based E-Voting

Yavuz *et al.* cover the safety, transparency and ease of use of eVoting-Systems. The presented application uses the Ethereum wallets (or an Android device for users without an Ethereum wallet) and the Solidity Smart Contract language to ensure one vote per user in a transparent system which allows for user privacy. The records of ballots and votes is held in the Ethereum blockchain after an election takes place. [Yavuz et al. (2018)]

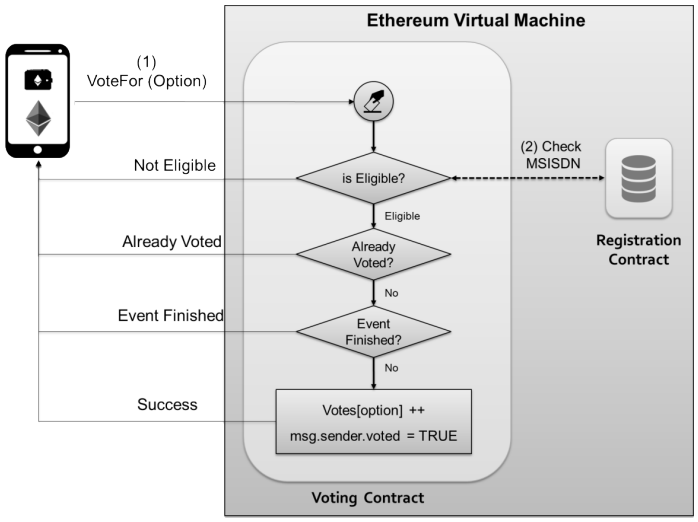

Figure 9: A voting mechanism aided by an Ethereum Virtual Machine

3.17 A Decentralized Ethereum-Based Marketplace Application

Blockchain applications can increase the level of users privacy, reduce cost in the fees users would have to pay in centralized systems, and remove the ability to arbitrarily block users from a service. In Ranganthan *et al.*, the Truffle development framework is used to create an Ethereum blockchain platform to form a decentralized marketplace application, whose functions are contained in a Smart Contract. The paper concludes that the average transaction runtime in the application is 3.8 seconds, and the cost of transactions is cheaper than both online (e.g. EBay) as well as offline (e.g. Sotheby's) marketplaces, according to the cost analysis presented. [Ranganthan et al. (2018)]

3.18 Attack and Defence of Ethereum Remote APIs

Faulty configuration and usage of blockchain technology can introduce vulnerabilities and reduce the security of applications using the technology. Countermeasures to possible attacks on the Go-version of the Ethereum client are presented in Wang *et al.* Encrypting traffic, adding access control, separating transaction signing and forwarding, using multi-signature technology and traditional security solutions (such as firewalls and intrusion detection systems) can be used to increase the security of Ethereum applications. [Wang et al. (2018)]

3.19 Blockchain for Trustworthy Coordination

Ciatto *et al.* describes how fully-decentralised, trustworthy coordination can potentially be realised using Smart Contracts to create a multi-agent system (MAS). The Ethereum technology is mapped onto the LINDA tuple-based coordination model, and two proof-of-concept implementations are being discussed. The cost of execution and the handling of multiple control flows are issues which are discussed in the paper. [Ciatto et al. (2018)]

3.20 Caterpillar: A Blockchain-Based BPMS

Running on top of an Ethereum blockchain, the Business Process Management System (BPMS) presented by Pintado *et al.* is a prototype that makes the creation of process model instances possible. Also, the state of process instances can be tracked and process tasks can be executed in this application. The Smart Contracts of the Caterpillar BPMS are generated by a BPMS-to-Solidity compiler and the blockchain stores its process instance states.[Pintado et al. (2017)]

3.21 An Ethereum-Based Smart Transportation System

A smart transportation system using DLT is proposed by Zichichi *et al.* In this infrastructure system, sensor-generated data is shared by users. To ensure the privacy of the participating users, Zero Knowledge Proof is used. In this application, data access and authorization are guaranteed by Ethereum Smart Contracts. [Zichichi et al. (2019)]

3.22 An Ethereum-Based Cloud User Identity Management Protocol

Guaranteeing user identity authorization without relying on third party services is important to ensure a user's trust in DApps. In Wang *et al.*, an Ethereum-based cloud user identity protocol is presented, in which a JSON Web Token is used in OAuth 2.0. Using this system enables an identity management without requiring a third party to authenticate users. The paper concludes that the Ethereum-based Identity Management (EIDM) described in the paper offers more diversified security guarantees than the already established Consolidated Identity Management (CIDM). It also improves practicability and flexibility according to the performance evaluation results. [Wang et al. (2019a)]

3.23 A Blockchain-Based Access Control System for Cloud Storage

Sukhodolskiy *et al.* discusses how information can be securely shared in an untrusted cloud environment. Using an Ethereum-based application prototype, the implementation of security measures such as logging of all important events

and ensuring privacy by only transferring the ciphertext of hash codes through the blockchain ledger is demonstrated. Advantages of the application are the integrity of the information about transactions, the impossibility to edit data and the ability to easily customize the access policy for the encrypted data. [Sukhodolskiy and Zapechnikov (2018)]

3.24 A Secure Cloud Storage Framework With Access Control

Handling sensible information in decentralized cloud services is also discussed in Wang *et al.* Here, a framework of Ethereum blockchain and ciphertext-policy attribute-based encryption is proposed in order to securely manage cloud storage and access control. The owner of data in this system can set access periods in which ciphertext of his/her data can be encrypted and shared with other participants in the system. [Wang et al. (2019b)]

3.25 Toward High-Performance Permissioned Blockchain

Excessive CPU scheduling, inefficient block broadcast and high latency of initial blocks synchronization lead to issues with the transaction throughput and scalability of permissioned blockchains. Because of this, Huang *et al.* propose an RDMA-based (Remote Direct Memory Access) permissioned blockchain framework called BoR (Blockchain over RDMA) to reduce the severity of these problems. It redesigns the block synchronization protocol and leads to a higher throughput, less CPU usage and lower latency. [Huang et al. (2019)]

3.26 Decision Support Systems - Definition and Implementations

Decision Support Systems (DSS) are software systems which can identify, process, compile, and support the evaluation of information relevant for human decision makers. This information can then be used for operational and strategic tasks. These include functions for sorting and filtering data, their flexible display and evaluation options, e.g. totals and average calculations, comparisons, and more. Also, DSS can be used to make forecasts or create scenarios based on the given information.

In the following paragraphs, we will give examples of DSS which were proposed or successfully implemented in different sectors of different industries.

The paper "Decision Support System for the Agri-food Sector – The Sousacamp Group Case" by Branco *et al.* describes a business group that had their decision making accuracy as well as their performance levels increased after implementing a Decision Support system which was implemented and tested in a real

environment. This was done by integrating IT/IS into the agri-food organizations business and operational processes. [Branco et al. (2015), p. 1]

In a review of DSS and applications in Ophthalmology[9], it is stated that Decision Support Systems in mobile apps can help doctors diagnose patients quicker and more accurately as well as make it easier to find the accurate way of handling the illness. This is necessary because the amount of information can be too large to be handled by a single human in a limited amount of time. This technology could be especially useful in developing countries as well as remote areas. The paper concludes that at the time of writing, very few such apps existed in the different mobile app stores. [de la Torre-Díez et al. (2014), p. 1]

The paper "Application Research of an Intelligent Decision Support System Based on Data Warehousing Technology" describes the integration of a Data Warehouse with DSS technology in the area of teaching management. Here, a DSS is used to solve decision-making problems. The system analyses a large amount of data and provides information to project managers to help them make decisions to improve the process performance. [Wang (2010), p. 4]

4 Focus and Problem addressed by this Thesis

A major problem with electronic decision-making systems (e.g. eVoting) is that users do not trust its results and organizations/governments are only reluctantly implementing these kinds of systems. The main topic and context of the task is the conception and implementation of a decentralized Business Decision Support System (BDSS) which allows authorized users to make decisions in a trusted environment. The aim of this blockchain-based system is to create an application which can be used to make tamper-proof decisions in organizations. Such a BDSS must be fast, reasonably cheap and robust against malicious attacks. Especially the last point is an advantage which distributed technologies have over a more traditional, central approach, such as a server-client architecture.
The implementation will be done in the form of a DApp and its functionality is executed in Smart Contracts. The focus of this thesis is on creating a properly working BDSS focusing on dedicated functionality which helps users to make trusted decisions in an organization by providing distributed software in the form of a DApp that can not be tampered with.

4.1 Scope and Requirements of the Implementation

Problems, challenges, and goals discussed in the following paragraphs and its implementation include allowing users to take part in different kinds of votings in the course of decision making (different possible configurations are ex-

[9] The field of medicine concerned with the treatment of eye disorders

plained in chapter 6 of this thesis). There will be comprehensive, sustainable and tamper-proof documentation of the decision making process. The user's privacy is preserved by decision-related personal information being only visible in a closed-off private blockchain in a cloud environment.
Users need to be correctly authenticated by the system in order to guarantee one vote per authenticated person. Organizer's rights must be managed, which will be done using roles in the ABW.

A private, permissioned blockchain system on the Azure cloud will be used to implement the BDSS system. This leads to an increased level of data security. On the other hand, this does have the disadvantage of making the system more centralized (compared to a public, off-cloud blockchain) since users need to be authenticated in order to take part in the organization and its decision making processes.

4.2 Problem Description

Online participation in decision making processes is a central part both of BDSS's as well as of eGovernance applications. Web-based applications and processes which let citizens participate remotely in political events have been tried and tested for a long time, but they did - despite many prototypes and test runs - not become mainstream and accepted by most of society. Over the last twenty years, eVoting technology has rarely been used in a real world environment, despite the long developmental history of eVoting processes.

4.2.1 Practical eVoting Implementations

An eVoting prototype, which was tested in New Zealand in 2001, was shown strong support by its users. The paper suggests that this technology can lead to more people taking part in political processes. Some advantages of the new technology were also identified, such as a shorter time of processing the votes, reducing the need for voting booths and staff, lowering the number of incorrect votes (e.g. by the system not allowing the user to check two candidates at the same time), a lower cost of the election and the system eliminating human administrative errors.[Carter and Fielden (2001a), p. 306]
Despite this, as the paper stated at the time that "the ballot box is one of the last bastions of the pre-digital information era"[Carter and Fielden (2001b), p. 286], which does still hold true to this day.
In 2002, eVoting was thought to have the potential of being a "new 'killer application' for the Internet and a suitable authentication tool for voting security"[Watson and Cordonnier (2002), p. 1]. Back then, smart cards, cryptography and Internet technologies were used to confirm the identity of voters. Those applications had a classical Client-Server architecture, which does have its own flaws - the main problem being that it requires the user to trust one single authority, which controls the voting server(s) - which may be overcome

by blockchain technologies supporting multiple ledgers.
The main security vulnerabilities of an electronic decision making tool were stated to be data leaking, data manipulation, and program manipulation in [Beckert and Beuster (2006), p. 68] in 2006. The confidentiality, integrity and availability of an application of this type must be guaranteed - the service and resulting data must be available and understandable to its users, and unauthorized third parties must not access it. Using distributed ledger technologies in a permissioned blockchain, a BDSS which can not be viewed or manipulated by an attacker can be created.
The lack of users trusting systems which enable online decision making was the main focus of [Antoniou et al. (2007)] in 2007. This was realized by creating a more transparently designed and built system, and more specifically, by implementing its three components, which are "the decomposition of eVoting systems into “layers of trust” for reducing the complexity of managing trust issues in smaller manageable layers, the application of a risk analysis methodology able to identify and document security critical aspects of the eVoting system, and a cryptographically secure eVoting protocol"[Antoniou et al. (2007), p. 378]. This framework, which verifies its security at each layer, can lead to an increased level of perceived security by its users. It does however not resolve non- technological issues, such as the voters right to verify that their vote was actually counted by the system[Antoniou et al. (2007), p. 387]. This problem can be resolved in a private blockchain which is closed-off to the public, but keeps results and decisions transparent to the users of an organisation.
In Fig. 10 [Mei (2009), p. 169] [Meier (2012), p. 149], we can see eDemocracy

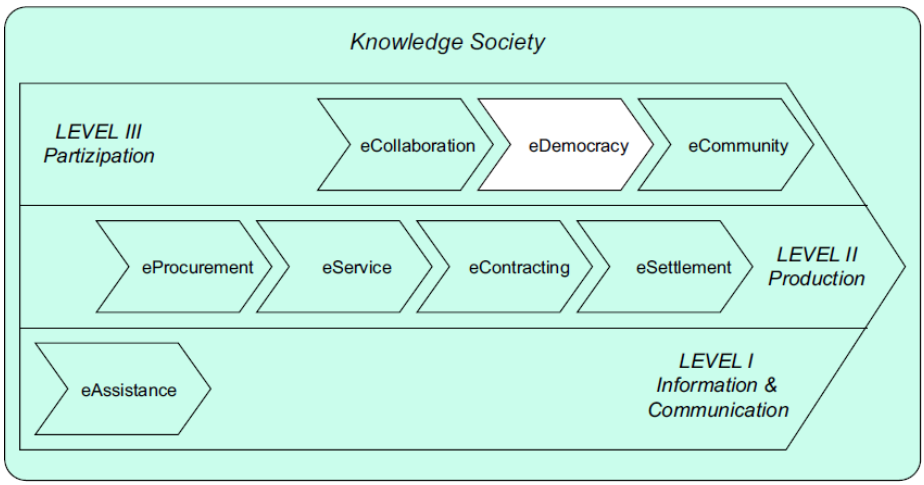

Figure 10: Knowledge Society

being, alongside with eCollaboration and eCommunity, a part of eParticipation. Looking at this summary of the knowledge society, the viewer of the figure may

notice that most of its parts and levels - including information, communication and production, have been successfully implemented over the last two to three decades. There is also a large amount of eCollaboration tools available today, as well as eCommunities (e.g. social media), but eDemocracy remains an application which is rarely used in formal decision making processes. Without the use of computer-based expert systems and knowledge, an information and knowledge based society cannot thrive.
Meier *et al.* notes in 2009 that most European countries have already made experiences with electronic decision making in eGovernment applications. Possible improvements which eVoting can provide were seen in a better documentation of the entire decision making process - before, while, and after the voting takes place. On an eGovernmental portal (or in the case of my implementation, the portal of an organisation), the information about candidates or decisions which users can decide upon can be arbitrarily detailed, including links to information pages.
An increased level of mobility is another advantage which electronic decision making can provide, and which benefits disabled persons in particular. Also, this ensures that citizens who live abroad can exercise their rights in a simpler way. Furthermore, electronic voting and elections can be used to ask additional questions to users. This can increase the level of eParticipation and make the decision making process more precise. [Meier (2009), p. 52-53] About eGovernemental portals (and the same statement is true for an organisations or a companies portal), Meier *et al.* concludes that "not only does it provide government services to citizens and businesses, it can also include procedures and knowledge bases for eDemocracy. The road to Public Memory is a long and rocky one, since important issues such as protecting the privacy of citizens, copyright issues for digital objects, archiving concepts and times for digital storage media etc. must be clarified on an ongoing basis and implemented using suitable methods and techniques. Last but not least, public memory, implemented in a publicly accessible eGovernment portal, enables democratic political controlling and thus paves the way for an information and knowledge society." [Meier (2009), p. 60, translated] What does the empirical evidence from real world eVoting applications suggest? In 2011, the Swiss voting system was described as being 'decentralized'[Gasser and Gerlach (2011), p. 102]. However, this was not due to the use of blockchain technologies, but because the 26 cantons of Switzerland and its different votings systems. The cantons of Geneva, Zurich, and Neuchatel started a series of eVoting pilot projects already in 2001 and 2002. Remote, electronic voting was tested until 2006, when the projects were concluded and the report about them stated that "the Federal Government deemed the tests a success and opined that there was neither reason to stop further tests nor to hastily extend them"[10]. Also, more than 10% of Swiss citizens living abroad[11] makes the ability to cast a vote remotely especially appealing.

[10]Federal Council (2006a) Bericht über die Pilotprojekte zum Vote électronique vom 31. Mai 2006 [Report on the e-voting pilot-projects]. Federal Gazette 5459

[11]see https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/migration-integration/auslandschweizer.html, retrieved on August 19th 2020

By 2011, three cantons had an e-voting infrastructure in a decentralized (meaning that most cantons have developed their own system independently) and distributed way. Potential problems the federal government has seen in this approach were mostly of a human nature and already mentioned before, namely "a potential participation gap as a result of the digital divide, the de-ritualization of political rights, shorter processes of opinion formation, information overload, and security risks." [Gasser and Gerlach (2011), p. 108] A lot of countries offer parliamentary websites which support petition systems. This allows authorized users to create new petitions on which others can vote. Such applications need to correctly authenticate users in order to attain credibility by a wider public. In 2012, it was possible to use made up names or non-existing / newly-created freemail addresses on the official German and Austrian petition websites. Protection from SPAM was handled via a Captcha mechanism. At the time[12], four German and two Austrian petitions had more or close to 1000 support declarations. The biggest and first transnational petition website is the ECI, which "due to its potential size is arguably the current flagship eParticipation project in the entire EU"[Prosser (2012), p. 15]. It allows EU citizens to vote on petitions which will be discussed by the European Commission if at least one million users have signed a petition. Authentification is a potential problem here, since different member states use different means of authenticating users, and not all of them require a passport or ID for this purpose. [Prosser (2012), p. 13-15]

4.2.2 Smart Contract Based Systems

Decision making systems using Smart Contracts have already been tested, for example BroncoVote, which is a university voting system based on Smart Contracts on a private blockchain using Ethereum. Users who belong to a university can take part in its surveys or votes. They are authenticated by their student- or university-ID. [Meier (2020), p. 342-344] Advantages of this system were mentioned to be a lack of a central authority which could manipulate the results, the ability of users to verify if their vote was counted correctly and the ability to see the correct results of the voting. A disadvantage could be that users still may not trust algorithms and data structures when making decisions, even when using secure blockchain technologies. [Meier (2020), p. 350-351]

A blockchain-based voting system using Smart Contracts was also implemented in Yu *et al.* It is based on Hyperledger Fabric and has the objective of ensuring the security of its voting results while preventing attacks on the system. The blockchain platform does guarantee verifiability in this system, while cryptographic techniques, proof-of-knowledge, and linkable ring signatures are used to increase the user's level of privacy and security. [Yu et al. (2018)]

[12] June 17, 2012

5 Design and Implementation

The distributed computing platform used in the implementation of the BDSS is a soft fork of Ethereum called Quorum and the blockchain environment are cloud system technologies provided by Microsoft Azure.
Technologies used in the implementation include the Azure Virtual Machine, the Solidity Smart Contract language, the Azure Blockchain Service (to set up a private blockchain), Quorum (a soft fork of Ethereum which was developed by JPMorgan) and the ABW as an environment to host and run DApps.

There are alternative approaches of implementing Smart Contracts, e.g. with Hyperledger Fabric, BitML (for Bitcoin Smart Contracts), Tendermint, and other Smart Contract technologies. For this implementation, the well-known and widely used Ethereum distributed computing platform will be used.
There are a variety of cloud services offering Blockchain as a Service support, which could be used in this implementation as well. Some well-known examples include Microsoft Azure, Google Cloud, and Amazon AWS.
The implemented prototype enables users of the system to vote in decision making processes and to sign up for events. Organizers in the system are able to create votings and events. Users are authenticated by an administrator before they can take part in any of the processes supported by the private blockchain. Administrators handle the Smart Contract code in the cloud, can add new versions of it and authenticate users by inviting them to the Smart Contract(s) on the website of the ABW implementation.

Different kinds of election configurations were implemented to support different requirements of an organization:

- In a common election or voting, each person gets one vote. This kind of vote can be used for important decisions, e.g., in order to elect executives of the organization and can be conducted using this implementation.
- There can be votings where organizers have veto rights, e.g., the financial executive can decide if money should be spent on expenditures.
- There can be votings where the executives do have a higher weight of their vote. Their decisions are more important in this kind of election.
- In some votings, it may be possible for a user to abstain from it and to give his/her vote to another member. This can be done, e.g., due to an illness of the user which prevents him/her from taking part in the election, or because the decision is not that important to him/her. It is an optional choice to enable abstaining from voting in elections and it may only be an option in less important decisions.
- There are also decisions without an election if an expert, who was appointed to an authoritative position, makes a decision in his/her field. In this

case, the process is finalized immediately and no voting takes place. This kind of decision is simply an entry of data into the blockchain.

5.1 Simple Blockchain Voting System

This implementation is a simplified version of the Voting System presented in the next chapter. The point of it is to explain how the voting functionality works before showing the complex version of the system and all of its possible parameters.
The Simple Blockchain Voting System (SBVS) does have organizers and mem-

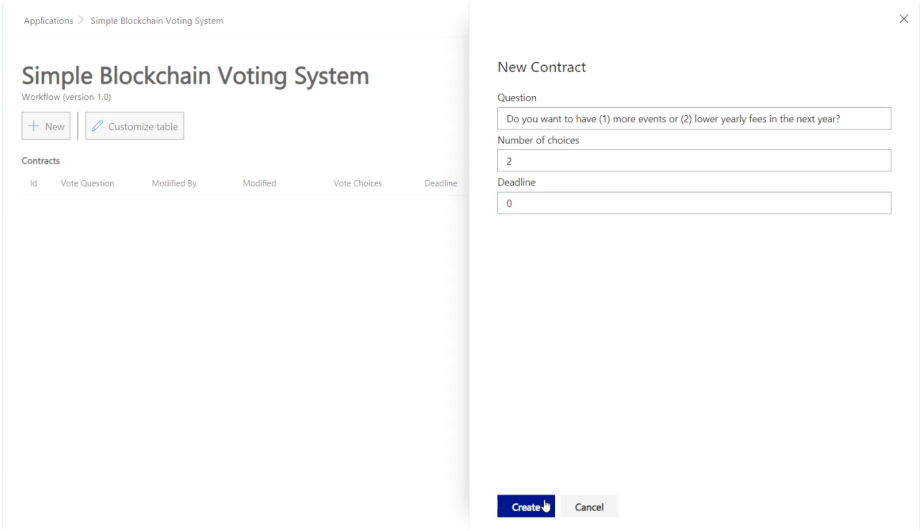

Figure 11: An organizer can type parameters of a new Smart Contract into these input fields

bers as roles. Organizers do create votes and members are voting in it. There are three states in the process: First, the organizer does ask a question, then members vote on it, and once the deadline is reached, an organizer can conclude the vote.
The parameters an organizer must enter to set up such a Smart Contract are the vote question, number of choices and an optional deadline. The user interface of the Smart Contract creation can be seen in Fig. 11.

The sequence diagram in Fig. 12 shows a typical process of the Simple Blockchain Voting Process, which is described in greater detail in chapter 6.
In this system, a user has access to a user interface provided by the ABW, which looks up a function defined in a JSON file (which contains allowed roles, states, transitions and functions) and triggers the function in the Solidity code.

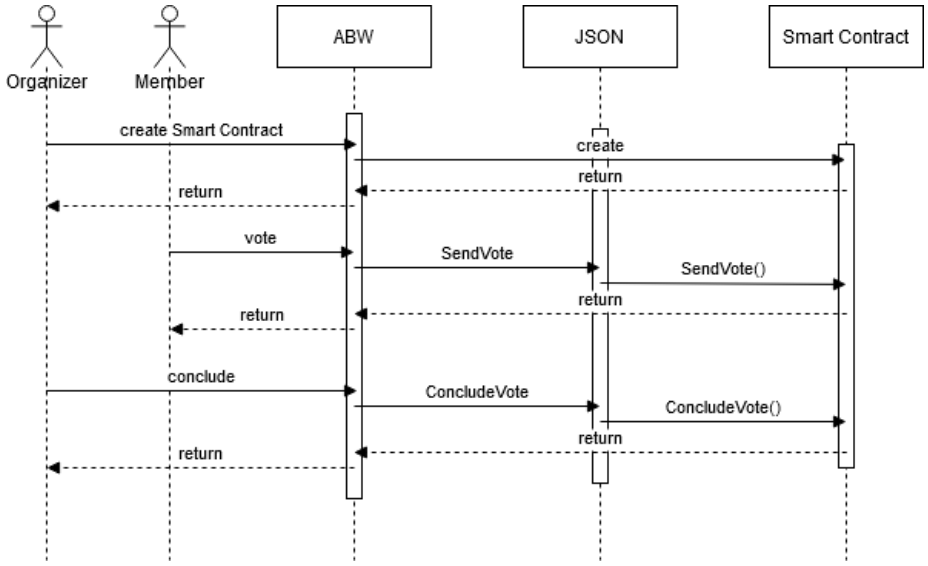

Figure 12: A Simple Blockchain Voting System diagram

In this diagram, an organizer is creating a Smart Contract, entering the parameters which will trigger the constructor of the Solidity file. Then, members of the system vote for their choices. Finally, an organizer concludes the decision making process.

5.2 Blockchain Voting System

This part of the implementation represents a decision making system which can be deployed in the ABW. The implementation consists of the Solidity file "BlockchainVote.sol", which contains all voting functionality, and the JSON file "BlockchainVote.json", which aids in enabling the interaction between the Azure service, including its user interface, and the Solidity code.

The UI of the ABW can be seen in Fig. 13. Here, a cancelled (failed) voting contract, its details and the previous activity of the Smart Contract, such as users voting or transferring votes, are visible to authenticated users.

5.2.1 Roles

Users of the network can be organizers or members.
Which functions a user is authorized to access depends on the parameters defined during the creation of the Smart Contract. Usually, an organizer starts a vote by creating the Smart Contract (see screenshot), and concludes it, while members can vote for it. However, by changing a few parameters, members can be

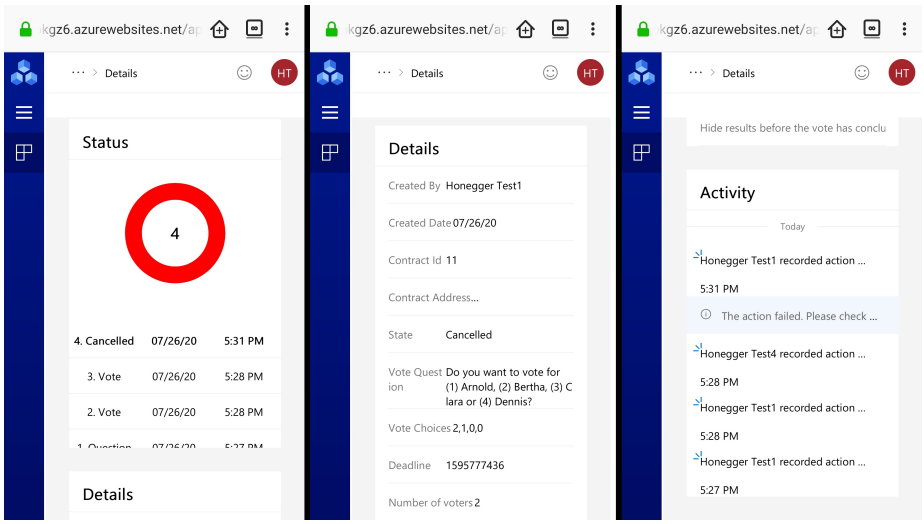

Figure 13: The ABW in a browser on a mobile device

excluded from a vote, or organizers can take part in it, or it can be simply a decision by one organizer.

5.2.2 States

The possible states of the decision making process are the following:

Question The first state after creating the Smart Contract where the (voting) question is being asked and nobody did yet vote.

Vote While voting is active, users can participate in the decision making process.

Finished The last state of this process if all conditions were met, e.g. number of voters which participated or the deadline reached. This state is reached once an organizer takes the action to conclude it.

Vetoed The last state of a vote if an organizer decides to veto a decision which was already concluded.

Cancelled The voting could not be concluded, e.g. because the deadline was reached, but not enough voters have taken part in the decision making process.

Roles, states and the transitions between them are defined in a JSON file which a developer needs to create. This JSON file is also connected to the

```
{
  "Name": "Vote",
  "DisplayName": "Vote",
  "PercentComplete": 60,
  "Value": 1,
  "Style": "Success",
  "Transitions": [
    {
      "AllowedRoles": ["Member"],
      "Function": "SendVoteAsMember",
      "NextStates": [ "Vote" ],
      "DisplayName": "Send Vote"
    },
    {
      "AllowedRoles": ["Organizer"],
      "Function": "SendVoteAsOrg",
      "NextStates": [ "Vote" ],
      "DisplayName": "Send Vote as organizer"
    },
    {
      "AllowedRoles": ["Organizer"],
      "Function": "ConcludeVote",
      "NextStates": [ "Finished","Cancelled" ],
      "DisplayName": "Conclude Vote"
    },
    {
      "AllowedRoles": ["Member"],
      "Function": "TransferVoteAsMember",
      "NextStates": [ "Vote" ],
      "DisplayName": "Abstain and transfer vote"
    },
    {
      "AllowedRoles": ["Organizer"],
      "Function": "TransferVoteAsOrg",
      "NextStates": [ "Vote" ],
      "DisplayName": "Abstain and transfer vote as organizer"
    }
  ]
},
```

Code Listing 1: JSON data snippet showing allowed transactions based on the 'vote' state of the process

Solidity constructor and functions used to run the Smart Contract. In the code snippet (listing 5.2.2), we see a part of the JSON code which defines how the Solidity code will work in the ABW. Specifically, it presents the "Vote" state and defines which possible actions can be taken once this state has been reached. Possible transitions in this state, which are allowed to users based on their role, are the functions:

- "SendVote" - Allows a member to vote in the system.
- "SendVoteAsOrg" - Allows an organizer to vote. It is important to have this as a separate function since it is possible to have votings where members, organizers or both can participate. In the corresponding Solidity function, it is checked if the user (based on his/her role) is allowed to vote.
- "ConcludeVote" - An Organizer can conclude a vote once the requirements have been met, e.g. a deadline reached and/or enough users having participated.
- "AbstainAndTransferVote" - Using this function, organizers and members can give their vote to another user.

Once a different state is reached, different transitions will be possible, which are defined in the JSON file.

5.2.3 Parameters

There are a number of parameters an organizer can use to set up an individual Smart Contract. Those parameters are used in the constructor of the Solidity contract, which is similar to object-oriented classes in other programming languages. The parameters an organizer can enter in the ABW UI can be seen in Fig. 14. Variables not visible in the screenshot are the vote question and the number of choices.

String Question The question/statement users are presented with when voting for a decision. The number and position of choices should be made clear in the question, e.g.: “Do you choose (1) Arnold, (2) Bertha, (3) Clara or (4) Daniel as head of our organisation?”. These numbers correlate with the array explained in the next point.

Integer Number of choices The amount of choices for the decision, e.g. the number of candidates members can vote for. In the (Solidity) Smart Contract implementation, this is represented in an integer array where a user vote increments one entry of the array by one. If e.g. users can vote for the candidates A, B, C and D, then they are represented by an array [0, 0, 0, 0] and a user voting for B changes this array to [0, 1, 0, 0].

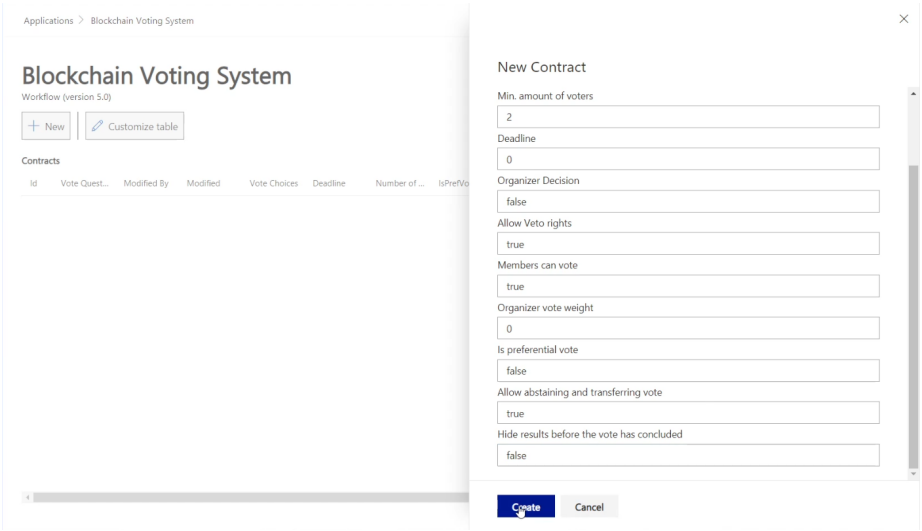

Figure 14: Available parameters of the Blockchain Voting System (BVS)

Integer Threshold Called the "Min. number of voters" in the UI, this parameter defines how many voters have to participate in the vote in order to successfully conclude it. If, for example, the threshold is set at "12", then twelve users have to vote to make the decision valid. If the threshold is not reached, but the deadline is, then the final state of the Smart Contract will be “cancelled”.

Integer Deadline Before the deadline ends, members can vote on the decision. Once it is over, organizers can conclude the vote to make the decision official. A value of 0 is allowed to create a vote without a deadline. If the deadline is accidentally set at a point before the current date, then it will automatically be set to zero. This value is an integer, and more specifically a Unix epoch timestamp (seconds since January 1st 1970).

Boolean OrgDecision If true, then it is not a vote, but a single decision by an organizer which will be stored in the blockchain. No voting takes place for this type of decision.

Boolean AllowVeto If true, then users with the role of an organizer can veto a decision after the vote was concluded. This leads to the vote being cancelled (not accepted). An organizer can veto a decision up to 24 hours after the vote was concluded.

Boolean MembersCanVote In a common voting process, this parameter is true by default. It lets members of the organisation vote for a decision.

Integer OrganizerVoteWeight If this variable is larger than zero, then organizers can take part in a vote. Also, usually every voter does have one (1) vote. If this integer parameter is larger than 1, then organizers do have a larger vote weight in the decision making process.

Boolean IsPrefVote Enables a preferential vote, also known as ranked voting. If there are the choices A, B, C and D in a common (non-preferential) voting, then voters can e.g. choose "3" to vote for C.
If it is a preferential vote, then users can express their choice in a more sophisticated way: If they like C the most, D is their second choice and A is their least favorite candidate/choice, then they can input “4, 3, 1, 2” as their voting choice. The favourite candidate will then get the largest amount of vote points, which are stored in an array. In this case, candidate C would get three points, D would get two and B receives one point.

Boolean AllowAbstain Allows a user to abstain from a vote and to give his/her vote to another user. Taken from a user-friendly drop-down menu, the wallet address of the user that receives the additional vote is used as input.

Boolean HideResults This variable is used to hide the combined results from the user. If true, then a user is still capable of seeing individual decision by other users, but the cumulative number displayed in the user interface will show e.g. "0,0,0,0" even if multiple users have already voted. Once the process is finished, the actual numbers will be revealed. This can be used to reduce the influence which other user's decisions have on the voter.

5.2.4 Voting

The sequence diagram in Fig. 15 describes a possible scenario of a process of the Blockchain Voting System. Here, the larger number of possible actions (compared to the simplified version in chapter 5.1) is evident: The organizer does have the option to transfer his vote (and he is also able to cast a vote). The function "TransferVoteAsOrg()" is necessary to determine whether the user has a role which is allowed to transfer votes in the current Smart Contract. Now, the member can vote twice since he has gotten a vote from the organizer. Then, the organizer concludes the vote. If the parameter for it is enabled, then he does have the option of vetoing the decision, which leads to the Smart Contract terminating in a cancelled state.

In the code snippet below (Code Listing 2), the vote functionality of the Blockchain Voting System can be seen. Its input is the voteNum, which is the choice a user has taken, and the vote_weight, which can be more than one (1) for organizers in special cases. The voteNum is an integer array in preferential votes, which expresses a users more and less preferred choices. Otherwise, it is still an array variable, but the user does only input a single value, which is then

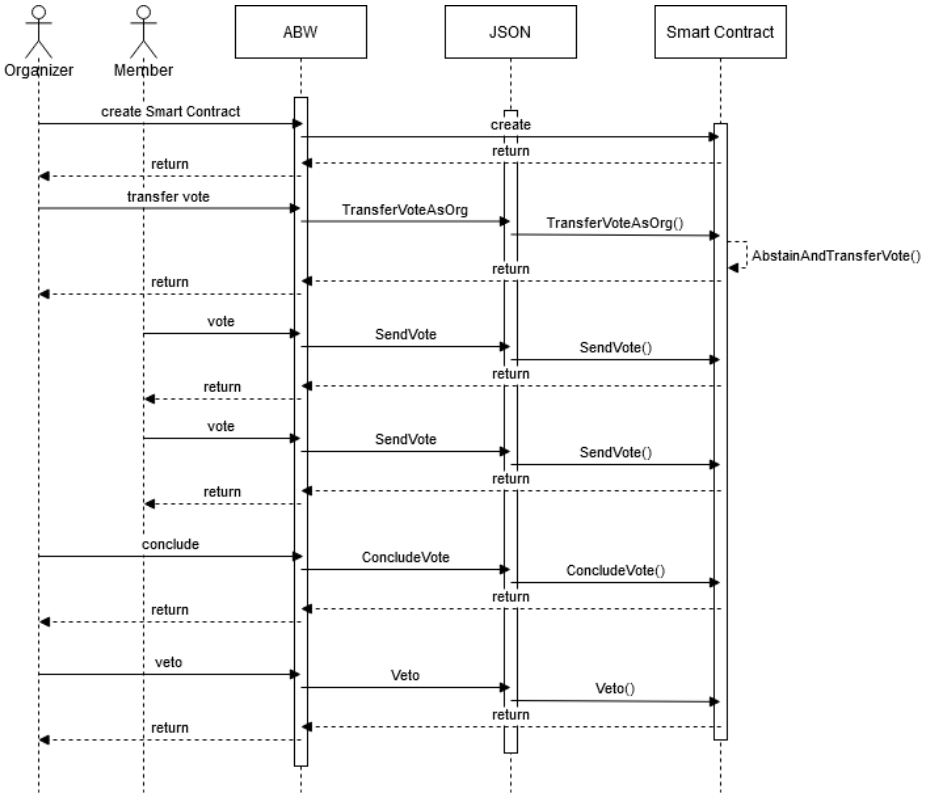

Figure 15: A Blockchain Voting System diagram

processed by the "vote" function.
The boolean value "voteInProgress" is used to check if another user is currently using the function. In order to prevent issues in the system, such as one user manipulating the system by voting with multiple devices at the same time and his/her votes erroneously getting counted multiple times, this global lock variable is used. It can be argued that such a measure could lead to multiple users getting blocked in the function if they are voting at the same time, however this potential problem is tested in chapter 6.4 to evaluate whether this could be an issue in a real world scenario or not.
The wallet address of a user, voter_wallet (msg.sender), is taken to uniquely identify him/her. This is crucial to the decision making process, as this variable will be stored in a mapping (with the function setWallet(voter_wallet)) to signify that a user has already voted. If "contains(voter_wallet)" is true, then he/she did already vote and can not repeat this process.
Also, the "Deadline" variable must be larger than the current time. If the deadline was set at zero by the organizer, then this variable is simply ignored and not

used in the process. Something which should be noted about this parameter is that the "currentTime" is determined by the "block.timestamp" function, which is the only way of getting the current time in Solidity. This function gets the timestamp as a uint256 value in seconds since the epoch [13], provided by a nodes in the network. A potential issue here could be that nodes can influence this value, and it can be manipulated to be up to 900 seconds off[14]. This was not an issue when experimenting with my implementation, since the value was exact to the second while using the Smart Contract in the ABW, however this limitation should be paid attention to by developers using the "block.timestamp" function in Solidity implementations.

```
1  // vote functionalitiy for members and organizers
2  function vote(uint[] memory voteNum, uint vote_weight) public
3  {
4      if(voteInProgress == false)
5      {
6          voteInProgress = true;
7
8          address voter_wallet = msg.sender;
9
10         uint currentTime = block.timestamp;
11         if(contains(voter_wallet) == false && (Deadline >=
               currentTime || Deadline == 0))
12         {
13             State = StateType.Vote;
14
15             // check if a vote was transferred to current user
16             if(containsTransferred(voter_wallet)){
17                 removeTransferredWallet(voter_wallet);
18             }
19             // if not, use own vote
20             else {
21                 setWallet(voter_wallet);
22             }
23             NumOfVoters = NumOfVoters + 1;
24
25             if(IsPrefVote == false){
26                 if(!HideResults)
27                     Choices[uint(voteNum[0]-1)] = Choices[uint(
                           voteNum[0]-1)] + vote_weight;
28                 else
29                     ChoicesHidden[uint(voteNum[0]-1)] =
                           ChoicesHidden[uint(voteNum[0]-1)] +
                           vote_weight;
30             }
31             else {
32                 uint checkNum = 0;
```

[13] see the documentation at https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=block#block-and-transaction-properties, retrieved on August 20th 2020

[14] see https://github.com/ethereum/wiki/blob/c02254611f218f43cbb07517ca8e5d00fd6d6d75/Block-Protocol-2.0.md, retrieved on August 20th 2020

```
33                  uint voteNumAccumulated = 0;
34                  for (uint i = 0; i < NumOfChoices; i++) {
35                      if(voteNum[i] > NumOfChoices || voteNum[i] < 1)
                            {
36                          voteInProgress = false;
37                          revert("Error - Incorrect number for
                                preferential vote detected.");
38                      }
39                      voteNumAccumulated = voteNumAccumulated +
                            voteNum[i];
40                      checkNum = checkNum + i+1;
41                  }
42                  //Check if the user did input too many choices (
                        array too long. Too short array will be caught
                        before.)
43                  if(getCount(voteNum) != NumOfChoices){
44                      voteInProgress = false;
45                      revert("Error - Inconsistent number of choices
                            for preferential vote detected.");
46                  }
47                  if(voteNumAccumulated == checkNum){
48                      for (uint i = 0; i < NumOfChoices; i++) {
49                          if(!HideResults)
50                              Choices[uint(i)] = Choices[uint(i)] + (
                                    NumOfChoices*vote_weight - voteNum[
                                    i]*vote_weight);
51                          else
52                              ChoicesHidden[uint(i)] = ChoicesHidden[
                                    uint(i)] + (NumOfChoices*
                                    vote_weight - voteNum[i]*
                                    vote_weight);
53                      }
54                  }
55                  else{
56                      voteInProgress = false;
57                      revert("Error - Incorrect numbers for
                            preferential vote.");
58                  }
59              }
60          }
61          else{
62              voteInProgress = false;
63              revert("Error - you can only vote once");
64          }
65          voteInProgress = false;
66      }
67      else{
68          revert("Error - voting currently in progress");
69      }
70  }
```

Code Listing 2: Voting functionality in the Solidity code

Next, it is confirmed if a user has gotten a vote from another user. If a vote was transferred, then it appears in an integer mapping and "containsTransferred(voter_wallet)" will return true. In this case, the integer in the mapping is reduced by one, and the user can vote with it. Otherwise, the function "set-

Wallet(voter_wallet)" is used, utilizing his/her own vote.
Then the "NumOfVoters" variable is increased by one. This variable is visible to users of the system in the UI, and it is used to verify if the threshold (min. number of voters required to successfully finish the process) has been reached.
If results are not hidden, then they are added to the "Choices" array. Otherwise they are added to the "ChoicesHidden" array, which is not visible to users, and then transferred to the "Choices" array only once the process has concluded.
If it is a preferential vote, then a number of conditions have to be met to check if the user input is valid:
The highest number a user inputs can not be a higher value than the "NumOfChoices" array is long. The array can not be longer or shorter than the number of choices. Also, an integer variable is used to confirm if the user has distributed the correct amount of vote weight (voteNumAccumulated == checkNum), e.g. if there are four choices, then it would be 10 (1+2+3+4) points which he/she can give to the different choices. This way, the user does still have some flexibility in his/her choices, e.g. he/she can choose "0,2,2,4" or "1,2,3,4", but he/she can not give one user an unusually low or high amount of vote weight, which secures the integrity of preferential votes in this implementation.

5.2.5 Transferring a vote to another user

The "WalletsTransferred" mapping maps wallet addresses to uint's in order to confirm whether or not a user has gotten vote(s) from other users. In the function "AbstainAndTransferVote" (see Code Listing 3), a user can input the target wallet address, which he/she chooses from a drop-down menu. This will only work if the organizer who has created the Smart Contract allows transferring votes. Also, a user can not transfer a vote to himself.

```
1  //do not vote and give vote to someone else
2  function AbstainAndTransferVote(address target_address) public
3  {
4      address voter_wallet = msg.sender;
5      uint currentTime = block.timestamp;
6
7      //transfering vote allowed and target is not own address
8      if(AllowAbstain && voter_wallet != target_address)
9      {
10         // if user did not vote yet and the vote is still active
11         if(contains(voter_wallet) == false && (Deadline >=
               currentTime || Deadline == 0))
12         {
13             // if user got transferred votes, then remove them
                   first
14             if(containsTransferred(voter_wallet)){
15                 removeTransferredWallet(voter_wallet);
16             }
17             //else, remove all votes of user
18             else{
```

```
                setWallet(voter_wallet);
            }
            addTransferredWallet(target_address);
        }
        else
        {
            revert("Error - could not abstain from vote.");
        }
    }
    else
    {
        revert("Error - could not abstain from vote.");
    }
}
```

Code Listing 3: Vote transfer functionality in the Solidity code

Next, the function examines if the user has already voted and if the deadline has not passed yet. If the voting is still active, then it will be checked if the user has gotten votes from other users (which he/she can theoretically give to yet other users), and if that is not the case, then his/her own vote will be transferred to the target user and the user will not be able to give his/her vote in the decision making process. The target's address is then added to the "WalletTransferred" mapping via the "addTransferredWallet" function.

5.2.6 Concluding the process

There are a few requirements which have to be met for an organizer to conclude a voting successfully. In the "ConcludeVote" function (see Code Listing 4) it is first validated if the number of users is at least as large as the predefined threshold - the number of users who need to take part in the voting. It should also be noted that if a user has transferred his/her vote to another user, that this action will not increase the number of users who have taken part in the process. Also, the deadline needs to be either disabled (if it has a value of zero), or it needs to have passed. If both conditions are met, then results of the voting will be shown (if the organizer did set the HideResults parameter to be true, otherwise they were already visible), and the process is finished successfully. The current time is then recorded in the finishTime variable, which is necessary to allow organizers to veto the decision up to 24 hours after the process has concluded.

```
// conclude the voting if all conditions are met. This leads to the
    last successful (finish) state of the contract.
function ConcludeVote() public
{
    uint currentTime = block.timestamp;

    // if threshold and deadline reached
    if(NumOfVoters >= Threshold && (Deadline < currentTime ||
        Deadline == 0))
```

```
8       {
9           //if choices were hidden, then reveal them now
10          if(HideResults)
11              for (uint i = 0; i < NumOfChoices; i++)
12                  Choices[uint(i)] = ChoicesHidden[uint(i)];
13
14          State = StateType.Finished;
15          finishTime = block.timestamp;
16      }
17      // deadline (exists and) reached BUT not threshold (not enough
            votes) => Cancelled state
18      else if(NumOfVoters < Threshold && Deadline < currentTime &&
            Deadline != 0)
19      {
20          State = StateType.Cancelled;
21      }
22      // conditions not (yet) met => do not finish or cancel
23      else
24      {
25          revert("Error - not all conditions met to conclude voting."
                );
26      }
27  }
```

Code Listing 4: Vote concluding functionality in the Solidity code

It is also possible that the deadline has passed, but not enough users have taken part in the process (NumOfVoters $<$ Threshold). In this case, the final state of the process will be "cancelled".
If an organizer tries to conclude the voting but the conditions to conclude it have not yet been met (e.g. deadline has not yet passed), then he/she will simply receive a warning and the voting process will continue.

5.3 Event Signup System

Another feature of the BDSS is the "Event Signup" Smart Contract, which allows organizers to create events to which both members and organizers can sign up. The time of the sign up process is defined by a deadline until which users can sign up, and after which the process can be concluded. The UI showing the parameters an organizer can enter into the fields is shown in Fig. 16.

5.3.1 States

CreatedEvent An event was just created by an organizer.

SignupEvent Users are signing up to the event, which is possible until the deadline passes.

Finished The sign up process was finished successfully.

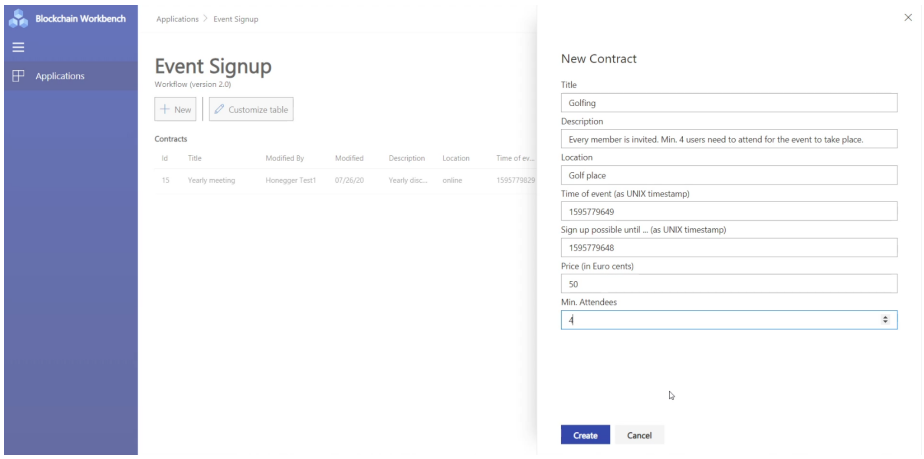

Figure 16: Creating an Event Signup Smart Contract.

Cancelled The sign up process could not be finished successfully, e.g. because not enough users signed up or the organizer has cancelled the event. Only the organizer who created the event can cancel it. This is ensured by confirming if the wallet address does belong to the same organizer.

5.3.2 Parameters

String Title The name of the event.

String Description A short description of it.

String Location The location where it takes place.

Integer TimeOfEvent The time when the event takes place (as a UNIX timestamp). The date must be in the future.

Integer TimeSignup The time (deadline) until which the users can sign up to the event. Needs to happen before the event starts.

Integer Price The expected costs / ticket price / entry fee / etc. for each individual user.

Integer Min. Attendees The minimum amount of users required to finish the sign up process successfully. If the deadline is reached, but not enough users did sign up, then the process will conclude in a "Cancelled" State.

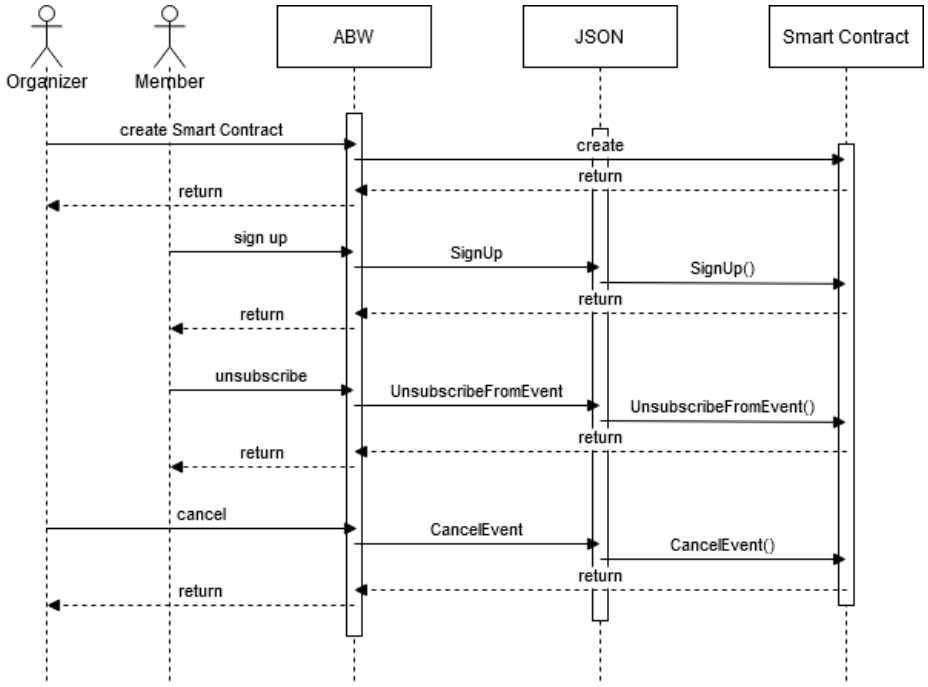

Figure 17: An Event Signup System diagram

A diagram explaining a possible process in the Event Signup System can be seen in Fig. 17. Here, an organizer creates an event Smart Contract, and a member signs up. Later, the member decides to unsubscribe from the event. Finally, the organizer chooses to cancel the event, ending the process in a "Cancelled" state. It is also possible for him to conlude the event successfully, which leads to the "Finished" state of the process.

5.4 Simultaneous Voting Script

To verify whether or not users are able to vote at the same time, a multi-threaded Python script is used to simulate users pressing the vote buttons at one moment. This script will be used in an experiment in chapter 6.4.

The following code snippet describes how four different threads start up and log into Microsoft Azure accounts. Usernames, passwords and the information whether they are organizers or members is stored in arrays, and this information is used as input to start the threads. There is a pause of a few seconds between starting the threads in order to decrease the chance of a browser crashing on start.

```
#start all voting threads
i=0
for user in users:

    #start thread
    if(use_same_accounts):
        x = threading.Thread(target=voter_thread, args=(users[0],
                                                        passwords[0],
                                                        is_organizer[0]))
    else:
        x = threading.Thread(target=voter_thread, args=(user,
                                                        passwords[i],
                                                        is_organizer[i]))
    x.start()
    i+=1
    # pause so that the threads will not crash on start
    sleep(6)
```

Code Listing 5: Starting multiple threads opening browser windows to vote simultaneously

Python Selenium is a library which provides a WebDriver API to remotely control or automate actions in a common browser like Firefox, Edge, Chrome, and more. This way, automated online beta tests (and other web tasks) can be conducted easily.
Inside the threads of this implementation, Selenium is used to make the browser automatically log in and navigate to the vote button of the Smart Contract UI. In the code snippet below, we can see how the main thread votes simultaneously using the other threads. The browser will look for the field where the vote number is typed in (the variable "vote_num_field" in the Python script) [15] using XPATH, then press TAB so the "Take action" button will be in focus again (and thus clickable), and then the thread will wait.

```
#enter number
vote_num_field = driver.find_element(By.XPATH, '//input[contains(
                                    text(),"")]')
vote_num_field.send_keys(vote_choice)
vote_num_field.send_keys(Keys.TAB)

#wait for exact millisecond, then
while(vote_time > time.time()):
    sleep(0.0001)

#click vote button ("Take action")
try:
    driver.find_element_by_id('id__38').click()
except:
    try:
        driver.find_element_by_id('id__32').click()
    except:
```

[15] The user interface showing a user voting can be seen in Fig. 20

```
17          try:
18              driver.find_element_by_id('id__34').click()
19          except:
20              driver.find_element_by_id('id__40').click()
21
22  print("current time: " + str(time.time()))
23  print("voting with " + user + ";" + datetime.now().strftime("%d.%m.
                                    %Y %H:%M:%S") + "\n")
```

Code Listing 6: Threads will wait and vote concurrently (in the range of 20ms)

All threads will then vote, but only at the time when the predetermined "vote_time", which is a UNIX timestamp, is reached. Then the "Take action" button is clicked by all browsers at the same time, and the time of this event is recorded.

6 Demo Application and Scenario

In this chapter, we are explaining the experimental evaluation of the Business Decision Support System (BDSS). This is done in a test run with example users.

In the ABW, an administrator (and every user who was added to a Smart Contract) can see all of their available applications (see Fig. 18). An administrator can upload new DApps or add a new version of an existing DApp.

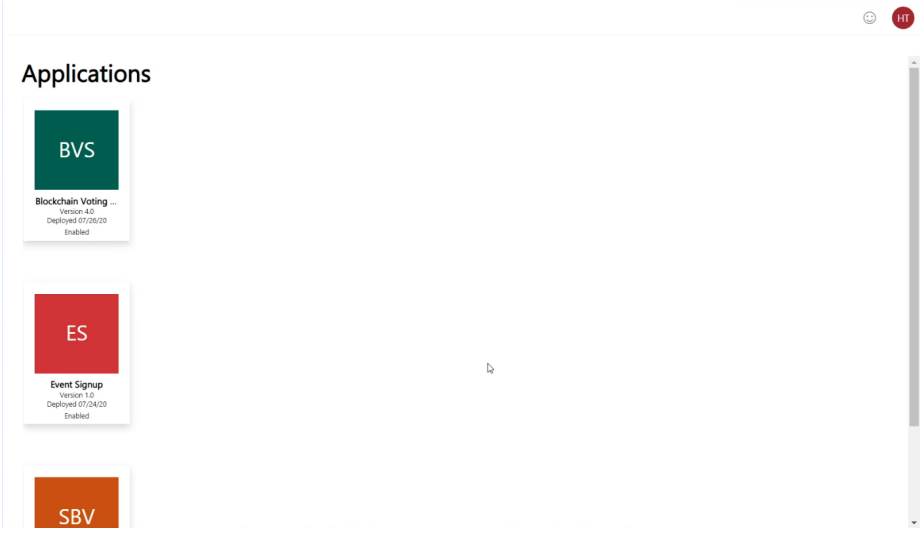

Figure 18: Overview over Smart Contract applications (DApps) while being logged in as a user (organizer)

By clicking on one of these DApps, the user can see more details about it, such as the parameters associated with it or information about previously terminated Smart Contracts. The user can either create a new Smart Contract if he/she is an organizer or take part in them as a member.
In the following chapters, we are demonstrating the use of my implementation using four test accounts. The first of these accounts (HoneggerTest1) does have the role of an organizer while the other three users (HoneggerTest2, HoneggerTest3, HoneggerTest4) do have the role of a member in the implementation. The different test users and their roles in the implementation are listed in table 2.

Users and roles in the BDSS		
User number	Account name (Azure Cloud)	Role (in the implementation)
1	HoneggerTest1	organizer
2	HoneggerTest2	member
3	HoneggerTest3	member
4	HoneggerTest4	member

Table 2: Overview of users and their roles in the Smart Contract implementation

6.1 Simple Blockchain Voting System

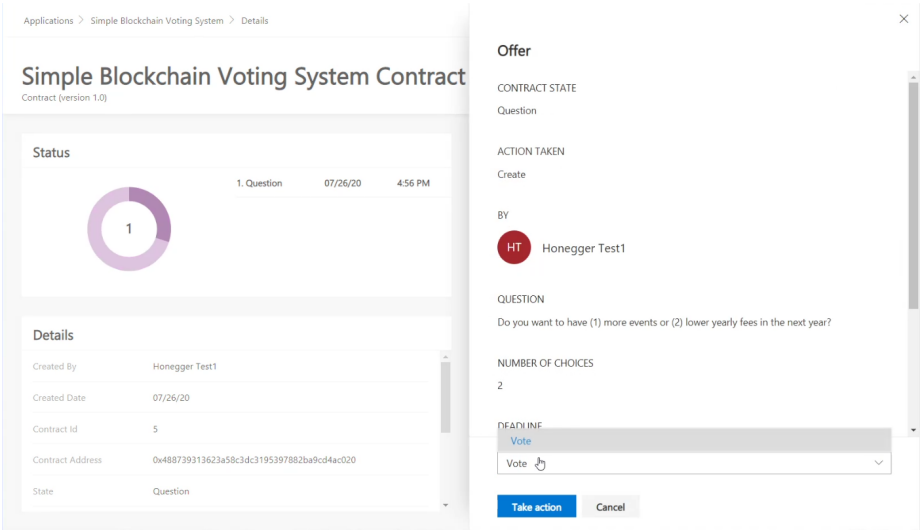

Figure 19: A member takes the action to vote in an active Smart Contract. On the right side, the previous action by an organizer (user 1, see table 2) can be seen.

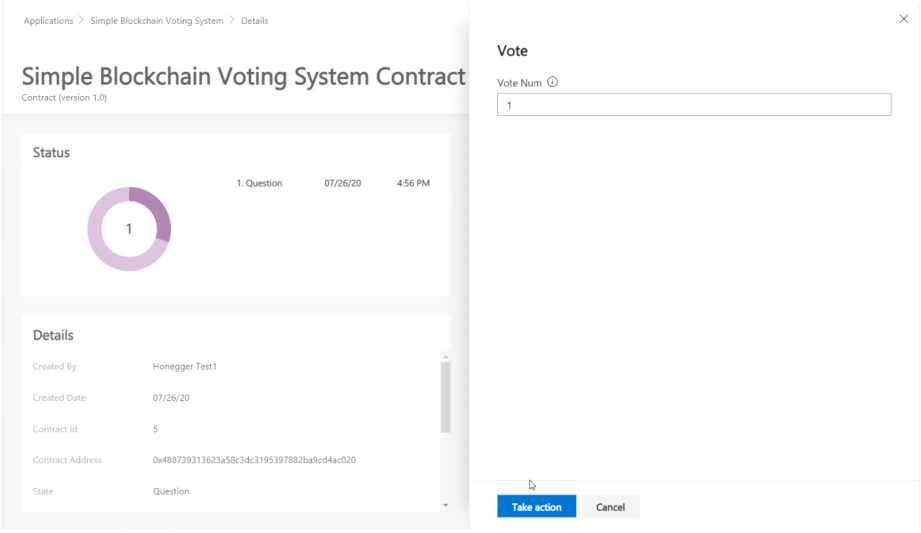

Figure 20: The user decides to vote for choice 1 (in this example: more events planned by the organisation)

First, we demonstrate the decision making functionality of the simplified version of the Blockchain Voting System. Here, we create a Smart Contract with the parameters 'Question', 'Number of choices' and 'Deadline'. Those parameters correspond to the variables VoteQuestion (String), Choices (uint array) and Deadline (uint, Unix timestamp) in the constructor of the Smart Contract file, which is written in Solidity.

In this example, we ask users of an organisation a simple question with two possible answers: Do they want to have more events (e.g. networking meetings, meetups, etc.), or do they want to lower their annual fees? By clicking on 'create' (see Fig. 11), we create the new Smart Contract with these parameters in a previously programmed Solidity file.

Logged in as a member of the DApp, it is possible to vote for a decision (Fig. 19) using an integer as an input, which is typed into a text field on the user interface and can be seen on Fig. 20.

We are using the HoneggerTest2 account to cast the vote, and once it is recorded, we are logging out (Fig. 21) and log into HoneggerTest3.

We are repeating this process for all three test members, and once everyone did cast their vote (Fig. 22), we are logging into the organizer account again.

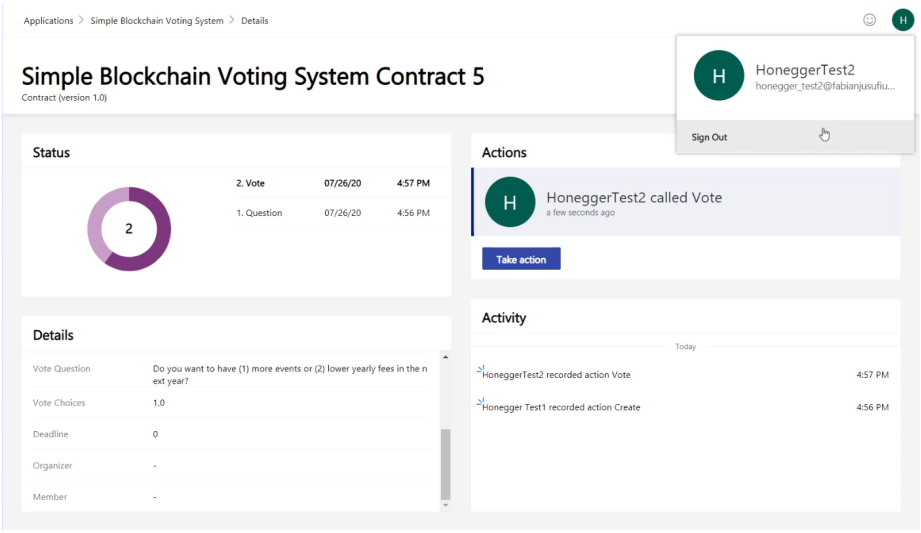

Figure 21: One member has voted and is logging out of his/her account

We are testing if it is possible to vote as an organizer, which fails as expected. This gives me a 'The action failed. (...)' warning message on the user interface, which can be seen in Fig. 24. In the simplified version of the Blockchain Voting System, it is not possible to vote as an organizer. We then take the action of concluding the vote (Fig. 23). The result of this test run is that the first choice was preferred by the members of this system.

This will lead the Smart Contract to go into the state of being 'finished' (Fig. 24), and now no more action can be taken - the result of the decision making process is final.

In a real-world example, the end of a decision making process will be mainly defined by a deadline at one exact point in time instead of being terminated by an organizer at any point in time. But in this example, we chose a deadline of '0', which leads to the deadline parameter being ignored. This was done for simplicity's sake. In the following examples, the more complex makeup of parameters of the Blockchain Voting System (not the simplified version of this chapter) will be presented, and examples using a deadline can be found in the subsections 6.2.2 and 6.2.3, as well as in the event sign up examples of subsections 6.3.1 and 6.3.2.

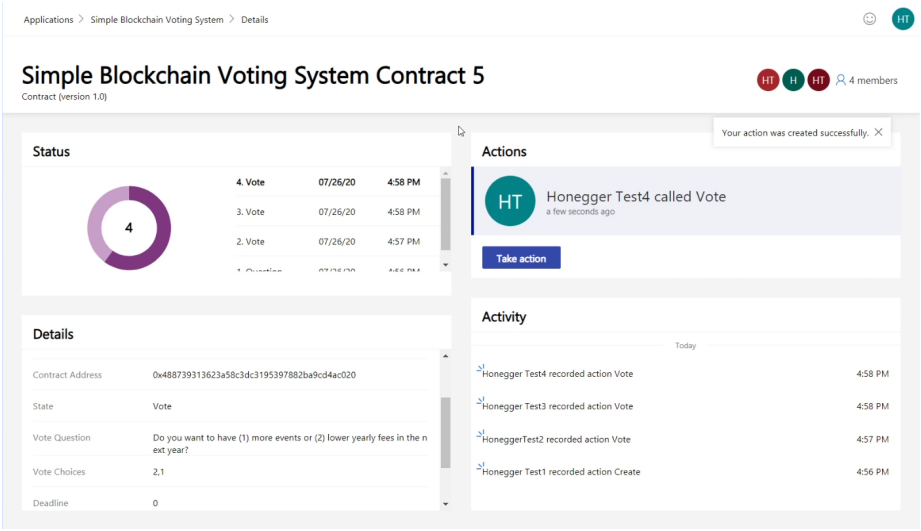

Figure 22: All three users have voted

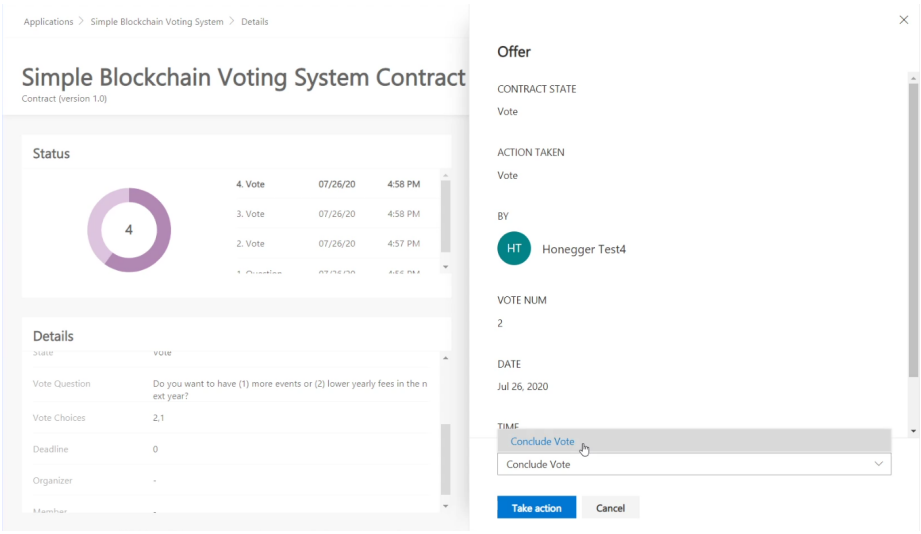

Figure 23: An organizer concludes the vote. On the right side, the previous action by user 4 (a member, see table 2) can be seen.

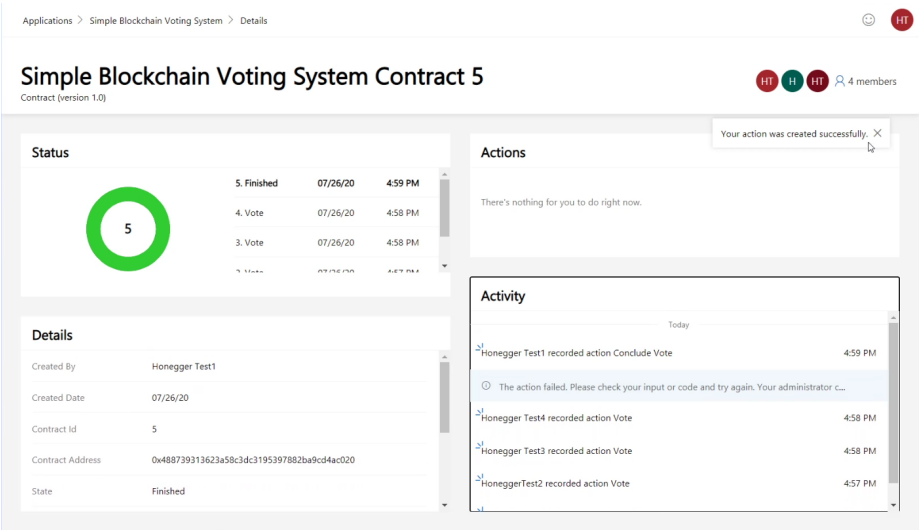

Figure 24: The final state of the Smart Contract is reached

6.2 Blockchain Voting System

The Blockchain Voting System (BVS) is a more complex implementation (compared to the SBVS) which gives an organizer a number of parameters which he/she can use to create the decision making process desired by the organisation or business.
Besides the questions, number of choices and deadline, he/she can also choose to allow veto rights, define how many users need to take part in the voting, give organizers a vote weight, create a preferential vote, allow abstaining from the vote (and allowing users to transfer the vote to someone else), and hide the result (Vote Choices variable, seen in Fig. 47) from the user while the vote is in progress. Those values are optional, and the organisations which handle the Smart Contracts must decide which parameters they choose to enable in their decision making processes.
An organizer can also decide if members, organizers or both can vote, or if it is a decision by the organisation itself, which can then not be voted on, but which is immediately terminated and directly stored in the blockchain.
All of the parameters mentioned here will be described in detail in the following subsections.

6.2.1 Transfer vote, threshold, conclude and veto

In this test run, we will see how a vote can be transferred to another user, how a voting process is concluded, and how this same process can then be vetoed by an organizer.

In this Smart Contract, there is a threshold ('Min. amount of voters') of two (Fig. 14). This means that at least two users need to take part in the vote for it to be terminated successfully. If this goal is not reached, then the Smart Contract would terminate in a failed state ('cancelled'). It is not an organizer decision, only members [16] can take part in it, and Veto rights are enabled for organizers. It is not a preferential vote - users can only express their wish by voting for one decision. Abstaining from the vote is possible in this example, and it will be used. Also, results are not hidden from the users. This would commonly be done in a real world example in order to not influence the decision of a user[17], but for testing and demonstration purposes, we will be able to see results in the following test runs.
This example's question is a vote for a candidate: "Do you want to vote for (1) Arnold, (2) Bertha, (3) Clara or (4) Dennis?". Users choose one of those four available candidates.

A user can choose to abstain from a vote and transfer it to another user who has access to the Smart Contract. He/She picks the correct user from a drop-down menu (Fig. 25). After pressing the "Take action" button (Fig. 26),

[16] If the organizer vote weight is 0, then voting is disabled for them

[17] Which would be influenced e.g. by the halo effect, giving already popular decisions even more votes

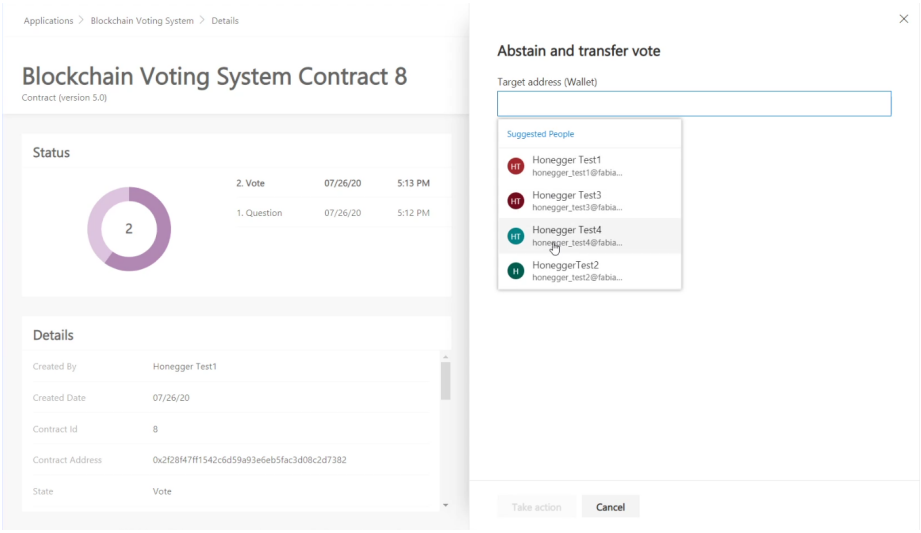

Figure 25: A user chooses another user he/she wants to transfer his/her vote to

this input will be processed as a wallet address (of the user to whom the vote is transferred) in the back-end (the Solidity code). In this example, test user 3 gives his/her vote to test user 4.
Then, test user 3 tries to vote or transfer his/her vote a second time. Both of these attempts fail as they should. We then log out of test user 3 (Fig. 27) and log into user 4. Next, test user 4 votes, and at first, his/her transferred vote (from user 3) will be taken in order to do that. Then he/she votes again (Fig. 28), and this time his/her own vote is counted. In practice, this means that he/she can vote twice, since he/she has received one vote. Finally, user 1 concludes the vote (Fig. 29) and the Smart Contract reaches the state of being "Finished" (Fig. 30).
But this is not the last state of the process in all BVS Smart Contracts. Here, an organizer can decide to veto this decision if this parameter is enabled. This leads to the Smart Contract reaching the final state of being "Vetoed" (Fig. 31).

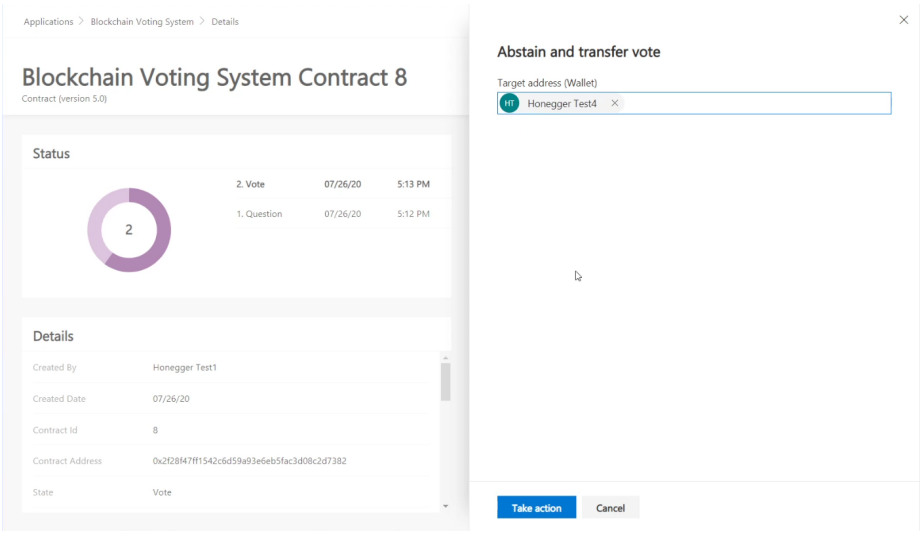

Figure 26: Test user 4 was selected. After clicking on "Take action", this choice is confirmed.

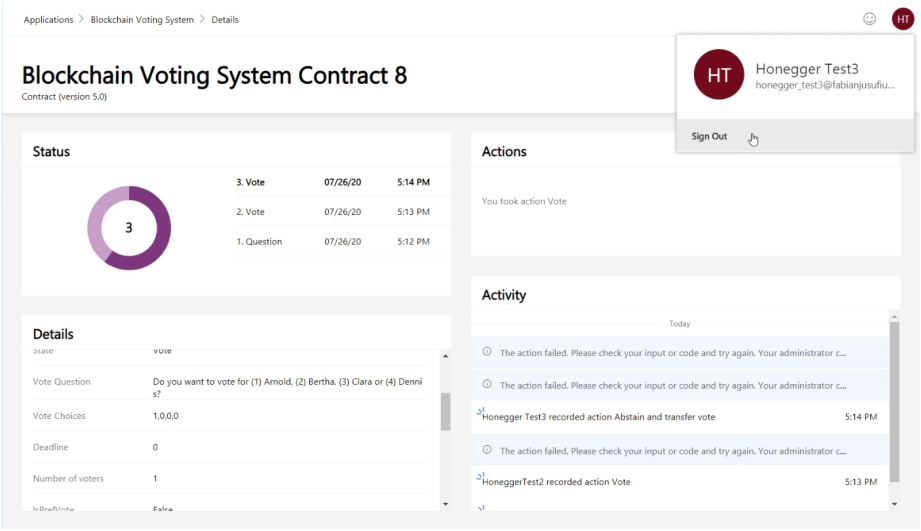

Figure 27: A user could not vote or transfer a second time and is logging out

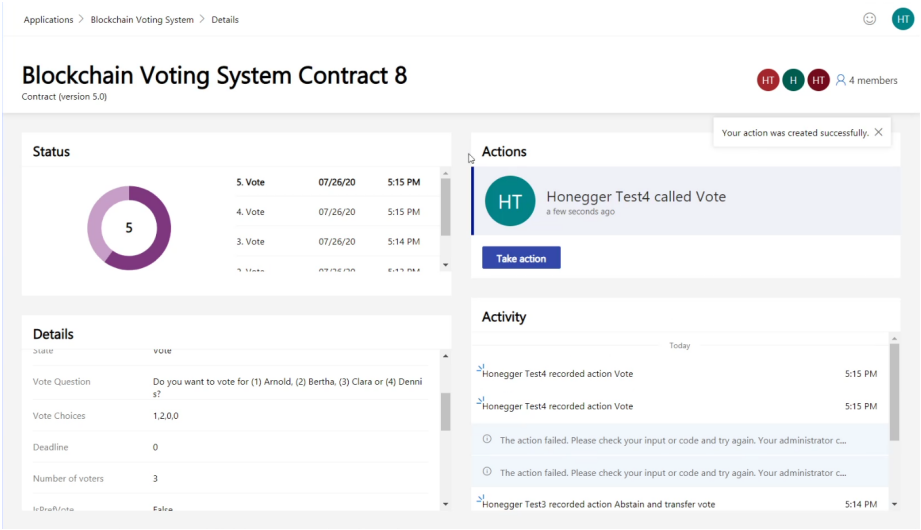

Figure 28: User 4 received a vote and can now vote twice.

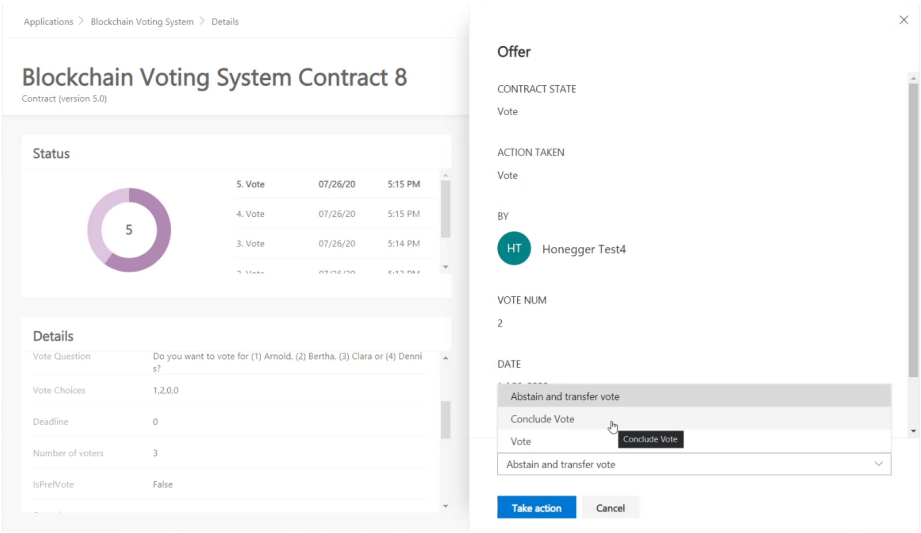

Figure 29: User 1 concludes the vote

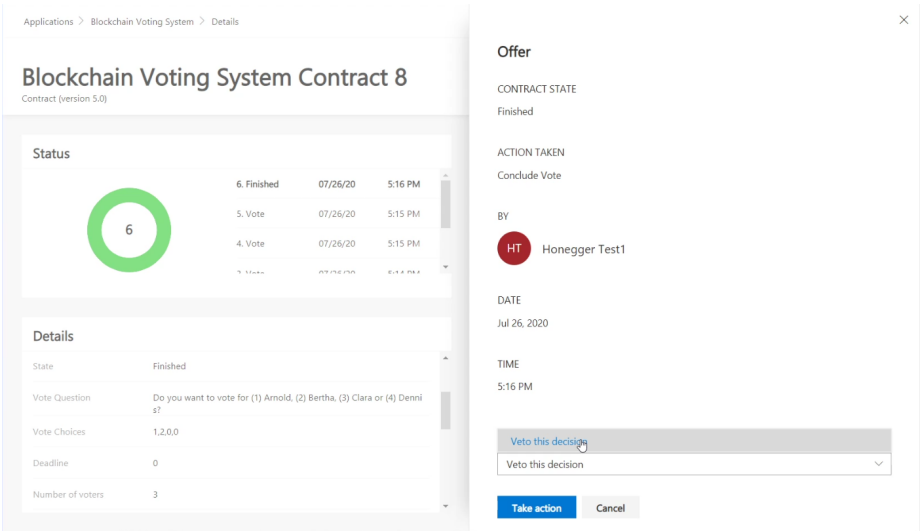

Figure 30: Once the voting is finished, it can be vetoed

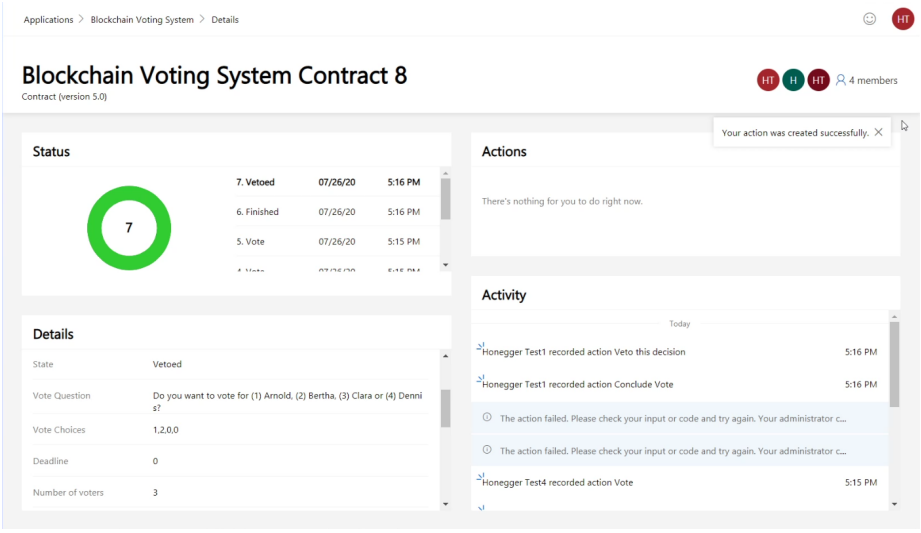

Figure 31: In this case, the last state of the Smart Contract is it being "Vetoed"

6.2.2 Preferential voting and successful conclusion after a deadline

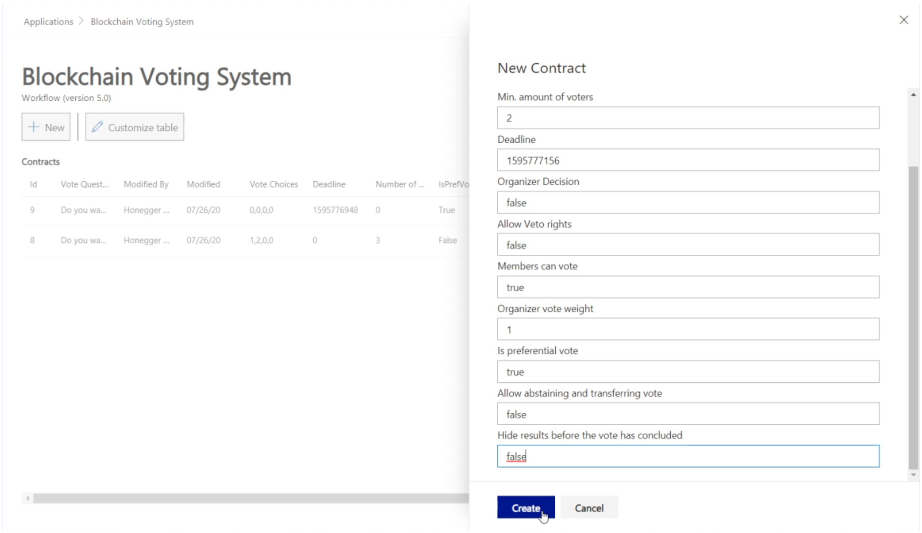

Figure 32: A GUI showing a Smart Contract where preferential voting is enabled.

The next Smart Contract handles the voting input as a preferential vote (Fig. 32). This means that he/she needs to rank the candidates from most to least favorite. In Fig. 33, we can see how the user picks the first candidate as his/her most and the last one as his/her least favorite candidate. Also, we are testing if we can transfer a vote after voting, which should not, and is not possible in this system.
The next user's favorite candidate is the second one, and his/her least favorite one is the last candidate. On the left part on the screen, in the "Details" section in Fig. 34, it can be seen how the first user's vote was processed - the most favorite candidate got 3 points, number two got 2 points, and number three got one point. Once the second vote is added to the resulting array, we get the result of 5,5,2,0 points (Fig. 35) for the four candidates (3+2, 2+2, 1+1, 0+0). In this simulation, we also tried to conclude the voting process before the deadline has passed, and this action failed as it should (Fig. 36). Finally, the vote can be concluded (Fig. 37) at a date later than the predefined deadline.

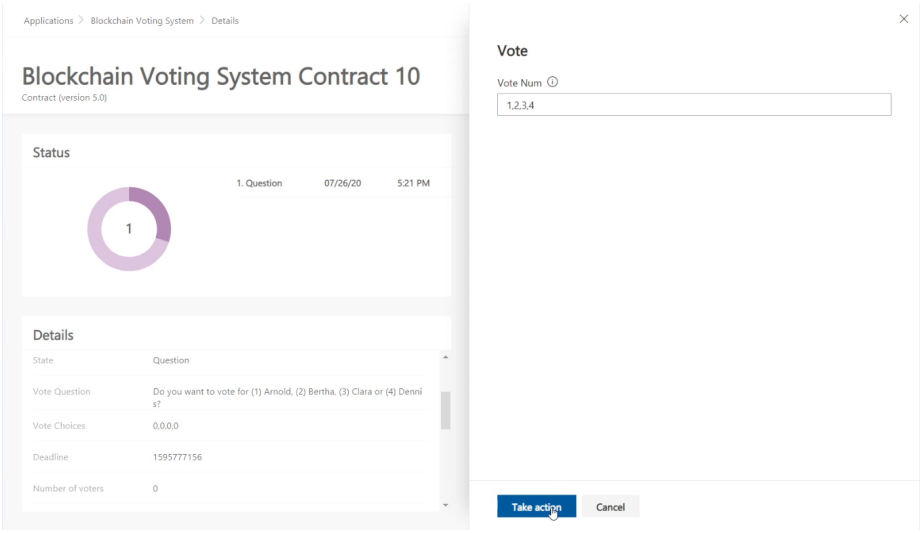

Figure 33: User 1 takes the action of casting a vote. The first candidate is his/her favorite choice.

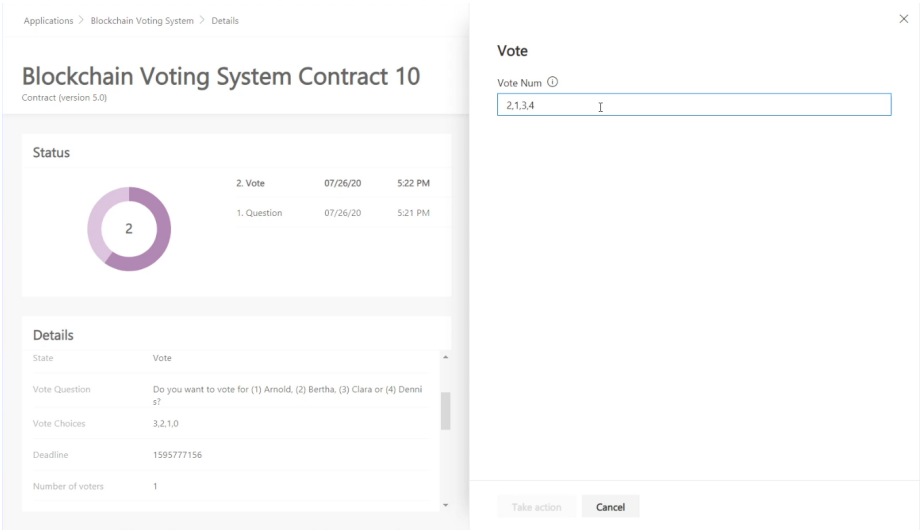

Figure 34: The second user votes. On the left side, the decision of user 1 is visible. These numbers ("3,2,1,0", compared to the numbers "1,2,3,4" in Fig. 33) are inverted because the favorite choice (1) gets the most points (3) in the election implementation.

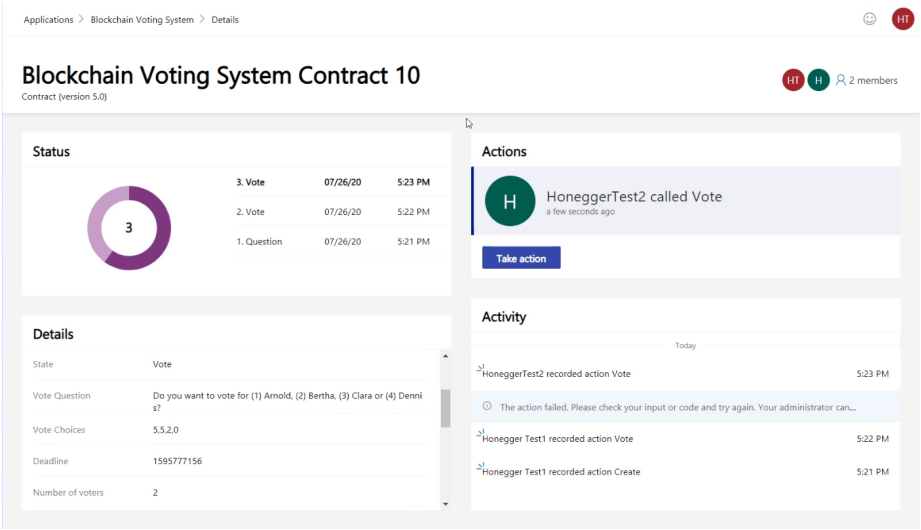

Figure 35: Two users have voted, and the preferential vote arrays are combined.

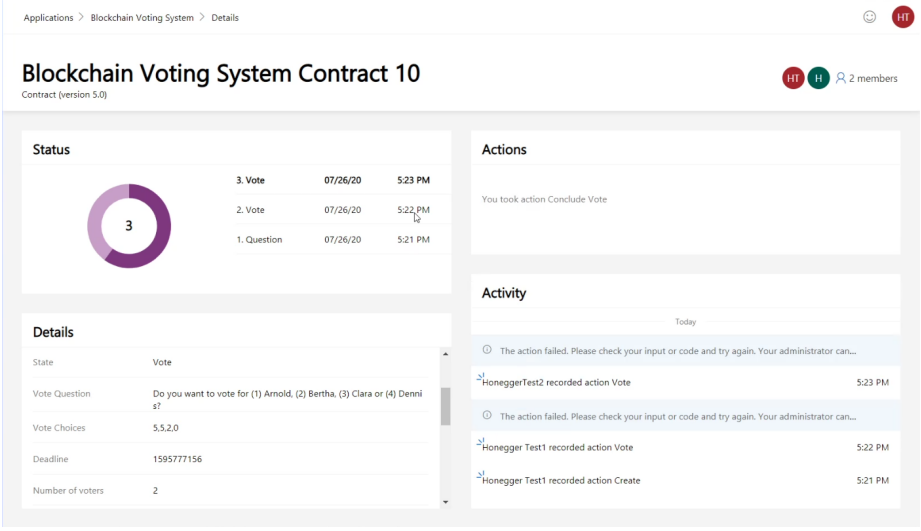

Figure 36: Concluding the process before the deadline has passed is not possible.

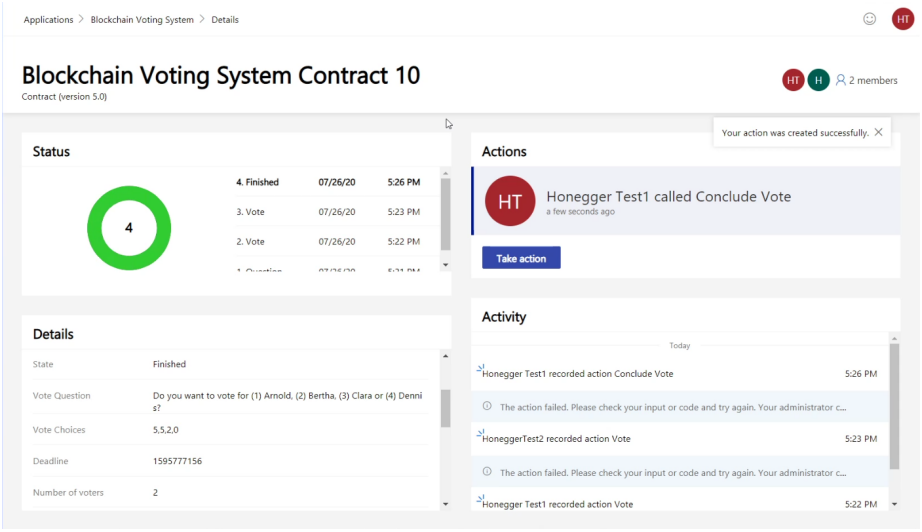

Figure 37: The preferential vote is finished. The result between candidate 1 and 2 is a tie.

6.2.3 Vote weight, early conclusion attempt and failed process due to deadline

In this Smart Contract, organizers do have a vote weight of two, which gives them double the vote weight of a regular member (Fig. 38). Also, there is a deadline to the decision making process, meaning that users can only take part before the deadline ends.

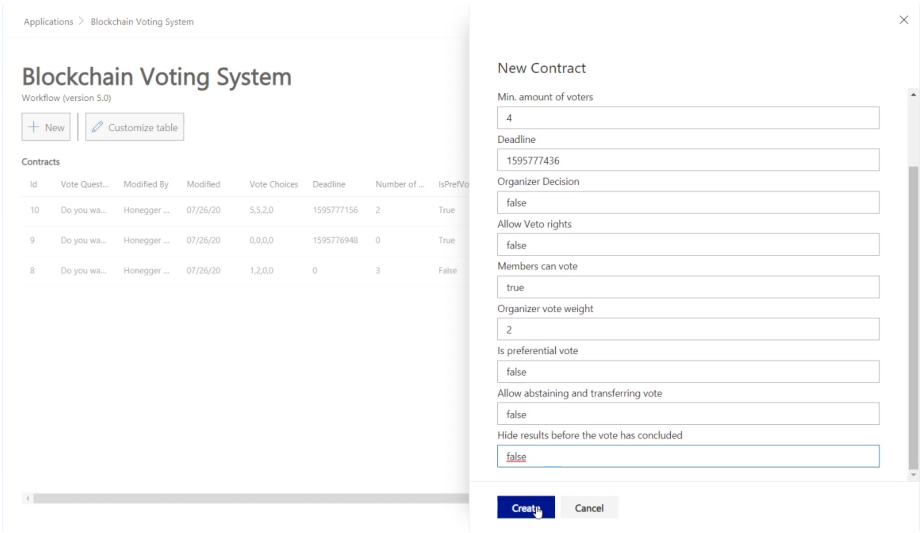

Figure 38: A GUI showing the creation of a Smart Contract with a vote weight, deadline and a minimum number of voters.

At first test user 1, the organizer in this system, does cast a vote for the first choice, which will be counted as two votes (Fig. 39). Next, user 4 votes for the second choice, and since he/she is a member (see table 2), his/her vote is counted as one vote, which we can see in Fig. 40. Now, the organizer tries to conclude the voting process, which fails as expected - the deadline was not reached yet, and users are still allowed to cast a vote (Fig. 41).
Next, the deadline is reached and the organizer attempts to conclude the vote (Fig. 42), but this action fails and the Smart Contract reaches its final state of being cancelled (Fig. 43). This happens because not enough users did take part in the decision making process - at least four users need to cast their vote, but only two did.

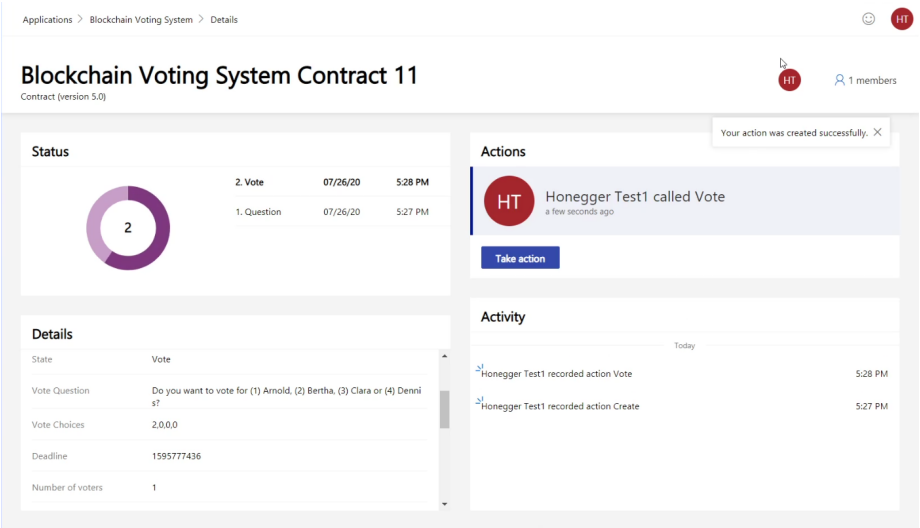

Figure 39: The organizer does have a vote weight of two.

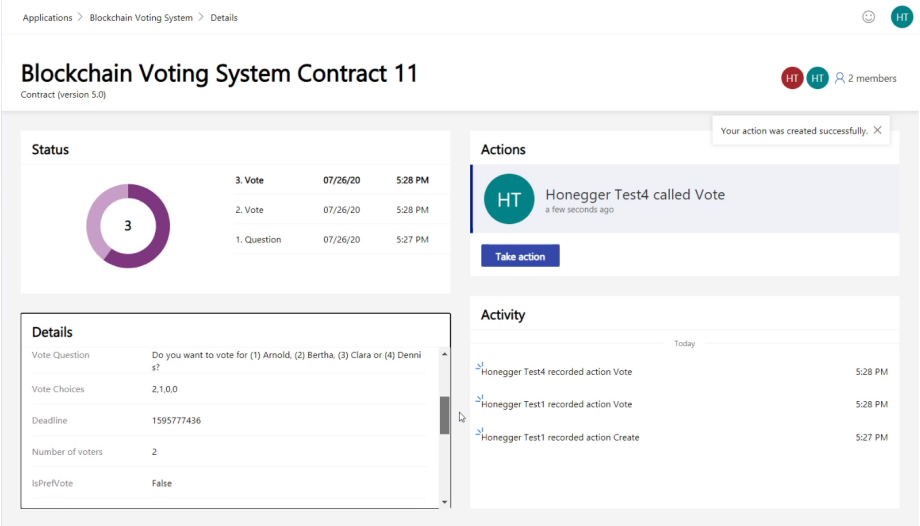

Figure 40: The member does have a vote weight of one.

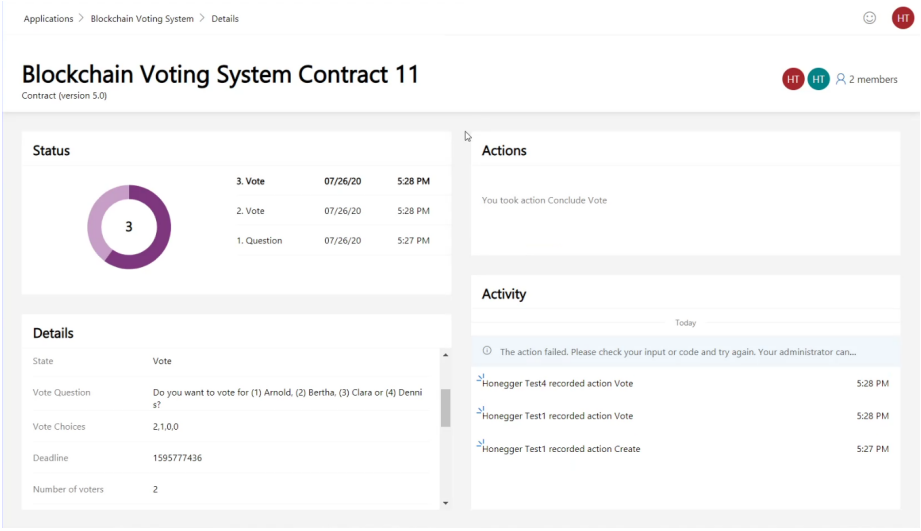

Figure 41: Can not conclude the process before the deadline ends. Users can still vote.

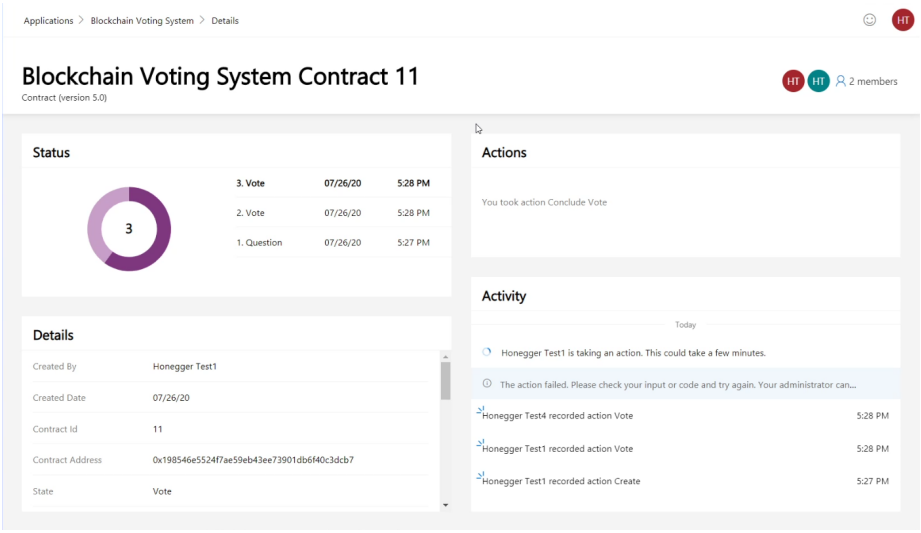

Figure 42: Attempting to conclude the voting process once the deadline is reached.

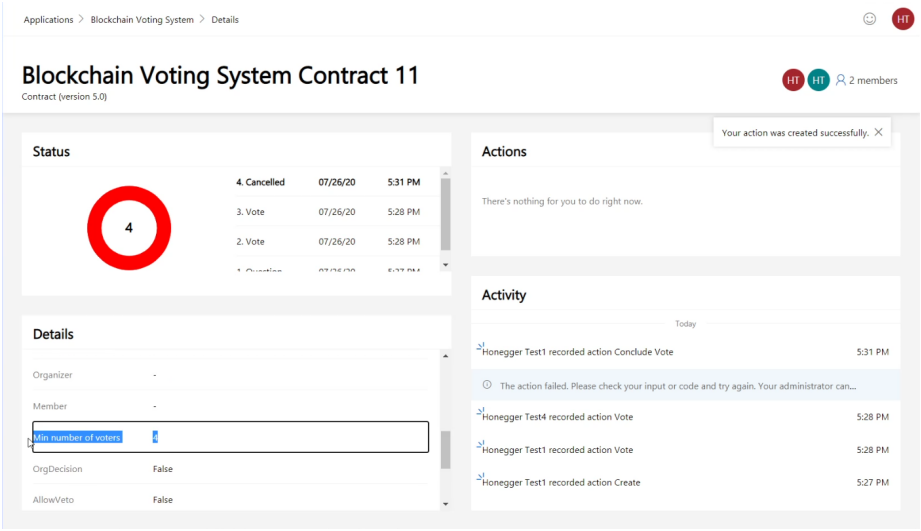

Figure 43: A minimum of four users need to vote to successfully conclude this process.

6.2.4 Hiding results before the process is finished

In the next test run, the results of the voting will be hidden in order to not influence the decision making process (Fig. 44). In Fig. 45, we can see that the results ("Vote Choices") are hidden - users only see "0,0,0,0", as if nobody has voted yet. This way, the voting will not be influenced e.g. by users voting for already popular choices. However, it is still possible for users to view and check if theirs or others' choices were correctly recognized by the BDSS (Fig. 46).

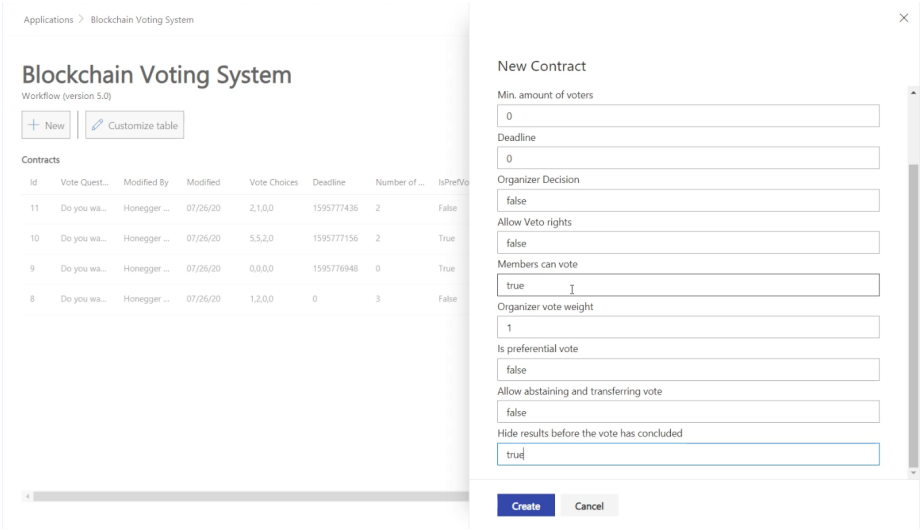

Figure 44: A Smart Contract where combined results of the voting are hidden.

Only once the voting process has concluded (Fig. 47), the cumulative results will become visible (Fig. 48). Now we can see that the voters chose the forth candidate with two votes.

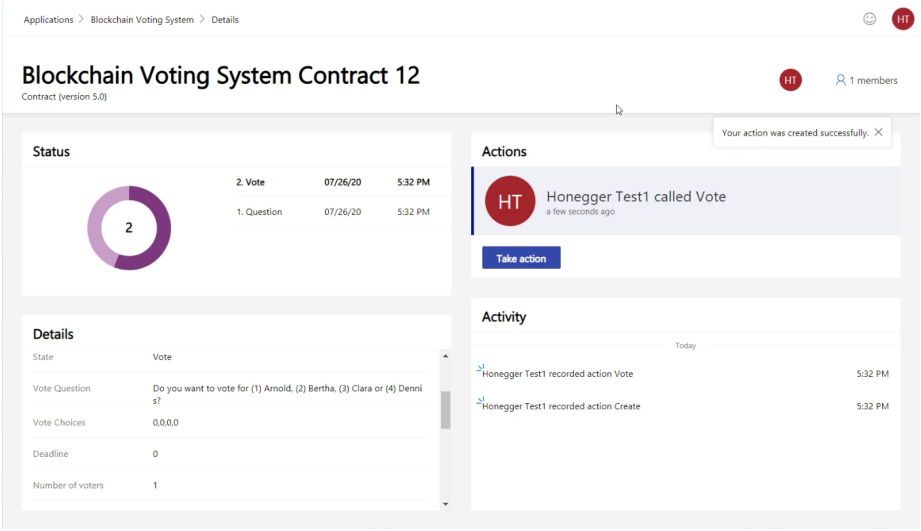

Figure 45: User 1 casts a vote, which is not visible in the choice variable ("Vote Choices")

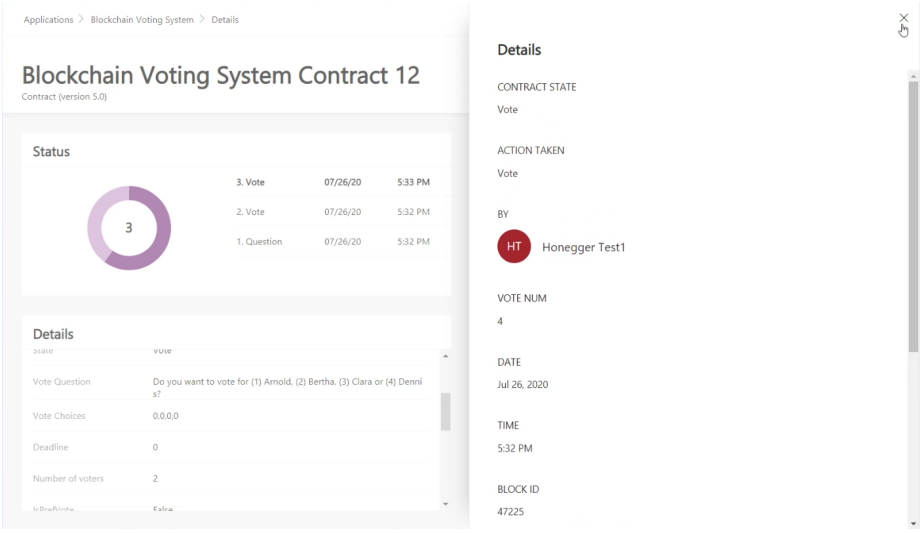

Figure 46: Individual choices can bee viewed.

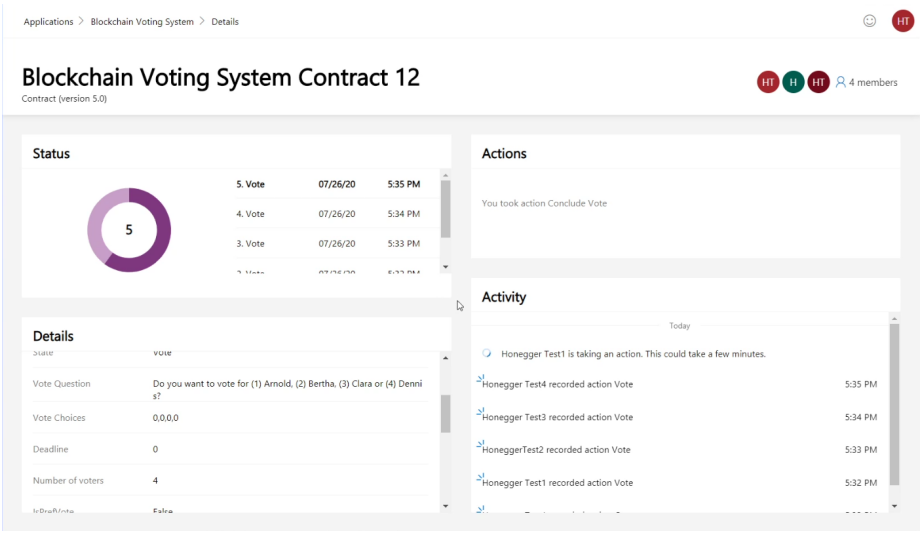

Figure 47: Four persons did vote, results are not yet visible.

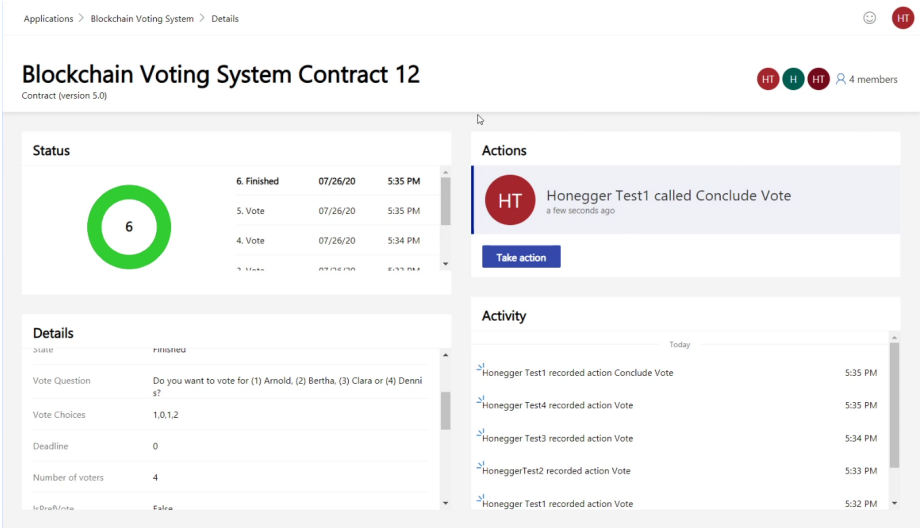

Figure 48: Results become visible once the process was concluded by an organizer.

6.2.5 Organisation decision and overview over processes

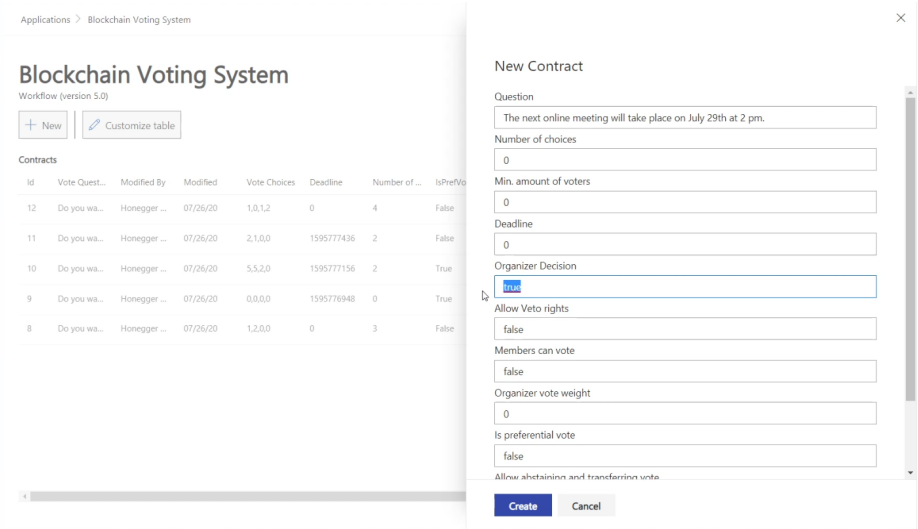

Figure 49: An UI that aids in the creation of a Smart Contract. This SC is a decision by an organizer.

The last demonstration of the BVS functionality is the most simple kind of decision possible to conduct with it. An organizer does have the ability to make an "Organizer Decision" (Fig. 49). By doing this, an organizer can directly store a decision in the blockchain. The Smart Contract will then immediately terminate in a "Finished" state (Fig. 50).

Another aspect of note about the ABW is that users get a handy overview of contracts (Fig. 51) which have already terminated as well as currently active Smart Contracts in which they can still participate. By clicking on the individual Smart Contracts, their specific values of the parameters and variables can be viewed in a practical way.

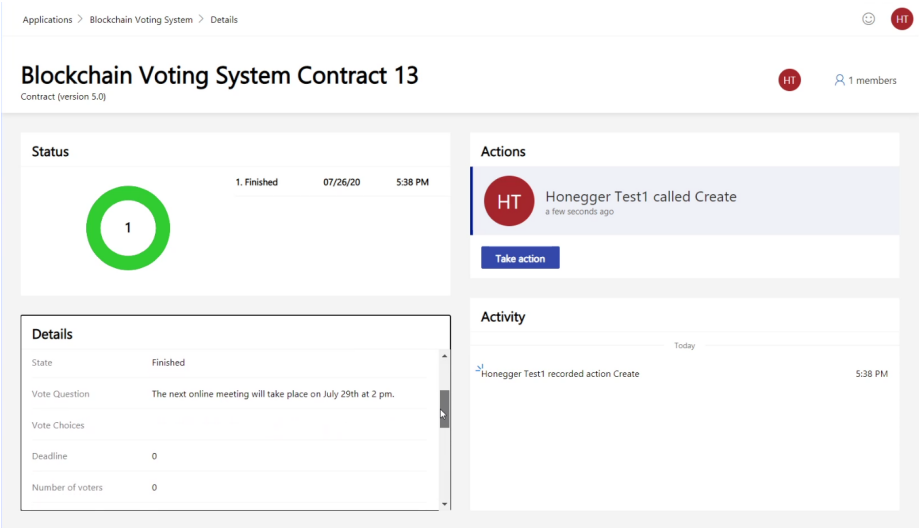

Figure 50: This Smart Contract's state is "Finished" at start.

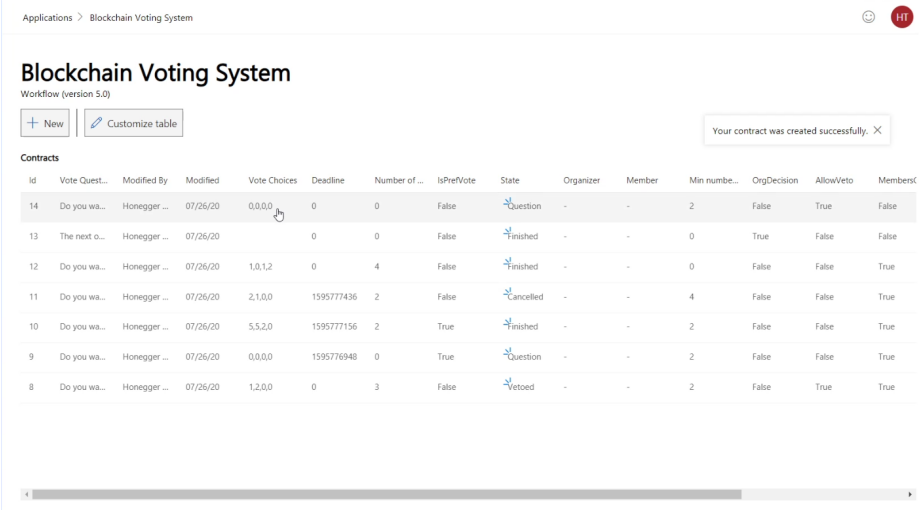

Figure 51: Overview over recent and past Smart Contracts in the ABW

6.3 Event Sign Up

6.3.1 Cancelled state example

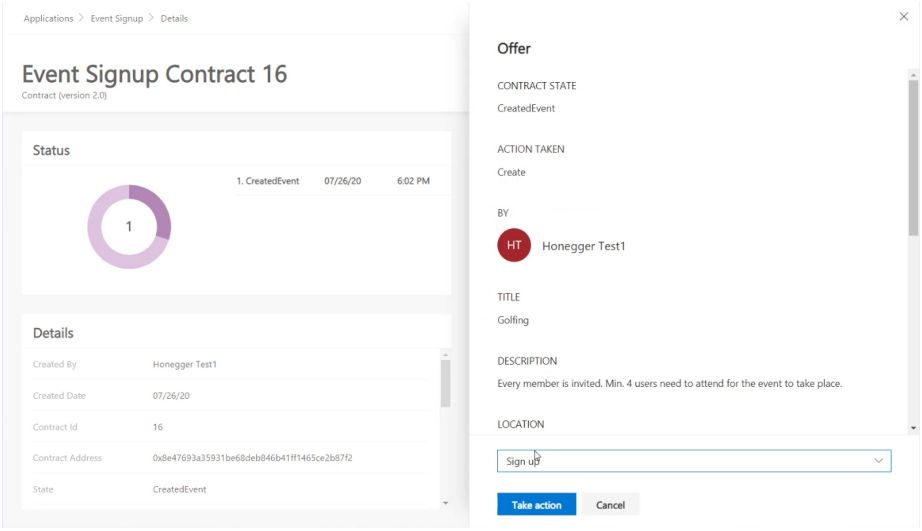

Figure 52: A user signs up to an event.

This BDSS allows its organizers to create events to which users can sign up. In this test run, an organizer creates a golfing event to which at least four people need to sign up in order for it to take place (Fig. 16). There is a time of the event and time for the users to sign up (which needs to end before the event takes place). It also displays the price of the event to the users.
Once the Smart Contract is created, users can sign up to it. In (Fig. 52), we see user 2 signing up after user 1 has done so already. After three users have signed up for the event, the organizer is concluding the sign up process (Fig. 53), which fails since the deadline was not yet reached (Fig. 54). Users can still sign up. Once the deadline passed, the organizer attempts to conclude the process again (Fig. 55), and the Smart Contract's state ends up being "Cancelled" (Fig. 56) since not enough users chose to attend this event.

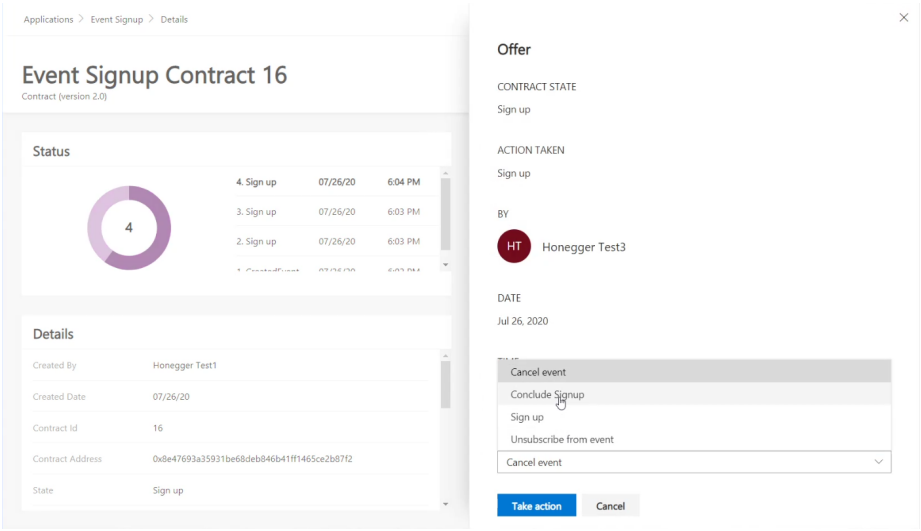

Figure 53: Once three users have signed up, the organizer tries to conclude the sign up process.

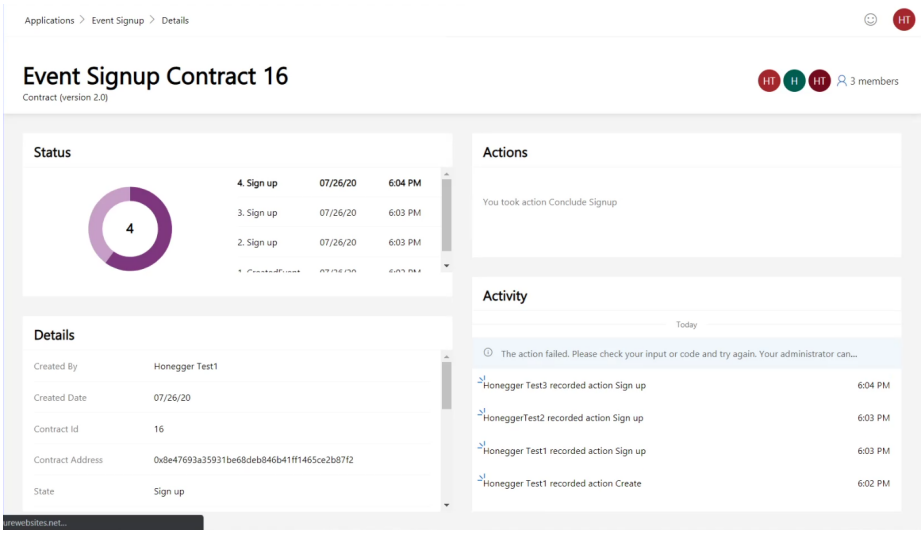

Figure 54: The deadline was not reached, so the process could not be terminated.

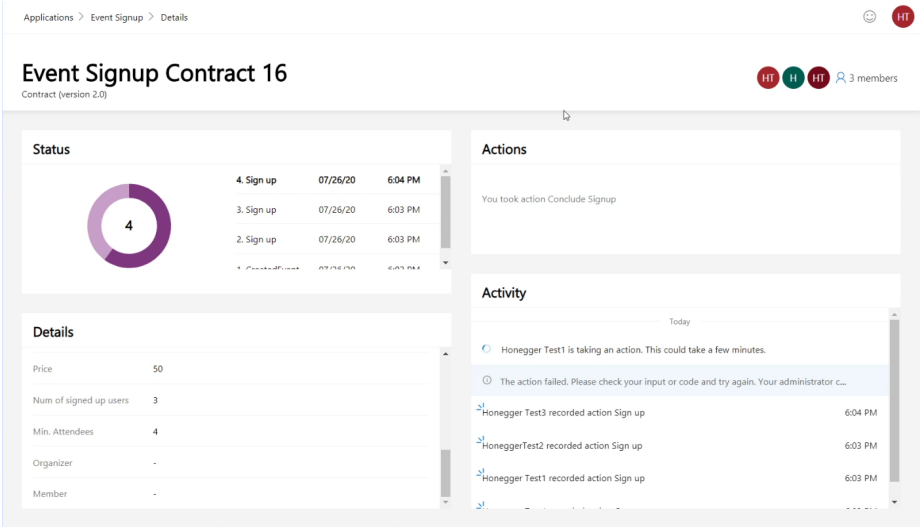

Figure 55: A second attempt of concluding the vote after the deadline has passed.

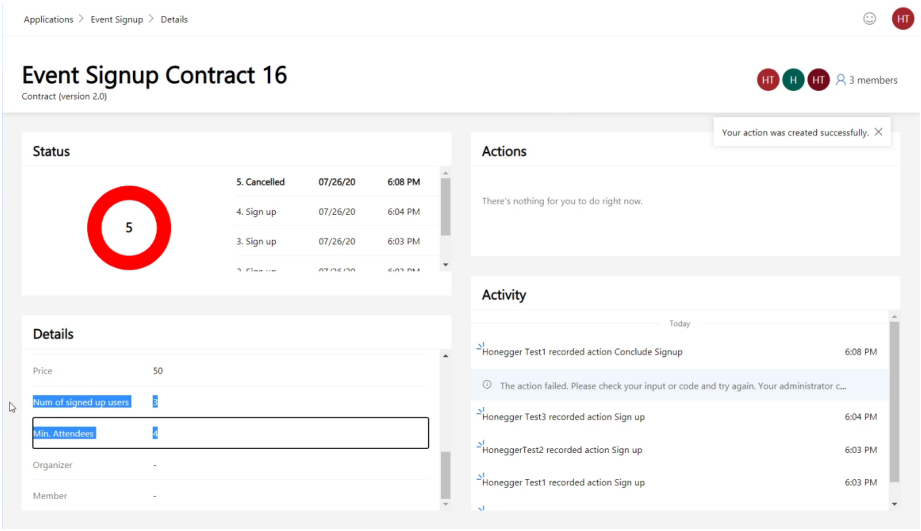

Figure 56: The number of signed up users is smaller than the minimum number of attendees, so the Smart Contract ends up in the "Cancelled" state.

6.3.2 Successful process example

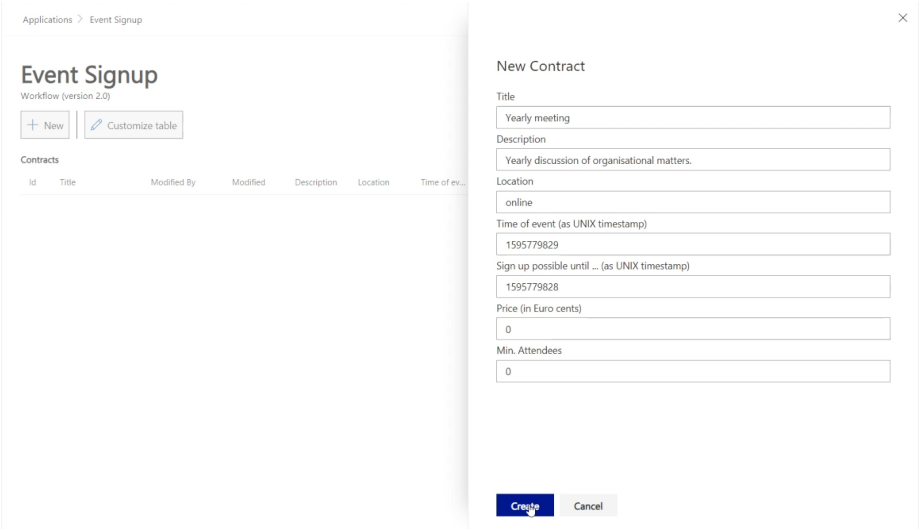

Figure 57: This GUI shows the creation of an event Smart Contract without a minimum number of attendees.

The next event sign up demonstration features a Smart Contract which does not have a minimum number of attendees - it will take place either way (Fig. 57).
Again, users start to sign up for the event (Fig. 58) and trying to sign up twice (with the same user account) fails as planned (Fig. 59). It is also possible to unsubscribe from an event (Fig. 60) if a user decides not to attend it. This is only possible before the deadline is reached, and it will result in decrementing the "Num of signed up users" variable by one. Next, a third user decides to sign up (Fig. 61). Since one user has unsubscribed, we now have two signed up users in total. Then, the fourth user signs up, increasing the number of users attending the meeting to three(Fig. 62). Finally, the organizer decides to conclude the sign up process(Fig. 63) once the deadline was reached (trying to conclude it early failed as expected, as can be seen in Fig. 64), and the process terminates successfully in a "Finished" state.

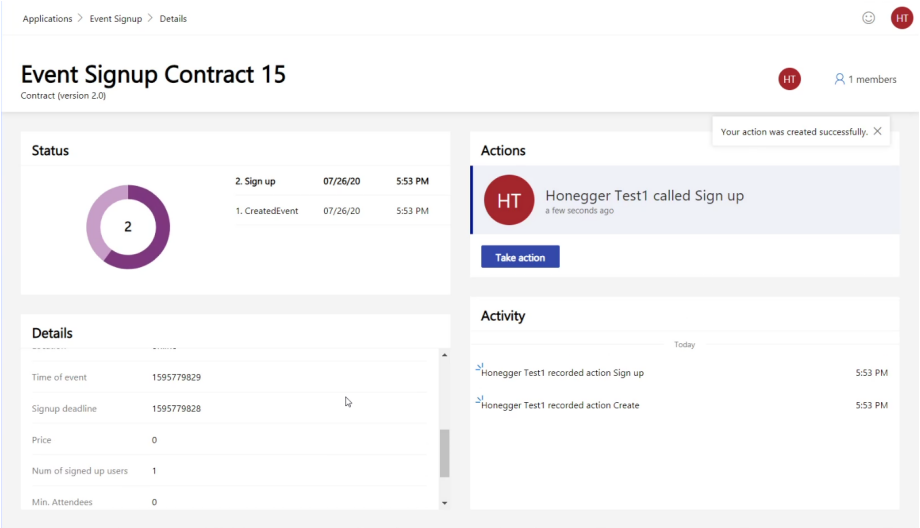

Figure 58: User 1 (HoneggerTest1, see table 2) signed up to the event.

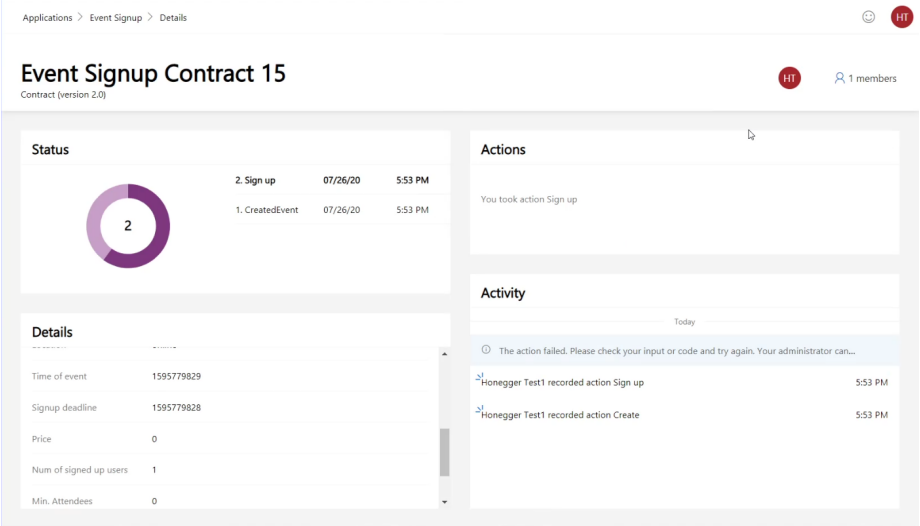

Figure 59: Signing up twice to the same event is not possible.

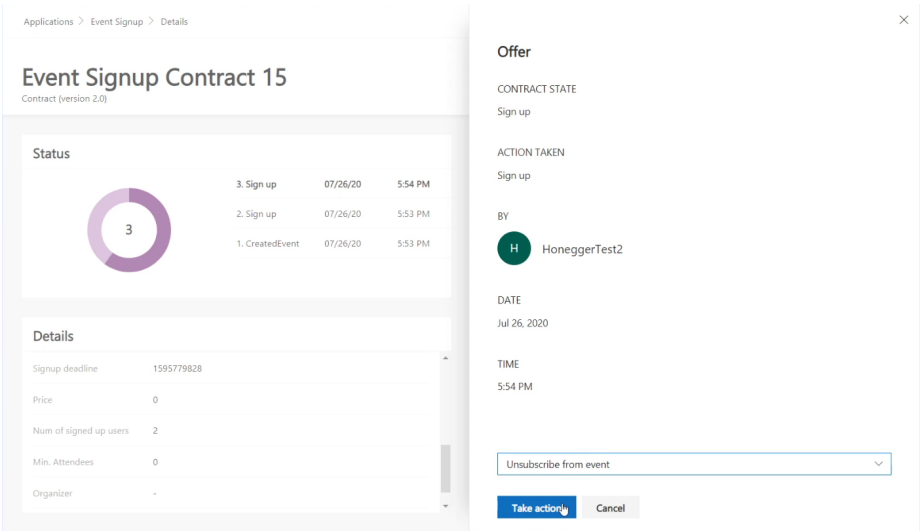

Figure 60: User 2 has decided to not attend the event.

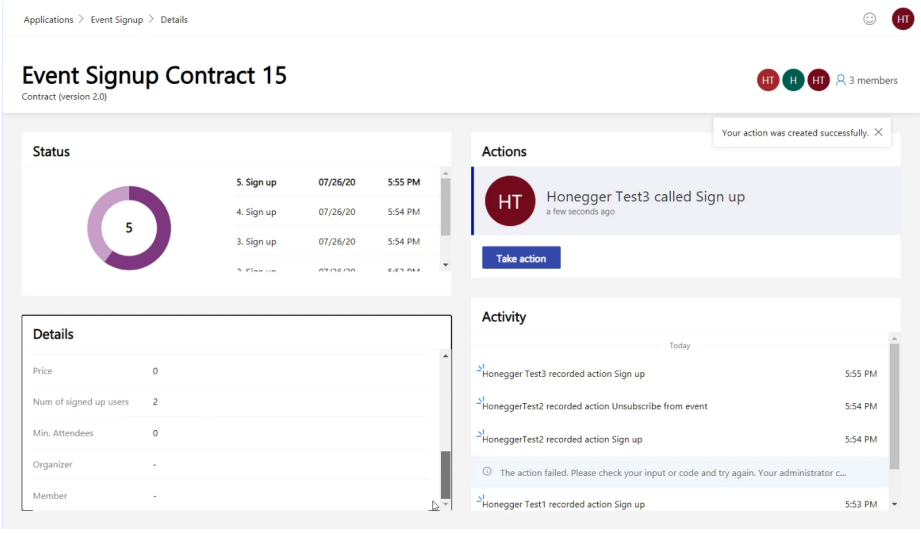

Figure 61: In total, two users attend the event.

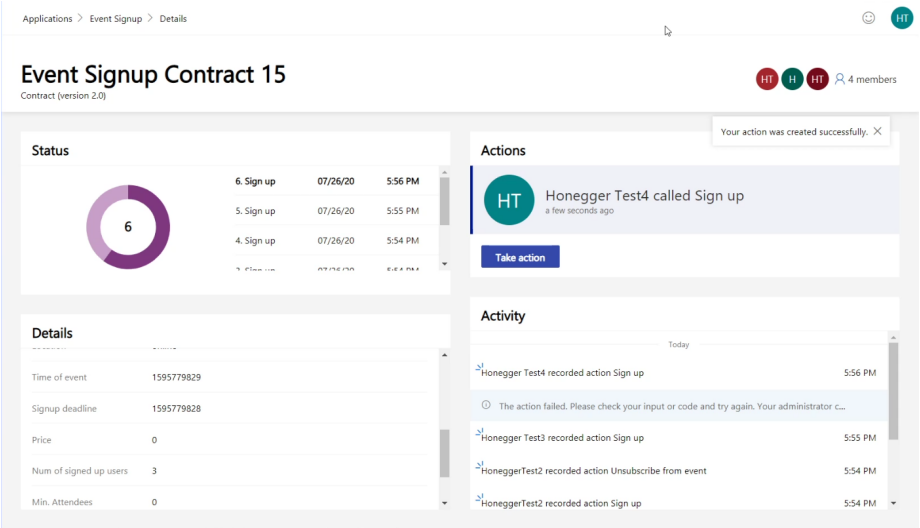

Figure 62: Four users have signed up, and one user decided to not attend the event.

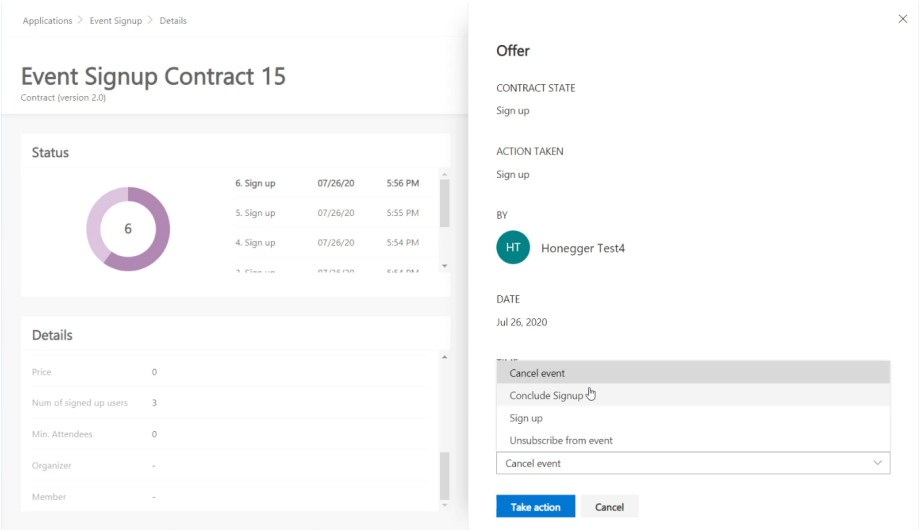

Figure 63: Organizers can create events, sign up to them, unsubscribe from them and cancel events.

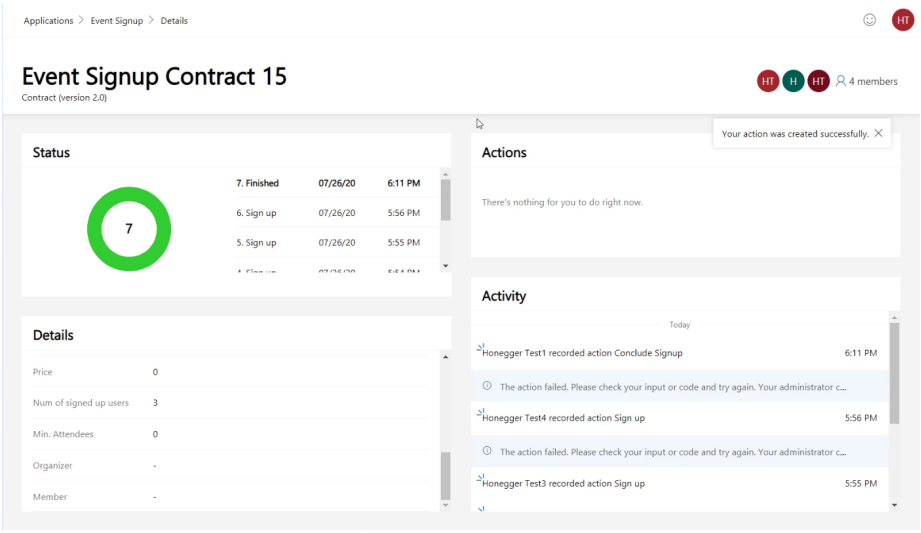

Figure 64: An organizer concludes the sign up process successfully.

6.4 Simultaneous voting experiment

For this implementation, it is important to test whether the voting process is consistent or not. Users should e.g. not be able to vote twice or more times by voting on multiple devices at the same time. Also, it should be possible for multiple different users to vote at the same time, despite the global lock provided by the "voteInProgress" boolean variable.

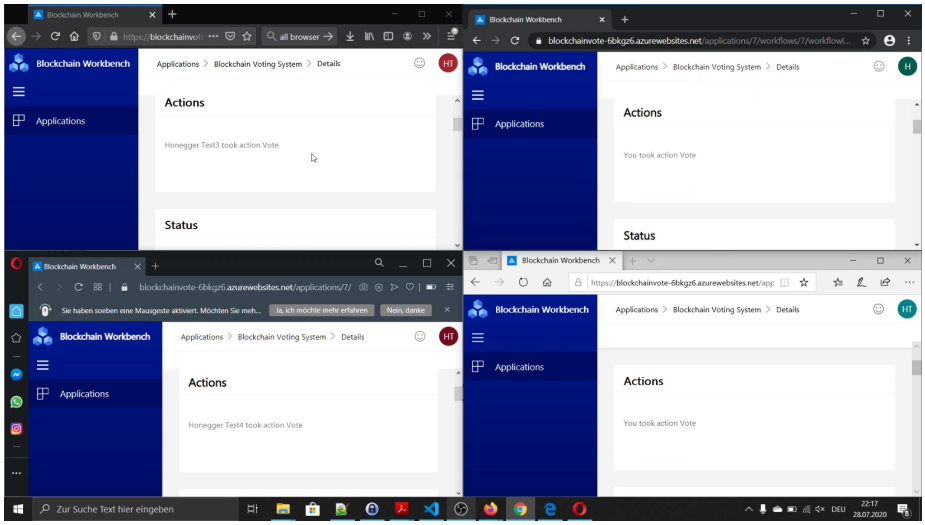

Figure 65: The ABW UI of the same user in four different browsers

We have tested this by logging into four different browsers at the same time - Firefox, Chrome, Edge and Opera (see Fig. 65). As a side note, the ABW website did not load in the Safari and Internet Explorer browsers. Then, we voted for the four different test users (see table 2) at the same time. This resulted in all four votes getting processed and confirmed by the blockchain. Their vote was counted, and the voting could be concluded.
Next, we logged into the first test user account in all four browsers and voted four times at the same time. As expected, only the first vote was counted, and three attempts failed because user has already voted.

Since there are limitations as to how simultaneously a human user can vote with four accounts (it took about two seconds to click the four buttons), we decided to test the implementation with a Python script using Selenium. The code of this script as well as its detailed description can be found in chapter 5.4.

In Fig. 66, we can see four different (simulated) users who are prepared to vote. Once the exact time at which they should vote is reached, then they will

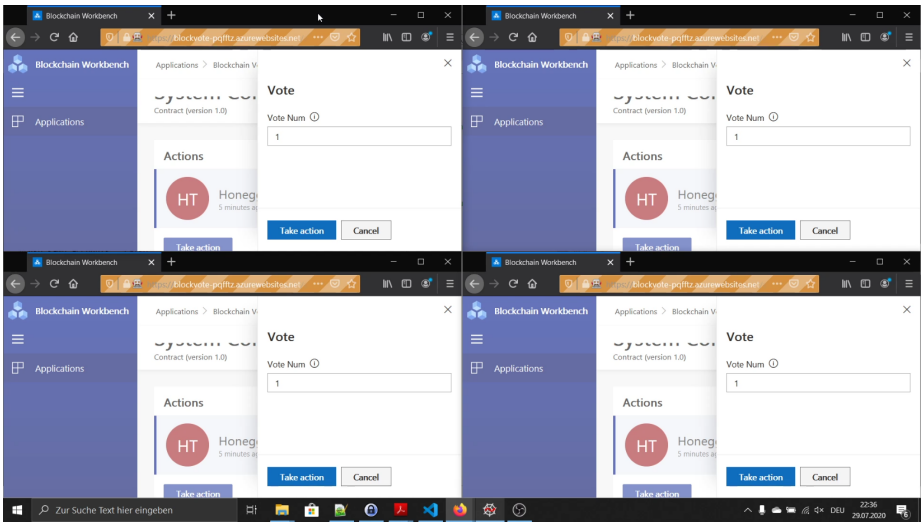

Figure 66: The ABW UI of four users in four different browser windows

simultaneously press the "take action" button and their vote will be registered by the blockchain. In my example, this did work and all four votes were cast in a span of 0.018 seconds. All votes were correctly recognized by the BDSS.

Next, we are testing if one user can be logged in their account in four different browsers and cast a vote in all four browsers at the same time. It should not be possible to abuse the system by voting at the same time with the same account - only one of these votes should be counted. In Fig. 67, we can see that the same user (Test user 1) was able to vote once and the three other attempts were rejected. The votes were cast in a span of 0.023 seconds. It was correctly counted as only one vote and three failed voting attempts.

Next, we are repeating this experiment multiple times using the Python script to see if the voting still concludes successfully. First, we are simultaneously voting ten times with the same user. The results can be found in table 3. In this table, the times at which the vote button was pressed can be seen in the "Time 1" to "Time 4" variables, which are Unix timestamps.
The focus of the table is the absolute difference - it is the time it has taken between the first and the last press of the vote buttons in the different browser windows. The average absolute difference among all ten test runs is 0.01705244 seconds - less than two hundredths of a second. All of the test runs were successful, meaning that only one of the votes was counted, whereas the other three votes resulted in failed attempts.

Now, we are testing if the same simultaneous voting experiment will work

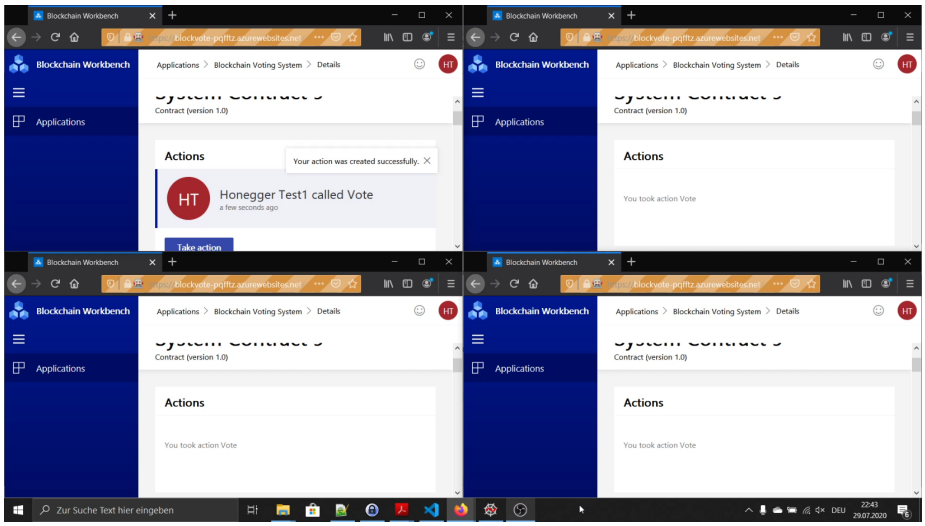

Figure 67: The ABW UI of the same user in four different browser windows

ten times for four different users. In order to get a successful test run here, all votes need to be successfully registered by the blockchain. The times at which the users voted (Unix timestamps) as well as their time differences can be seen in table 4. This time, the average absolute difference among all test runs clocks in at 0.02244579 seconds. All of the test runs were successful for the test users - their decisions validated correctly.
In both experiments, four users could successfully take part in the decision making process in a range of approximately 20 milliseconds. The results of these experiments suggest that a large number of users can take part in the system without the system failing or blocking the vote functionality. This is especially true considering that in a typical voting, the deadline will occur not on the same day, but a couple days or weeks later. Thus, this DApp solution may be a suitable prototype for even larger organisations and companies.

Same User Simultaneous Voting						
Test Run	Time 1	Time 2	Time 3	Time 4	Absolute Difference	Successful
1	1597342250.1805558	1597342250.182549	1597342250.1915588	1597342250.1945217	0.0139659	Yes
2	1597342525.1571655	1597342525.169135	1597342525.1711297	1597342525.1791139	0.0219484	Yes
3	1597342675.5142796	1597342675.5172343	1597342675.5242467	1597342675.5272076	0.012928	Yes
4	1597342824.8921382	1597342824.8961315	1597342824.9041445	1597342824.9080944	0.0159562	Yes
5	1597343107.9649386	1597343107.9699259	1597343107.9749134	1597343107.9808981	0.0159595	Yes
6	1597343250.5892835	1597343250.5962296	1597343250.603205	1597343250.609187	0.0199035	Yes
7	1597343533.7266111	1597343533.7325997	1597343533.7345903	1597343533.736638	0.0100269	Yes
8	1597343733.4098368	1597343733.4327729	1597343733.4417508	1597343733.444743	0.0349062	Yes
9	1597343880.8270302	1597343880.835008	1597343880.8389957	1597343880.8429852	0.015955	Yes
10	1597344032.5144334	1597344032.5174243	1597344032.5194166	1597344032.5234082	0.0089748	Yes

Table 3: Voting with the same user at the same time

Different Users Simultaneous Voting						
Test Run	User 1	User 2	User 3	User 4	Absolute Difference	Successful
1	1597344965.192553	1597344965.1905587	1597344965.2065163	1597344965.2015297	0.0159576	Yes
2	1597345244.1919932	1597345244.1899967	1597345244.2139342	1597345244.2030077	0.0239375	Yes
3	1597345392.7192261	1597345392.715233	1597345392.698244	1597345392.7002387	0.0209821	Yes
4	1597345542.169464	1597345542.1555007	1597345542.164481	1597345542.1834228	0.0279221	Yes
5	1597345694.6158326	1597345694.6178265	1597345694.6238112	1597345694.626803	0.0109704	Yes
6	1597345858.5809262	1597345858.5919166	1597345858.5839143	1597345858.5869067	0.0109904	Yes
7	1597346009.17513	1597346009.1771245	1597346009.1831086	1597346009.1970718	0.0219418	Yes
8	1597346155.2336063	1597346155.239591	1597346155.2356012	1597346155.2465742	0.0129679	Yes
9	1597346351.1901324	1597346351.1582172	1597346351.171183	1597346351.1661973	0.0319152	Yes
10	1597346507.720729	1597346507.715748	1597346507.755636	1597346507.7087631	0.0468729	Yes

Table 4: Voting with four different users at the same time

7 Overall Conclusions

Advantages offered by the Ethereum technology for the implementation of a distributed organization software are its widespread use, consistency, traceability and provision of smart logic. More decentralization, provided by independent nodes outside of the cloud service solution, could potentially increase the trust users and organisations have in the results of electronic decision making systems.
The Smart Contract based application implemented in this thesis runs on a permissioned distributed ledger where each user does have an Ethereum wallet and can participate once in every election. It also enables users to sign up to events which were created by organizers. The implementation runs on the Microsoft Azure cloud and uses the ABW, which also provides a user-friendly user interface. Security and privacy of the application are provided by the implemented functionality as well as by the fact that the blockchain is closed off for third parties and unauthorized users. The simultaneous voting experiment shows that the implementation can handle its intended workload.

8 Limitations and Future Work

The capabilities and limitations of the implementation are defined by its core aspects: The level of privacy and decentralization offered by the DApp, as well as the level of trust which users can have in the decision making process. Further, service costs of running the distributed system in a cloud environment and the number of users which the service can handle concurrently must be considered.

Large-scale usage The prototype described in this example is designed for organisations or companies, where the number of users is limited to a few hundreds or thousands of users. While the experiment in chapter 6.4 suggests that this system may be able to handle such a workload, it is not made e.g. for national elections with millions of voters.

Privacy The amount of privacy provided to users could be increased using functionality which is provided or may be developed for the Quorum platform in the future. Zero-knowledge-proof procedures or blind signatures could improve the level of privacy a user has in a blockchain-based system.

Decentralization The level of decentralization provided by a cloud solution approach is limited, and to be more exact, it is not a decentralized solution at all, but one managed by an entity, e.g. an organisation, company or institution. This is the point of blockchain cloud solutions, but in future implementations, it may be possible to run additional nodes independently from the cloud solution,

increasing the level of decentralization.

Service costs It should be considered that blockchain solutions tend to have relatively high costs compared to more traditional approaches. This can include processing power, electricity costs, or in the case of Blockchain-as-a-Service solutions in a cloud environment, the service costs required to the customer. For example, running an ABW instance for a month leads to an estimated cost per month of $143.16[18]. In this example, we used the "Basic" price tier of the workbench, which is intended for development and testing of applications, comes with one vCore, five GB of storage space, one validator node and one transaction node. The "Standard" price tier, which is made to run production workloads and comes with two vCores as well as two validator nodes, is even more expensive at an estimated $897.58 per month. Also, the increased complexity of using a blockchain implementation must be considered. A blockchain solution should therefore only be used if the requirements of the implementation make it absolutely necessary.

When counting only days in which the service ran for 24 hours a day (excluding

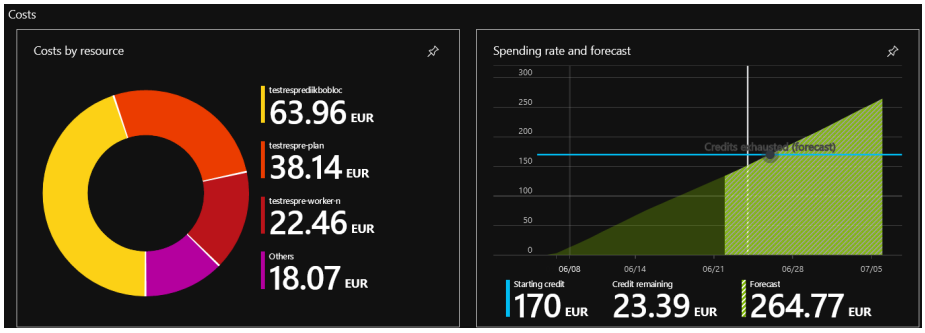

Figure 68: Costs accumulated while running the ABW. Screenshot of the Azure Portal website.

the day of creation and termination of the ABW service), we have accumulated costs of €139.01 in 16 days; between June 8th and June 23rd (see Fig. 68). On average, it did cost €8.69 per day to run the service.

User adoption Finally, the prototype implemented for this thesis has not been tested in real-world scenarios with a large number of participants. To reach a wide acceptance of such a system, it needs to be tested by a large number of people in an organisation. This way, it could be possible to prove its usability, practicality and security in an authentic setting. This could be

[18]Region West Europe. Prices may differ in other regions or at a later date.

achieved by testing the BDSS in small, closed groups and gradually widening its deployment to a larger scale, namely the entire company or organisation.

9 Appendix

Below, you can find useful links with information about the ABW as well as a link to the (private) Gitlab which hosts the source code of the Smart Contract implementation as well as the Python script of the simultaneous voting experiment.

9.1 Azure Components and Information

Azure Blockchain Workbench (official page on microsoft.com):
https://azure.microsoft.com/en-us/features/blockchain-workbench/

ABW documentation:
https://docs.microsoft.com/de-de/azure/blockchain/workbench/

ABW example implementations:
https://github.com/Azure-Samples/blockchain

ABW deployment instructions:
https://docs.microsoft.com/en-us/azure/blockchain/workbench/deploy

9.2 Source Code URL

This QR code is the link to the Smart Contract source code on Gitlab.
Source code URL: https://git01lab.cs.univie.ac.at/a1127222/blockchain-based-business-decision-support-system.

References

[Rouhani and Deters (2017)] S. Rouhani and R. Deters. Performance analysis of ethereum transactions in private blockchain. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 70–74, Nov 2017. doi: 10.1109/ICSESS.2017.8342866.

[Rouhani and Deters (2019)] S. Rouhani and R. Deters. Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access*, 7, 2019.

[Lange et al. (2019)] Maik Lange, Steven Chris Leiter, and Rainer Alt. Defining and delimitating distributed ledger technology: Results of a structured literature analysis. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 43–54, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Iqbal and Matulevičius (2019)] Mubashar Iqbal and Raimundas Matulevičius. Comparison of blockchain-based solutions to mitigate data tampering security risk. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 13–28, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Kakarott et al. (2019)] Julian Kakarott, Katharina Zeuch, and Volker Skwarek. License chain - an identity-protecting intellectual property license trading platform. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 29–42, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Meroni et al. (2019)] Giovanni Meroni, Pierluigi Plebani, and Francesco Vona. Trusted artifact-driven process monitoring of multi-party business processes with blockchain. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 55–70, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Köpke et al. (2019)] Julius Köpke, Marco Franceschetti, and Johann Eder. Balancing privity and enforceability of bpm-based smart contracts on blockchains. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Block-*

chain and Central and Eastern Europe Forum, pages 87–102, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Haarmann et al. (2019)] Stephan Haarmann, Kimon Batoulis, Adriatik Nikaj, and Mathias Weske. Executing collaborative decisions confidentially on blockchains. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 119–135, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Schäffer et al. (2019)] Markus Schäffer, Monika di Angelo, and Gernot Salzer. Performance and scalability of private ethereum blockchains. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 103–118, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Klinkmüller et al. (2019)] Christopher Klinkmüller, Alexander Ponomarev, An Binh Tran, Ingo Weber, and Wil van der Aalst. Mining blockchain processes: Extracting process mining data from blockchain applications. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Štemberger, Andrea Kő, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 71–86, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30429-4.

[Pierro and Rocha (2019)] G. A. Pierro and H. Rocha. The influence factors on ethereum transaction fees. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 24–31, May 2019. doi: 10.1109/WETSEB.2019.00010.

[García-Bañuelos et al. (2017)] Luciano García-Bañuelos, Alexander Ponomarev, Marlon Dumas, and Ingo Weber. Optimized execution of business processes on blockchain. 09 2017. doi: 10.1007/978-3-319-65000-5_8.

[Kirkman and Newman (2018)] S. Kirkman and R. Newman. A cloud data movement policy architecture based on smart contracts and the ethereum blockchain. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 371–377, April 2018. doi: 10.1109/IC2E.2018.00071.

[Nayak et al. (2018)] S. Nayak, N. C. Narendra, A. Shukla, and J. Kempf. Saranyu: Using smart contracts and blockchain for cloud tenant management. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 857–861, July 2018. doi: 10.1109/CLOUD.2018.00121.

[Khoury et al. (2018)] D. Khoury, E. F. Kfoury, A. Kassem, and H. Harb. Decentralized voting platform based on ethereum blockchain. In *2018*

IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), pages 1–6, Nov 2018. doi: 10.1109/IMCET.2018.8603050.

[Yavuz et al. (2018)] E. Yavuz, A. K. Koç, U. C. Çabuk, and G. Dalkılıç. Towards secure e-voting using ethereum blockchain. In *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–7, March 2018. doi: 10.1109/ISDFS.2018.8355340.

[Ranganthan et al. (2018)] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov. A decentralized marketplace application on the ethereum blockchain. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 90–97, Oct 2018. doi: 10.1109/CIC.2018.00023.

[Wang et al. (2018)] X. Wang, X. Zha, G. Yu, W. Ni, R. P. Liu, Y. J. Guo, X. Niu, and K. Zheng. Attack and defence of ethereum remote apis. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, Dec 2018. doi: 10.1109/GLOCOMW.2018.8644498.

[Ciatto et al. (2018)] G. Ciatto, S. Mariani, and A. Omicini. Blockchain for trustworthy coordination: A first study with linda and ethereum. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 696–703, Dec 2018. doi: 10.1109/WI.2018.000-9.

[Pintado et al. (2017)] Orlenys Pintado, Luciano García-Bañuelos, Marlon Dumas, and Ingo Weber. Caterpillar: A blockchain-based business process management system. In *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Management (BPM 2017), At Barcelona, Spain*, 09 2017.

[Zichichi et al. (2019)] Mirko Zichichi, Stefano Ferretti, and Gabriele D'Angelo. A distributed ledger based infrastructure for smart transportation system and social good. *ArXiv*, abs/1910.03280, 2019.

[Wang et al. (2019a)] S. Wang, R. Pei, and Y. Zhang. Eidm: A ethereum-based cloud user identity management protocol. *IEEE Access*, 7, 2019a.

[Sukhodolskiy and Zapechnikov (2018)] I. Sukhodolskiy and S. Zapechnikov. A blockchain-based access control system for cloud storage. In *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 1575–1578, Jan 2018. doi: 10.1109/EIConRus.2018.8317400.

[Wang et al. (2019b)] S. Wang, X. Wang, and Y. Zhang. A secure cloud storage framework with access control based on blockchain. *IEEE Access*, 7, 2019b.

[Huang et al. (2019)] B. Huang, L. Jin, Z. Lu, X. Zhou, J. Wu, Q. Tang, and P. C. K. Hung. Bor: Toward high-performance permissioned blockchain in rdma-enabled network. *IEEE Transactions on Services Computing*, pages 1–1, 2019. ISSN 2372-0204. doi: 10.1109/TSC.2019.2948009.

[Branco et al. (2015)] Frederico Branco, Ramiro Gonçalves, José Martins, and Manuel Pérez Cota. Decision support system for the agri-food sector – the sousacamp group case. In Alvaro Rocha, Ana Maria Correia, Sandra Costanzo, and Luis Paulo Reis, editors, *New Contributions in Information Systems and Technologies*, pages 553–563, Cham, 2015. Springer International Publishing. ISBN 978-3-319-16486-1.

[de la Torre-Díez et al. (2014)] Isabel de la Torre-Díez, Borja Martínez-Pérez, Miguel López-Coronado, Javier Rodríguez Díaz, and Miguel Maldonado López. Decision support systems and applications in ophthalmology: Literature and commercial review focused on mobile apps. *Journal of Medical Systems*, 39(1):174, Dec 2014. ISSN 1573-689X. doi: 10.1007/s10916-014-0174-2. URL `https://doi.org/10.1007/s10916-014-0174-2`.

[Wang (2010)] F. Wang. Application research of an intelligent decision support system based on data warehousing technology. In *2010 International Conference on E-Business and E-Government*, pages 1773–1776, May 2010. doi: 10.1109/ICEE.2010.448.

[Carter and Fielden (2001a)] Craig Carter and Kay Fielden. Towards cyberdemocracy: True representation. In Won Kim, Tok-Wang Ling, Yoon-Joon Lee, and Seung-Soo Park, editors, *The Human Society and the Internet Internet-Related Socio-Economic Issues*, pages 285–298, Berlin, Heidelberg, 2001a. Springer Berlin Heidelberg. ISBN 978-3-540-47749-5.

[Carter and Fielden (2001b)] Craig Carter and Kay Fielden. Towards cyberdemocracy: True representation. In Won Kim, Tok-Wang Ling, Yoon-Joon Lee, and Seung-Soo Park, editors, *The Human Society and the Internet Internet-Related Socio-Economic Issues*, pages 285–298, Berlin, Heidelberg, 2001b. Springer Berlin Heidelberg. ISBN 978-3-540-47749-5.

[Watson and Cordonnier (2002)] Anthony Watson and Vincent Cordonnier. *Voting in the New Millennium: eVoting Holds the Promise to Expand Citizen Choice*, pages 234–239. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-46138-8. doi: 10.1007/3-540-46138-8_38. URL `https://doi.org/10.1007/3-540-46138-8_38`.

[Beckert and Beuster (2006)] Bernhard Beckert and Gerd Beuster. A method for formalizing, analyzing, and verifying secure user interfaces. In Zhiming Liu and Jifeng He, editors, *Formal Methods and Software Engineering*, pages 55–73, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-47462-3.

[Antoniou et al. (2007)] A. Antoniou, C. Korakas, C. Manolopoulos, A. Panagiotaki, D. Sofotassios, P. Spirakis, and Y. C. Stamatiou. A trust-centered approach for building e-voting systems. In Maria A. Wimmer, Jochen Scholl, and Åke Grönlund, editors, *Electronic Government*,

pages 366–377, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74444-3.

[Mei (2009)] *eDemocracy*, pages 163–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-00130-7. doi: 10.1007/978-3-642-00130-7_8. URL https://doi.org/10.1007/978-3-642-00130-7_8.

[Meier (2012)] Andreas Meier. *eDemocracy*, pages 149–168. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-24494-0. doi: 10.1007/978-3-642-24494-0_8. URL https://doi.org/10.1007/978-3-642-24494-0_8.

[Meier (2009)] Andreas Meier. Elektronische abstimmungen und wahlen. *HMD Praxis der Wirtschaftsinformatik*, 46(1):51–61, Feb 2009. ISSN 2198-2775. doi: 10.1007/BF03340325. URL https://doi.org/10.1007/BF03340325.

[Gasser and Gerlach (2011)] Urs Gasser and Jan Gerlach. *Electronic Voting: Approaches, Strategies, and Policy Issues—A Report from Switzerland*, pages 101–128. T. M. C. Asser Press, The Hague, The Netherlands, 2011. ISBN 978-90-6704-731-9. doi: 10.1007/978-90-6704-731-9_7. URL https://doi.org/10.1007/978-90-6704-731-9_7.

[Prosser (2012)] Alexander Prosser. eparticipation – did we deliver what we promised? In Andrea Kő, Christine Leitner, Herbert Leitold, and Alexander Prosser, editors, *Advancing Democracy, Government and Governance*, pages 10–18, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32701-8.

[Meier (2020)] Andreas Meier. *Blockchain-Voting für MyPolitics und OurPolitics*, pages 337–353. Springer Fachmedien Wiesbaden, Wiesbaden, 2020. ISBN 978-3-658-28006-2. doi: 10.1007/978-3-658-28006-2_16. URL https://doi.org/10.1007/978-3-658-28006-2_16.

[Yu et al. (2018)] Bin Yu, Joseph K. Liu, Amin Sakzad, Surya Nepal, Ron Steinfeld, Paul Rimba, and Man Ho Au. Platform-independent secure blockchain-based voting system. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *Information Security*, pages 369–386, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99136-8.