University of Zurich^UZH

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

ReM: A Blockchain-based Reputation System for Infrastructure Providers

Dominik Jurilj
Zürich, Switzerland
Student ID: 13-924-220

Supervisor: Muriel Franco, Eder Scheid
Date of Submission: September 2, 2020

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

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

Abstract

Reputation systems have become a widespread method for representing perceived quality and setting expectations on a product or a service. With the rising popularity of blockchain technologies and the growing possibilities of Smart Contracts, there is an ever increasing number of decentralized reputation systems in different areas of research. While Infrastructure-as-a-Service (IaaS) providers have gained a considerable market share in the last years, a general lack of platforms distributing important information about their services to potential customers is encountered. This thesis presents the Reputation Manager (ReM), a decentralized reputation system built on a reverse auction system for IaaS providers, which allows for a transparent communication from customers of such IaaS (to both other customers and the IaaS providers) about multiple prevalent quality factors of the services. The thesis discusses multiple design choices of well-known reputation systems and includes arguments for advantages and opportunities in the decentralization thereof. Furthermore, it includes a theoretical solution for establishing trustworthy and controlled interactions within such a system and showcases a prototypical implementation in the form of the ReM. The proof of concept is reinforced by multiple real-world use cases, which are further analysed for their costs, limitations, and practical feasibility. A discussion on multiple additional possibilities and features of the ReM, as well as on the classification of quality for IaaS providers, concludes this thesis.

Contents

Chapter 1

Introduction

With the widespread popularity and access to mainstream internet, online shopping has become a big part of daily life as well [26]. Businesses such as Amazon [1] and eBay [13] have prospered and make billions of US-Dollars of annual revenue [53, 52]. However, physical products are not the only goods traded online since the internet has nearly endless opportunities for other markets as well. With increasingly complex systems, applications, and files, issues in the form of weak performance, insufficient storage space, security concerns, and many more may arise for the average consumer. Instead of upgrading or building and managing their own systems, customers have the option to use external computation power or storage space by sourcing them from Infrastructure-as-a-Service providers (infrastructure providers) [29]. With many businesses seeking to get a share in an increasingly saturated market, customers may need help to make the right choice between multiple ones available and seemingly identical products or providers. To solve this, many online retailers use some kind of reputation system for their products, which makes it easier for customers to decide on which product might be best suited for their needs [73]. Furthermore, there are websites specifically dedicated to either collecting reviews or ratings, or simply writing reviews about different products to help project a perceived quality about the product (e.g., Techradar [56]).

With such easily accessible and centralized platforms (i.e., platforms managed and possibly tampered by a central authority), security and trustworthiness may become an issue [70]. One possibility to move away from potentially conflicted reputations [51] is to move away from centralized solutions for reputation systems by using the newer blockchain technology [7]. Decentralized systems have become a widespread method of adding credibility to information by storing immutable transaction details across all nodes of the system. This decentralization opens different new opportunities of handling all sorts of data and establishing trust in the form of contracts. There is a steadily increasing number of solutions for decentralized reputation systems for various areas such as e-payment systems [46] or vehicular networks [85, 65]. With this rising amount of works on decentralized reputation systems, it becomes evident that a blockchain-based reputation can implement features that centralized reputation systems struggle with. The goal of this thesis is to provide insight on advantages of decentralized reputation systems and show an implementation for a blockchain-based reputation system for infrastructure providers.

1.1 Motivation

Although there are many different reputation systems in use across many platforms, most of which are centralized solutions for product reviews such as the one implemented on Amazon [47]. New approaches using decentralized reputation systems could turn out useful in various topics across research, such as vehicular networks (reputation in inter-vehicle communication) [85], more specialized systems such as platforms for job searching [50, 74], and many other. To the best of our knowledge, there are currently no solutions for a decentralized reputation system for infrastructure providers. Furthermore, most online collections showcasing reputation or reviews for infrastructure providers appear to be privately handled (i.e., a web magazine writing a review such as PCMag [77] or Techradar [56]) in contrast to having a number of users leaving a reputation score and a review (e.g., G2 [26]). The lack of the latter may be due to general issues with centralized reputation systems, such as a lack of verification (e.g., how can a third-party website know if a user has used the rated service?). This opens opportunities for a blockchain-based reputation system where users can monitor and be monitored by other users and the system can set restrictions by using Smart Contracts in general. For the case of reputation for infrastructure providers, this may as well be an opportunity to investigate what the reputation score should contain, what an expected service should resemble, and how the reputation may show the highest integrity possible. Finally, the BRAIN [60], which was developed at the University of Zurich by the Communication Systems Group, can be used for further purposes by including a reputation system that considers the mentioned aspects. Thus, this thesis aims to make first advances in this research area and to serve as a steppingstone for future theses about the same topic.

1.2 Thesis Goals

This thesis focuses on the creation of a decentralized reputation system for infrastructure providers, where the goal is to ensure transparency, reliability, and integrity to both the providers and the customers. The customers should be able to leave an authentic rating, whereas other users can trust said customer to be a real user that is not forced or influenced to fake ratings in a malicious manner. To achieve this, an analysis on existing (decentralized) reputation systems shall be conducted and discussed. Important factors for rating an infrastructure provider will be considered and summarized, of which the most prevalent ones shall be used in the reputation system. The prototype of the reputation system should ensure that it contains a design that is as close to the current best-practices as possible. Furthermore, it should be built upon the reverse auction system found in the BRAIN and include functionalities such as creating, holding, and ending an auction (for the auctioneer), bidding in an auction (for the provider), showing an active auction and their bids, viewing providers and their reputations, viewing individual ratings of a provider, and creating a new rating for a provider. All of the functionalities should be applicable over a customized user interface. To show the feasibility of the solution and the prototype, multiple use cases of the ReM shall be showcased and discussed for both successes and limitations.

1.3 Thesis Outline

The thesis will go over an introduction to the thematic (Blockchain, Smart Contracts, etc.) in Chapter 2. Concepts such as blockchains, Smart Contracts, and reputation systems are discussed. It is followed up with an introduction and explanation to the BRAIN, a more in-depth analysis on both centralized and decentralized reputation systems, and the security concerns thereof in Chapter 3. A solution is showcased in Chapter 4. The solution goes over various aspects and design decisions for creating a decentralized reputation system for infrastructure providers. A prototypical implementation thereof is shown in Chapter 5, of which multiple use cases and visualizations are presented in Chapter 6. An economic analysis and a discussion on various limitations, both design-wise as well as technology-wise, are included in the evaluation as well. Finally, conclusions and possibilities for future works are a part of Chapter 7.

Chapter 2

Background

2.1 Blockchain and Smart Contracts

Blockchain [7] is a decentralized distributed ledger technology [71] used to store immutable data that is openly accessible by any participant in the blockchain. While the blockchain technology is still rather young, there are already multiple popular platforms such as Ethereum [16], Hyperledger Fabric [23], and IBM Blockchain [25]. However, when generally talking about blockchain technology nowadays, it is (apart from "crypto mining" [58]) mostly associated with digital wallets and cryptocurrencies such as Bitcoin [7, 6]. According to [7], there are currently over 47 million unique wallets across 140 countries, with over 100 million completed transactions. The popularity is certain to rise, as many big companies such as IBM and more than 90% of European banks are researching and investing into blockchain options [55].

The popularity, especially in finance, comes largely from the design principles of blockchains. One of the main goals of blockchain systems is to ensure that a transaction between two parties is immutable, verifiable, secure, resilient, and transparent [86]. While Satoshi Nakamoto's (the anonymous inventor of Bitcoin [39]) Bitcoin Whitepaper [76] does not specifically mention decentralization as part of the core design, current understandings of state-of-the-art blockchain technology primarily point towards it being the most important factor for the aforementioned goals [86]. The key point is moving away from a traditional centralized system, in which one company or server holds all the data about the service specific data [45]. The decentralization of data changes major points in how communication and information is handled in general. Blockchain's structure is mainly comprised of blocks, nodes, and transactions. A single block contains a multitude of immutably stored transactions, where the size of the block determines the maximum number of transactions stored [86]. Blocks are stored on nodes (e.g., computers, servers, or other similarly functional devices) and all nodes are connected to each other, which allows them to constantly exchange the latest status of the blockchain data. Each node always stores a copy of an accessible version of the current blockchain, i.e., the currently longest blockchain [78, 81, 61].

For a transaction to take place, a definitive way to ensure "trust" between the parties partaking in the transaction is needed. Naturally, the nodes cannot simply assume integrity of every single participant in the blockchain, which makes some form of contract a requirement to bind both parties into a verifiable and transparent transaction, which, by design of blockchain technology, every other node indirectly participates in by observation [86]. Every transaction is ultimately stored into a block, often called a public or digital ledger. To ensure trust in transactions in blockchain technology, every party in a transaction has to be able to leave a signature to make it credible, verifiable, and persistent. Since there are many transactions per blockchain happening, there is a need of some sort of automation in order to improve resource management and maintain sufficient quality such as high speed, low costs, and trustworthy network security. While Bitcoin was first to implement a feature that validates transactions after certain conditions are met, its usage was limited to the currency use case. Ethereum on the other hand is a "Turing-complete" [5] language and allows developers to design their own Smart Contracts, which expands the use of Smart Contracts to more than financial ones [64]. These Smart Contracts implemented by the more refined (i.e., not overly specialized) blockchain technologies are able to provide a trustworthy, safe, and fast transaction, which do not require a middleman or witness for communication and signing transactions [80]. Similarly to the mentioned validation process in Bitcoin, they require certain developer-defined conditions to be met before allowing a transaction to happen [64]. By considering the basically limitless functionalities that can be added to Smart Contracts in Ethereum, there is huge potential for newly formed business opportunities in areas including healthcare, real estate, reputation systems, and many more [80]. Ethereum's Smart Contract functionality is implemented by Solidity [36], a Smart Contract language specifically created for it. One of the advantages in Solidity is the possibility to include inheritance between its constructs, similar to classes in an object oriented programming language, which allows for a far more efficient use of variables and reduces the amount of code needed in the right circumstances [34].

2.2 Reputation Systems

With the dawn of public Internet access in 1991, the business possibility called *ecommerce* (electronic commerce) became reality. Online shops such as Amazon [1] were founded and started growing in popularity and size, slowly becoming the standard for many lines of commerce [75]. Ecommerce started to transform traditional stores to extend their offer by having their own online shops, in certain cases it even forced smaller private stores lacking the resources to match the convenience of online shops to close [62]. With the number of online shops and their popularity growing, many new problems, both technical and ethical/legal, emerged. While the idea at first might have been meant for convenience, scamming businesses flourished by mostly faking entire websites to look like genuine online stores, which could for example charge money without ever sending the objects, or by scamming through an intermediary site such as eBay [82, 57, 13].

Ultimately, *reputation systems* became a very popular method to protect integrity and make it harder for scamming websites. It appears to be a relatively reliable and beneficial system since it is used by almost any online retailer or service provider. One of the earliest works on reputation systems already defined the importance of longevity of the review data, the open accessibility, and the use of feedback for decision making [79]. A striking design characteristic of websites such as Amazon, eBay, and Google Play Store [20] is that the individual ratings for products are very easily visible and accessible, usually containing a direct link (typically in the form of a hyperlink on the star- or percentage-rating) to the product's ratings. Depending on the platform, rating and/or reviewing of a product might be limited to the users that effectively have bought that specific item, to make integrity more likely.

One of the most important design choices in reputation systems is what kind of ratings they collect and what attributes they may have. As mentioned, the most common method of giving a perceived representation of the quality of a product or service appears to be using a numerical (or a symbol thereof) value, ratio, or percentage (or an equivalent symbol thereof, e.g., stars). Trustpilot [41], one of the largest review platforms in the world, provides a lot of insight on what reviews may mean and how they are calculated [42]. They use a combination of written reviews and a one to five rating system, where each value has a rather clear definition [43]. Considering that online shops such as Amazon, eBay, and many more follow suit, it can be assumed that this kind of solution for reputation systems is currently accepted as the "best practice" available.

2.3 Infrastructure Providers

Infrastructure-as-a-Service has become another attractive market opportunity over the past few years. Products such as Microsoft Azure [28], Amazon AWS [2], and Google Drive [19] have begun to be offered in variably configurable packages, containing a multitude of possible combinations in, e.g., storage space, computational power, security measures, operating system compatibility, and more. Depending on the region the customer lives in, certain providers offer servers situated geographically closer than their competition. While this may not be a crucial point for storage-only related services (e.g., cloud services), it may immensely affect the choice for users who need to fulfil tasks externally with a preferably low latency. Since this infers that significantly large companies (e.g., Amazon) may cover a wider area, both technologically and localization-wise, smaller infrastructure providers find themselves at a disadvantage. Thus, a new blockchain platform dedicated to giving an equal opportunity to every infrastructure provider participating, while providing them the possibility to gain reputation as well, may prove to be a advantageous project in multiple ways.

Chapter 3

Related Work

3.1 BRAIN

The **B**lockchain-based **R**everse **A**uction for **I**nfrastructure Supply in Virtual **N**etwork Functions-as-a-Service (BRAIN) [60] was a project developed by the Communication Systems Group at the University of Zurich. With the Network Functions Virtualization (NFV) becoming an interesting area of attention for both academic and industrial purposes, economic opportunities for offering and distributing Virtual Network Functions (VNF) have surfaced. With this, a way for offering a marketplace to both customers and providers of platforms, that are able to deploy specific VNFs, has been developed in the form of BRAIN. The BRAIN is a reverse auction that is primarily targeted at customers who want to deploy a VNF and do not have the necessary NFV-enabled infrastructure to deploy it on. Furthermore, it is a means for infrastructure providers to offer their services (in the form of Infrastructure-as-a-Service) in a decentralized system by bidding in auctions created by the customer looking for the required specifications. When a certain set of requirements defined by the auctioneer is met by the infrastructure provider, the provider may place their bid on the auction.

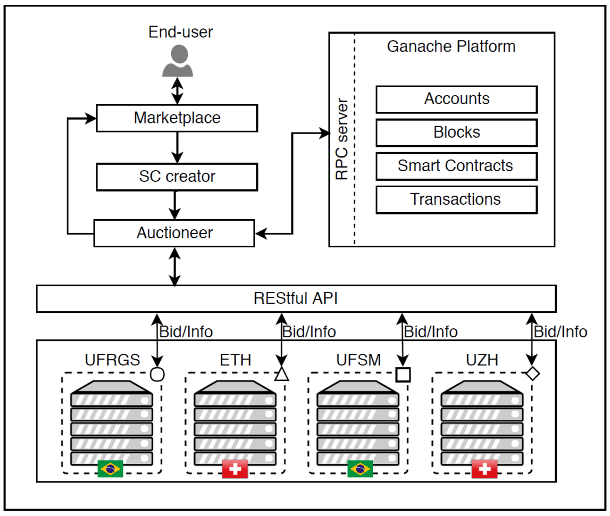


Figure 3.1: Overview of the BRAIN architecture as shown in [60].

The important difference between the standard well-known providers, such as Amazon AWS [3] or Microsoft Azure [29], and the auctions in BRAIN is the specific offering of exact requirements in BRAIN. Where the first might offer the same packages distributed across different categories (i.e., normalized packages in multiple price categories), BRAIN answers the exact demand of the customer. The auctioneer (the user that starts the auction) defines their requirements locally in three sets of information:

- **Info:** VNF name, VNF type, VNF developer, contract period
- **Resources:** memory, disk, CPU cores
- **Priorities:** additional memory, additional disk, additional CPU Cores, latency max, packets per second capacity

Table 3.1: Functions in BRAIN [60].

Name	Attributes	Definition
getInfo()	Smart Contract address	Function that returns a part of the information set (*Info*) mentioned above.
getResources()	Smart Contract address	Function that returns a part of the information set (*Resources*) mentioned above.
getPriorities()	Smart Contract address	Function that returns a part of the information set (*Priorities*) mentioned above.
submitBit()	Smart Contract address, bid value, bidder address	Used by a provider to submit a bid, where the bid is their asking price for the service.
endAuction()	Smart Contract address	Declares that a winner has been selected.
getWinner()	Smart Contract address	Returns the bid value and the address of the winner.

These are used to provide a way for the bidders (infrastructure providers) to first of all check if they can meet the demanded requirements, and second, to calculate the price of such a service if an agreement could be fulfilled. The calculation is done in a localized manner as well since each provider has different prices and may have special offers for situations such as, e.g., a long contract period. Both the opening of an auction, as well as the requisition of information is handled by using Smart Contracts. Furthermore, after collecting bids for a certain amount of time (the auctioneer has the power to choose the point in time), the auction may be ended by the auctioneer and a winner is automatically selected by picking the bidder with the lowest price. BRAIN's functions can be summarized as shown in Table 3.1. For the implementation, BRAIN uses latest technologies such as an Ethereum blockchain [16], Solidity Smart Contract programming language [36], and Python [33] for simulating the start of an auction and placing bids in it.

3.2 Works on Reputation Systems

This section will go over multiple papers analysing and comparing requirements for modern reputation systems. While this thesis will generally take inspiration from newer models and methods, it is to note that the work presented in [79], as mentioned in Section 2.2, was published in the year 2000 and shows multiple characteristics described are well applicable two decades later. They already mention problematic areas such as users having an incentive to vote, the general distrust for "newcomers" (e.g., a new product or a new provider for a service), acquisition of an impactful enough sample size, and how to calculate the final score out of the ratings (e.g., mathematical averages vs. one factor having a higher weight than another). While they note several practical and theoretical issues regarding reviews (and the platforms they are performed on) in general because of said reasons, they still found the early internet-based reputation systems to be performing reasonably well at the time [79].

[63] provided a more in-depth survey about reputation systems in their paper published in 2014, where they both created a taxonomy about existing systems, and proposed a new reference model for context and general interaction for future systems. Their analysis of existing reputation systems implemented by bigger companies and websites provides valuable insight for making requirements for a new system. Particularly their findings regarding the common dimensions in explicit reputation systems, which are the ones implemented with the purpose of providing specific information and estimations about trust between the users and the service or product used. They define multiple points that provide important information about either the voter (e.g., history in the form of past interactions) or the system itself (e.g., context of the vote, collection of counting votes, representation of the votes, aggregation of the reputation score). Apart from the given examples, there are more uncommon points such as (but not exclusively) which entities are targeted, how the system handles interoperability with other systems, and if there is an authority to govern the reputation system.

Since this thesis will propose a solution for a decentralized reputation system, further discussion will contain works about decentralized reputation. The additional dimension of the reputation system being deployed by using a blockchain technology may both alter certain design choices and possibilities, as well as open new methods and opportunities to implement such a system. [54] provides design considerations for such a decentralized system by combining the research results of [63] work about standard reputation systems with findings of [69] and [68] works on reputation in peer-to-peer networks [54]. In a similar manner as above, they also separate into design choices that are either setting expectations on either the user or the system. Main additions contain concepts such as user consent (e.g., whether a reputation or claim is revealed), participation rules (e.g., which users can vote), confidentiality of data (e.g., keeping a voter private), and management of performance of the system and behaviour of the users.

Further, [49] discusses the possibilities of a blockchain-based reputation system in detail. Same as the works above, they provide an extensive taxonomy of reputation system characteristics. They specifically mention the importance of trust between users/peers and the prerequisites of "risk and interdependence" in such a system. Furthermore, they also include a blockchain taxonomy, as well as suggestions on which practices (in both reputation system and blockchain technologies) are the most probable to be the better choice from a design perspective. This was done after surveying both scientific paper and business initiative assessments regarding reputation system and blockchain features. According to them, several trends have been exposed, such as information on the dimension (how many aspects the reputation system targets; findings point to a single target), the aggregation (deterministic, i.e., no probabilistic calculation), value control (the community should control the reputation), data aging (reviews shall not deteriorate with age), and the selection (a user wants to primarily transact with partners that have a high reputations core). Moreover, they state important blockchain characteristics such as predominantly being publicly available, having no fees, and using the Proof-of-Work consensus. It is to note however, that the factors regarding blockchain are mostly incompatible with existing blockchain technologies that do not support customization like this are used (e.g., even if it were the preferred method, Ethereum would not support the Proof-of-Stake consensus method as of the writing of this thesis [84]). In the end, even if these suggestions may provide a extensive point of reference for future works, Bellini et al. have concluded that there is no single best practice regarding neither the reputation system nor the blockchain technology choice.

3.3 Examples of Existing Decentralized Solutions for Reputation Systems

While there are no decentralized reputation systems specifically for infrastructure providers at the time of writing this thesis, there is a progressively rising number of decentralized solutions for reputation in different areas such as:

- [46] applies several concepts such as different weighting for reputation/trust depending on time differences, a higher trust in people with a higher reputation, and loss of trust in their model for trust in blockchain-based e-payment systems.
- [85] and [65] propose different solutions and evaluation methods for blockchain-based trust between peers in vehicular networks.
- [83] investigates possibilities of using a blockchain as a "Proof of Intellectual Work" or as "Intellectual Currency".
- [74] present a blockchain-based decentralized framework for crowdsourcing.
- [50] introduces a work about building trust on crowd-sourcing platform and discusses different important concepts, such as calculation and updating of reputation, rewards for positive usage, and robustness against attacks.

Note that this is not an exhaustive list, rather a short list of works about reputation and different implementations thereof. Numerous similarities can be found across multiple fields such a system might be used in, reinforcing the last sections notion of having no best practice regarding the definition of a (blockchain-based) reputation system.

3.4 Security in Decentralized Reputation Systems

As mentioned in [49], decentralization alone does not prevent manipulation and certain attacks on the system. It emphasizes the need for additional security measures to prevent or mitigate attacks. The main attacks in current distributed reputation systems mentioned in [59] are the **Bad-Mouthing attack** (intentionally lying about the review, where the review is usually negative), the **Bad-Collusion attack** (multiple nodes colluding to lower the reputation of a specific node, or to improve their own), the **Sybil attack** (the attacker uses multiple identities to amass a large influence), the **Re-entry/Whitewashing attack** (leaving an account that has a bad reputation and using a new one), and the **Ballot stung attack** (trying to increase their own reputation by spending tokens while neglecting legitimate transactions). [49] discusses how most of those attacks could be prevented with a correct use of a blockchain technology and Smart Contracts, while [50] shows a developed solution which has measures for multiple of these points. Furthermore, [72] introduces methods for rater credibility requirements and evaluations thereof, which can potentially be integrated for additional protection against attacks such as Bad-Mouthing and Bad-Collusion.

While the concept of decentralization speaks against having a central authority with additional capabilities, they may prove to be essential for stopping well-coordinated attacks such as mentioned in [4] or [8]. However, such a central authority may not have enough impact depending on the type of attack. For attacks that are directly targeted at the blockchain or at Smart Contract functionalities as mentioned in [48], [11], and [17], there may not be an easily accessible method for developers to protect against. As discussed in [66], types of consensus are a non-negligible factor and, depending on the chosen blockchain technology, do not have many options (e.g., Ethereum only having the Proof-of-Work consensus at the time of writing this thesis [84]). To conclude, while some attacks can be protected against in an active way, others (such as choice of the blockchain technology) do not provide an opportunity to implement additional security measures.

Chapter 4

Solution

The main goal of this work is to propose the Reputation Manager (ReM) which will serve as a reputation management system for infrastructure providers (including a punishment and reward system depending on behaviour, mainly perceived by the reputation), but also consider the authenticity and eligibility to vote of both the users. Additionally, the ReM will fully integrate the functionalities of BRAIN with adjustments to the handling of both the major participants (customers and infrastructure providers) and the auctions. While the reputation management concept of the ReM can be applied for differing reputation systems for infrastructure providers, this solution will be tightly coupled to the BRAIN when considering interfaces and data accumulation.

This chapter will focus on providing information about the concepts, structures, and design choices necessary to create the ReM and its connection with BRAIN. First, they will discuss the different participants (*User*, *Provider*, and *Manager*) including their interactions and attributes. The relationship and interactions between both distinct and identical types of participants will be considered. Next, the factors and weights for the reputation system and the individual *Rating*s will be defined. After establishing an understanding about the involved parties and ratings, measures to enable a secure and trusted communication (*Credibility* and *Eligibility* of a *User* or *Provider*) in the system will be explored, which opens more aspects to consider when discussing the participants. In the final two sections, a better overview on the interactions in the system will be explained and a solution for the integration into BRAIN (and the refactoring thereof) will be investigated. The sections try to follow a bottom-up representation where possible. Concepts that are briefly mentioned in the beginning may not be explained until a later section. An overview of the relations and interactions is provided in Figure 4.1. Green lines and entities signify freely available roles and functions, whereas red symbolizes restricted available functions, moderated and permitted by the manager, which is shown as yellow. BRAIN's refactoring is marked with a blue sign. Thus, familiarity with the concepts of auctions, auctioneers, bids, and bidders from BRAIN (Section 3.1) is necessary.

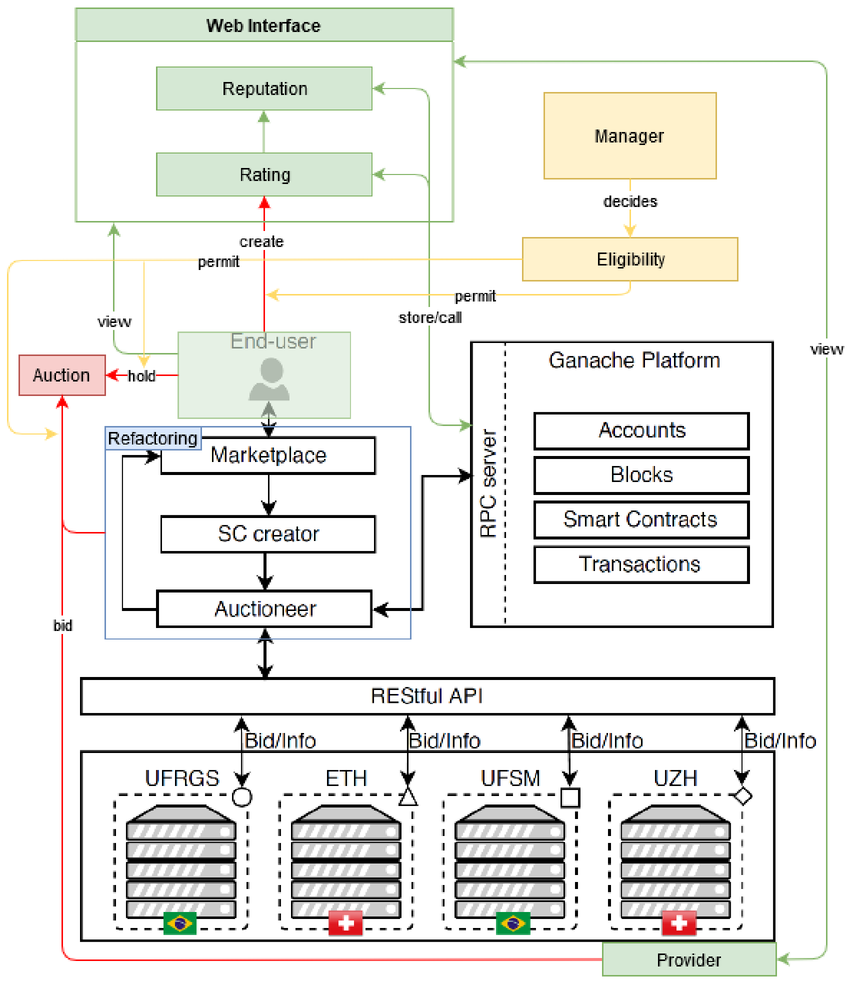


Figure 4.1: Overview of the ReM (extension of Figure 3.1).

4.1 Users

By considering Sections 2.2 and 3.2 it can be deduced that there is a need for voters of some form for a reputation system to get input data. Since one of the main goals of the ReM is to have it be integrated into BRAIN, it is assumed that the voters in this case will be people intending to buy a service from an infrastructure provider. However, this does not necessarily put a restriction on a provider to not be able to take advantage of the platform as a buyer, i.e., providers could be customers as well. Taking this into consideration, a *User* can be any entity that is actively utilizing the auctions of BRAIN as an individual looking to buy a service or to rate a provider. To keep the "confidentiality" aspect of reputation systems, Users should be identified by their unique address in the Blockchain.

A user should generally be able to view a list of all providers and their reputations unrelated to whether they have used or provided a service or not. Analogous to BRAIN, they should also have the possibility to start a new auction with different information, resources, and priorities, where the providers can place their bids, but unlike in BRAIN, the user (auctioneer in BRAIN) should not be forced to automatically accept the lowest bidder as the winner, but rather have a choice in which provider they want to select. While the auction is running, a list of the bidding providers, including their reputations shall be viewable. To end an auction, the auctioneering user has the opportunity to either choose a winner and thus end the auction, or in the case that they decide against picking any of the providers for any possible reason (e.g., bad reputation scores or the request having become redundant) they should be able to cancel the auction without having selected a winner.

Finally, a user should be able to leave a rating for a provider. Taking the concepts behind reputation systems into consideration once again, it is necessary to keep integrity and control over the individual ratings, which is where the concept of credibility and eligibility becomes necessary. More information on those will follow in the Sections 4.5 and 4.6.

4.2 Providers

As mentioned in the previous chapter, the *Provider*s are the equivalent of bidders in BRAIN. They have the option to bid in an auction if their service options correspond to the information, resources, and priorities given by the auctioneering user. The bid will be calculated in a localized manner for each provider, keeping this as another feature that is identical to BRAIN. For clarity and ease of use, apart from identifying the providers by their address in the blockchain, it is beneficial to have a service or provider name. The provider will thus supply the auctioneering user with a set of information containing their chosen service name in addition to the bid and address. This should allow for a more user-friendly identification when displaying the reputations of the bidding providers. All ratings and the average reputation for a specific provider should be viewable by any user.

4.3 Manager

The *Manager* is a user with special privileges. In everyday networks or systems, it might be referred to as the administrator of a system. While it is a user, it should not interact with the system in the same way a user might. The manager's function is to administer every transaction and to exact punishment for any fraudulent behaviour of either user or provider. This concept may speak against the idea of decentralization, but as discussed in Section 3.4, having a fully automated security or behaviour-check system is a colossal undertaking prone to errors. While automated punishment is certainly necessary, the manager should have the possibility to hand out punishments (or revoke them) manually. The manual handling and analysis of credibility and eligibility paired with metadata of transactions allows for a more directed way of dealing with perpetrators or redeeming falsely accused ones.

4.4 Ratings and Reputation

In Sections 2.2 and 3.2, multiple different ways to rate or review a product or service have been considered and showcased. Since the use case of the ReM (and BRAIN) is primarily targeted for human interaction, a fully automated solution depending on sent and received data such as in vehicular networks [85, 67] is not a suitable choice. Furthermore, since the provided services may reveal to have ambiguous qualities, e.g., a very low cost but overall bad user experience because of technical problems such as disconnecting for longer periods of time during the use of service, more than one scoring factor is needed. While there are no reputable theses or surveys to be found about the most important selling factors for infrastructure providers at the time of writing this paper, it can be estimated which factors could be used for ratings while investigating the main advertising points of bigger infrastructure providers. The following non-exclusive list shows multiple rather prevalent factors that are mentioned in both advertisements (e.g., Procloud [31], Nexellent [30], Gryps [22], Swisscom [38], Microsoft Azure [29], Amazon EC2 [3]) as well as third party reviews or recommendations (e.g., PCMag [77], Cloud Carib [9], IBM [24], edgeuno [14]) of those services:

- **Security**: How susceptible is the system to attacks?
- **Stability**: Does the service provide a stable user experience or does it, e.g., disconnect frequently?
- **Bandwidth**: How much bandwidth can the provider offer?
- **Support**: Does the provider offer assistance in the case of technical problems?
- **Uptime**: Does the service often have downtimes?
- **Price**: Does the price match the relative service offers?
- **Location**: Where are the servers of the provider situated?
- **Latency**: What is the delay when using the service?
- **Interoperability**: How well does the service function with other services or applications?
- **Usability**: What restrictions regarding interfaces does the service have?
- **Integration**: How simple is it to integrate the results of the service into personal systems?
- **Functionality**: Does the service offer every functionality they advertise?
- **Capacity**: Does the service contain enough storage possibility (e.g., maximum data size in a cloud service)?
- **Technology**: How recent is the technology used by the provider and has the correct choice in technology been made?

While extremely specialized use cases for utilizing an IaaS may prove to be depending on a perfect execution of one criteria, it can be assumed that the average use case of the ReM (previously being an even more non-specialized bidding place in the form of BRAIN because of only considering the lowest priced bidder that can fulfil the service needs with minimal requirements) should appeal to a broader set of customers that mostly look for an adequate and not necessarily perfect IaaS solution. While considering this case, the above list can be summarized into a smaller and uncluttered subset of five factors:

- **Security:** While a security factor is depending on the service, it could contain protection against breaches of data centres, tampering or other cyber-attacks. A high score is given if the data used and the connection to the service is secure.
- **Performance:** Contains factors such as bandwidth, latency, etc. A high score means the service kept the contracted values in the specified areas.
- **Uptime:** Low score when a frequent loss of service happens.
- **Price:** The score depends on the ratio of features offered relative to the price.
- **Support:** In the case of a negative experience in any area, the infrastructure provider may offer assistance or additional services. A high score means the provider accommodated to the needs of the customer.

It is immediately apparent that some of the general problems of reputation systems are inherent in this scoring system. First, what each customer perceives as a "good" value can not be defined perfectly. Second, depending on the needs and expectations of a user, they may either rate a service more leniently or more harshly. Lastly, there is the case especially but not exclusively for the "Support" factor that a customer may not experience any features of a specific service at all. Solving these arguments does again not have a clear solution, they will largely be accounted for with the right choice of scoring and calculation of a Reputation out of the individual ratings. As seen in Sections 2.2, 3.2, and 3.3, there are multiple ways for reviews and ratings to be implemented. While there is no clearly winning solution, the way to handle the scores (e.g., when does a provider get a certain score for the factor "Price") can be considered the critical point in creating a rating. One *Rating* in the ReM can thus consist of a set of standard 1 to 5 scores where each score's definition is as shown in Table 4.1.

Table 4.1: Available scores and their definitions.

Score	**Definition**
1	The provider does not uphold the contract agreement in an extreme way. Since this score can have a critical influence on both trust for future users, as well as on the credibility of the provider (and potentially for the scoring user, in the case of deceit), this score should only be given in extreme cases of breaching agreements.
2	The service has a major problem in the scored aspect. While the service is still usable, it is certainly not recommended.
3	The service is provided in the expected way but with small issues or complications.
4	The service is provided in the expected way. The agreed resources and priorities are met.
5	The service is provided in a flawless manner.

While small deviations between the five factors may exist, this scoring system should prove to be sufficient and inclusive enough to provide an unambiguous score for each factor [43]. Considering the aforementioned problem of not being able to rate, e.g., "Support" if this feature was not used, some scores may inflate or deflate depending on the rating users and their sample size. In some instances, a user might think it is unnecessary to give a score to a factor they did not actively use, which can lead to unexpected numerical values in the rating. In this case, a standardized answer must be chosen. Since the scoring system accepts the score 4 as the "normal" or "expected" outcome, it could be seen as the preferred input in such a case. In a very high sample size this could distort the reputation of a provider into the positive since a score of 4 is numerically clearly in the upper half of the scoring system. For both this case and the extreme cases (score average below 2) it is beneficial to have further information provided in a rating to eliminate remaining uncertainties and ambiguity. The widely adopted way for reputation systems in general is the possibility to include a written review or proof of misconduct (or consistency, i.e., a positive behaviour of the provider). One way to include this without cluttering the blockchain too much is to include a hash code, which leads to a certain entry in an external database dedicated to storing this type of information. Having information of that sort could greatly help users to choose a provider [8], give necessary feedback to providers on their services, and (when used for further analysis) the reputation, credibility, and eligibility may be either automatically or manually adapted by the manager. To summarize, a user can create a rating consisting of five factors (Security, Performance, Uptime, Price, Support) with values between 1 and 5, and a potential hash code for additional information. The average of each rating score of a single provider defines their reputation scores. Averaging those reputation scores would equal the providers' reputation. Every user should be able to view both the reputations (and scores) and the individual ratings of a provider.

4.5 Credibility and Eligibility

Decentralizing the reputation system makes it by design more robust against certain cyber-attacks such as external manipulation of a data centre, as discussed in Section 3.4. Apart from cyber-attacks from a technical point of view, there are many miscellaneous misbehaviours from actual users which a reputation system should have some protection against, which gives the system requirements for additional security measures. With the use of Smart Contracts there are multiple cases where ensuring that only the users that are supposed to interact with the system in a specific way is implementable in a rather simple way. By posing some restrictions to users depending on what they have done so far regarding ratings, auctions, used services, or with analyses of metadata (e.g., date of first interaction with the blockchain vs. date of first rating) in the system, certain cases can be eliminated almost certainly if it is assumed that only one individual has access as a certain user (i.e., no account sharing or hijacking). To decide whether a participant (either user or provider) is eligible to interact with a part of the system, an identification through consideration of *Credibility* and *Eligibility* shall occur in virtually any step or involvement of the participant using the system.

Credibility: The credibility of a participant is a numerical value which can be used to decide if a participant is seen as trustworthy. It is an attribute that can change after a participant actively induces a change in the blockchain or in the perception of information on the blockchain. For users this would contain creating, ending, and cancelling an auction and especially rating a provider, whereas for providers it contains a correct fulfilment of the contract by providing the service the user subscribes for. To lower the chances of falsely and immediately restricting a participant from making transactions in the system, the credibility should use the concept of threshold tiers. These tiers would facilitate differentiated weighting of ratings made by the user.

If a users' credibility base value equals 1.0, the thresholds could be divided into different tiers with changes to the ratings' weight as shown in Table 4.2.

Table 4.2: Threshold values with a potential reduction/increase in weight of the rating.

Value	Weight change
0.00 - 0.19	Ratings have no weight and are not considered to be part of the reputation. Additionally, the participant is excluded from further use of the system except for viewing purposes.
0.20 - 0.49	Great reduction in weight of the rating (factor x0.5).
0.50 - 0.79	Small reduction in weight of the rating (factor x0.75).
0.80 - 1.20	Ratings are included as they are (factor x1.0).
1.21 - 1.50	Small increase in weight of the rating (factor x1.25).
1.51+	Great increase in weight of the rating (factor x1.50).

The weight reductions of a rating are provided to increase the chance of stalling malicious behaviour, while the weight increases may be used to encourage truthful interactions when creating a rating for a provider. Either way there is a necessity of an algorithm to increase/decrease credibility in an as-correct-as-possible way. The manager is tasked with algorithmic decision-making when to adapt the credibility of a user and to hand out punishment if necessary, for the special cases of false accusations this should be done manually. For special perpetrators, a blacklist only accessible and manageable by the manager should exist in the system. While the manual approach is not a feasible solution in a reputation system with a large sample size, it is in today's technologies still a necessity to have, as seen with mass review deletions in, e.g., Google Play Store [4].

Eligibility: The overall eligibility of a user or provider is comprised of multiple conditions in addition to the credibility of the participant. Both the credibility as well as the other part of the eligibility by itself can under the right circumstances prevent the participant from either interacting with the system, or only with certain of its functions. Either way, eligibility and credibility could be observed individually, but are tightly coupled together because of the credibility's power to override any eligibility of a participant. If the credibility falls below the critical threshold (<0.19), it automatically means that a User may not rate anymore or in a case of a provider that they may not bid anymore. What follows are requirements and restrictions necessary to cover some of the explicit reputation system goals as stated in Section 3.2:

- **The user has not used a service from a certain provider:** The user cannot rate said provider.
- **The user has already rated a Provider:** The user cannot rate that provider again.
- **The participant is on the blacklist:** The participant cannot interact with the system apart from viewing actions (e.g., viewing the list of providers and their reputation/ratings).
- **The numerical value of credibility of the participant is below the critical threshold (<0.19):** The participant cannot interact with the system apart from viewing actions (e.g., viewing the list of providers and their reputation/ratings).

Smart Contracts make it possible to enforce this set of rules, as well as add more restrictions to the system as it grows. Both the credibility and eligibility verifications should thus be callable for a multitude of uses as well as they should be extensible for additional cases and restrictions. Since the credibility is a functional subset of the eligibility, the eligibility gives rights to perform transactions in the blockchain as a whole, thus it should be the functionality that expends as the system matures by receiving, e.g., more complex automated security features. An overview over the rating process is shown in Figure 4.2.

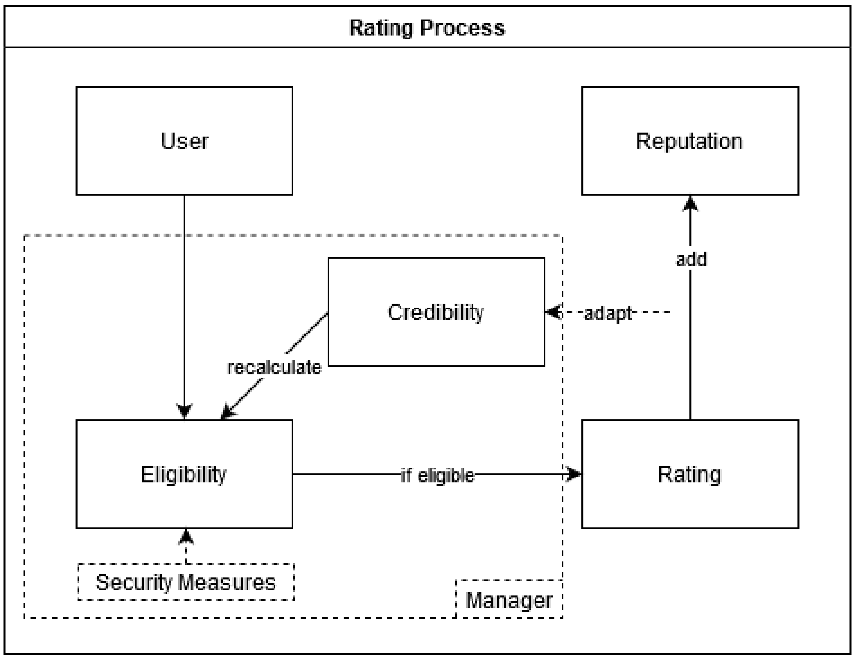


Figure 4.2: Overview of the rating process in the ReM.

4.6 Interactions and Integration in BRAIN

After deployment, the ReM is available to both users and providers to auction and to bid, respectively. In contrast to BRAIN, where an auction is isolated to itself and its bidders, the ReM integrated onto BRAIN will grow in usefulness the more auctions and ratings happen. At early stages of the blockchain, there will be little information available about both participants, which might be a discouraging disadvantage for early adopters of the system. However, with the system maturing by storing more transactional data onto the blocks and growing its userbase, the benefit should increase for trustworthy participants by giving more precise information about the integrity of both users and providers. Figure 4.3 depicts an overview of the relationships in the ReM.

While auctions are held and ratings are being input, the manager acts as a passive moderator by analyzing credibility and eligibility values and excluding participants showing malicious behaviour or a record thereof. While the eligibility is in one part dictated by the thresholds of the credibility value, it is a prime opportunity to offer additional security options to keep transactions confidential, resilient, sustainable, valuable, and in a user consented context. Since functions such as choosing the winner manually are not supported by the implementation of BRAIN, refactoring and extension of BRAIN's code and structure are needed. To create ReM as a reputation system integrated in BRAIN, following adapted or additional functionalities are needed:

- Move from an isolated auction to an auction with many spectators by storing transactional data as a proof of conduct.
- Change the automated winner selection by replacing it by a handpicked choice option, where the User may choose from a list of bidders. Furthermore, the bids and reputations of said bidders should be available for the auctioneer.
- Both users and providers shall be excluded from holding auctions or bidding, respectively, if the manager decides that they are not eligible to do so.
- For ease of use, a user should have access to a graphical user interface (GUI) in form of a web interface, where the providers are listed in a tabular form. BRAIN's methods of getting information about the auction and having the bids calculated in a local manner (the "bid algorithm") shall be preserved.

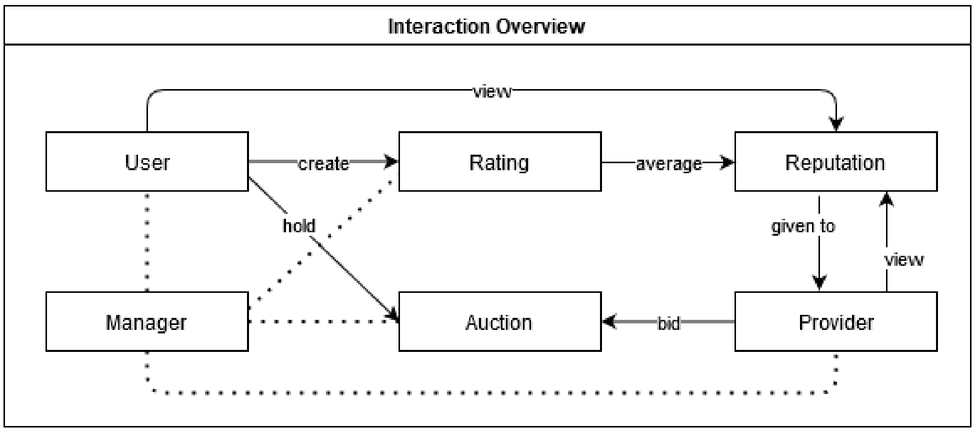


Figure 4.3: Interaction overview of the ReM.

Chapter 5

Prototypical Implementation

The prototype for the ReM has been developed with the same technologies as the BRAIN. It runs on an Ethereum blockchain using Solidity 0.6.3 Smart Contracts and is deployed and tested using a local Ganache [40] workspace in combination with a Python 3.8 script. Additionally, Remix-IDE [35] has been used to write and test the Solidity code before including it in the Python script. For the web-based user interface, Python Flask 1.1.1 [18] has been used to provide necessary RESTful APIs to support the inter-component communication.

Following the concepts illustrated in Chapter 4, several definitions about the relationships between standard participants (user, provider) and concepts (auctions, bids, ratings, reputation, credibility, eligibility) may be extracted. Since the manager is a special entity that will possess automated functionalities as well as multiple restrictedly available functions, the manager position will be assigned by using the Solidity *constructor()* function. With this, either the deploying account address in Remix or the first Ganache account address, respectively, will become the manager. To ensure that certain functions can only be accessed by the manager, a Solidity *modifier onlyManager()*, which requires that the sender's address equals the manager's, is implemented. Apart from being the main actors, both the users and the providers contain enough information to warrant being a *struct* to help storing data such as their *address*es. Furthermore, since the reputation of a given provider can be calculated out of their ratings and ratings are heavily associated with providers, another *struct* with the name *Rating* is implemented and included in the *Provider* in the form of a *mapping* from *uint* to *Rating*. *Rating* includes the five rating factors discussed in Chapter 4 as well as an additional *string* for a database access hash address. Thus, the requirements for the "reputation system" part of the ReM are sufficiently covered when considering the main actors and their correlations.

Next, the credibility and eligibility shall be implemented. One of them is included as an *int* in the *User*, while the other is calculated on calling a function, where the requirements will depend on the function itself. Since the base value of credibility has been defined as 1.0 in Section 4.5, it is implemented to automatically be set to the value 1000 upon creation of a new user because Solidity does not support floating point numbers at the time of writing this thesis [37]. The same concept of up- and downscaling a numerical value for calculation and viewing will be applied for the rating - reputation (e.g., 5.0 equals 5000) relation as well. To apply changes to the credibility of a user, the function *setCredibility* is implemented privately to be called after performing any interactions with the system that might warrant a change in the value. The algorithm for detecting and applying such changes is an opportunity for future work on this topic.

Similar to the *setCredibility* function, a get-function for the eligibility named *getEligibility* is created. Where the first defines the credibility value in a user and is viewable at any time, the second has additional checks and potential calculations inside and returns a *bool* value depending on whether the user is eligible to perform an action or not. Various requirements can be set for the eligibility, and while multiple of them can be implemented in the same function, some of them are more valuable outside of it. If every possibility was covered with *getEligibility*, the probability for more errors and a rather bloated function arises, which would add unnecessary complexity for future extensions of the ReM. Since most of the requirements can be checked with a single *require()* call in Solidity, the *getEligibility* function will only be used for deciding if a user is able to rate a provider. For this, the function shall check the credibility of the user, if the confirmation passes, it is required that the user has already used the service (to stop new Users from signing up and assumedly directly manipulating the reputation of a provider) and that they have not rated it yet. To perform these checks the *struct User* shall have two *mappings* where the used and rated providers, if available, are mapped into.

For the majority of other use cases, the eligibility-checks can be reduced to confirming that the sender's address is the same as the target's they are trying to call a function upon with a *require()* function. This can be applied in a similar manner to the providers, where a separate function for the eligibility is not necessary and replaceable by a *require()* call in the affected functions. For this proof of concept, the credibility of a provider will not be considered because of the aforementioned lack of an implemented algorithm. Instead, an additional *blacklist()* will be used to exclude them, which can also be used to exclude special perpetrators (marked by additional tracing or checks) in the user area. With this set of information, the *User*, *Provider*, and *Rating* can be implemented as shown in Listing 5.1. Not mentioned variables such as *providersCount* and similar are included to serve as anchors for iterating through mappings.

Listing 5.1: An overview of the *User*, *Provider*, *Rating*, and *Auction struct*s.

```
    struct Provider{
        string name;
        address providerAddress;
        mapping(uint=>Rating) ratings;
        mapping(uint=>address) ratingFrom;
        uint ratingsCount;
        bool exists;
        bool blacklisted;
    }

    struct User{
        address user;
        mapping(uint=>Provider) ratedProviders;
        mapping(uint=>Provider) usedProviders;
        uint ratedProvidersCount;
        uint usedProvidersCount;
        int credibility;
        bool exists;
        bool blacklisted;
    }

    struct Rating{
        uint security;
        uint performance;
        uint uptime;
        uint price;
        uint support;
        string hash;
    }

    struct Auction{
        mapping(uint=>Provider) bidders;
        uint biddersCount;
        mapping(uint=>uint) bids;
        bool exists;
    }
```

Apart from the already mentioned *usedProviders* and *ratedProviders*, one of the requirements for a user to rate a provider is the existence of both. To provide an overview of the registration of a new provider, the integration into BRAIN shall first be investigated. The ReM will accommodate to the existing implementation of BRAIN, where an auction may be started over a local python script and a provider may read the auction information with the same method as well. To accomplish this, all of BRAIN's functions require to be refactored for the possibilities to support dynamic auction creation and completion (i.e., address-based information, resource, and priorities calls and assignments). Furthermore, it needs the additional feature for the auctioneer to be able to choose a winner instead of automatically being allocated the bidder with the lowest price. Finally, all the functions need to be included in the ReM. To account for the possibility to have multiple auctions running at the same time (each restricted to one address, one user may not hold two auctions simultaneously), the *struct Auction* and the main information holders *Info*, *Resources*, and *Priorities* are implemented. While it may be the more suitable solution to have the latter three associated to the *Auction* by directly including them in it, the restriction of having a set amount of maximum attributes in a Solidity *struct* (16 in the used version of Solidity), it is not an advisable choice since the three information *struct*s together contain a combined amount of 13 attributes. With the *Auction* itself having an absolute minimum of three (auction address, bidders, bids), this option becomes unavailable, especially if there are to be future extensions of this system. The prototype will thus associate 4 *struct*s to create one auction, interconnected by an address.

Table 5.1: Functions used in the prototypical implementation of ReM.

Name	**Attributes**	**Definition**
getInfo()	auction_address	Parallel to *getInfo()* in BRAIN, additional target auction.
getResources()	auction_address	Parallel to *getInfo()*.
getPriorities()	auction_address	Parallel to *getInfo()*.
setInfo()	vnf_name, vnf_type, vnf_developer, contract_period	For dynamic creation of an auction. Technically BRAIN used this with the*sc_creator.py*.
setResources()	mem, disk, cpu	Parallel to *setInfo()*.

setPriorities()	country, additional_memory, additional_disk, additional_cpu, latency_max, packets_per_second	Parallel to *setInfo()*.
submitBid()	auction_address, bid_value, provider_name	Identical as in BRAIN, but with a target address.
cancelAuction()	- - -	Cancel the held auction without a winner.
endAuction()	winner_address	Selecting a winner and ending the auction. This function includes a ReM equivalent of *getWinner()* from BRAIN.
startAuction()	- - -	Opening a new auction.
getBiddersCount()	auction_address	Returns *biddersCount*.
getBids()	auction_address, bid_position	Returns bids of the target auction.
addBlacklist()	target_address	Adds the address to the blacklist.
isBlacklisted()	target_address	Returns whether the address is blacklisted.
getProvidersCount()	- - -	Returns *providersCount*
getRatingsCount()	provider_address	Returns *ratingsCount*
getRating()	provider_address, position	Returns the rating mapped to the given position.
addUser()	user_address	Registers a new user in the ReM.
addUsedProvider()	user_address, provider_address	Adds the provider to the *usedProviders* of the user.
addRatedProvider()	user_address, provider_address	Adds the provider to the *ratedProviders* of the user.
addProvider()	provider_name, provider_address	Registers a new provider in the ReM.
getProviderName()	provider_address	Returns the name of the provider.
getProviderNames()	position	Returns the name of the provider mapped to the given position.
getProviderAddresses()	position	Returns the address of the provider mapped to the given position.
setRating()	provider_address, security_value, performance_value, uptime_value, price_value, support_value, hash	Adds a new rating with the given parameters to the provider.

getEligibility()	user_address, provider_address	Returns whether a user is eligible to rate a the provider.
getReputation()	provider_address	Returns the individual reputation score averages, as well as the average reputation.

An auction may be opened after the user has successfully called the *setInfo()*, *setResources()*, and *setPriorities()* functions, of which the attributes are identical to the ones in BRAIN, except for the aforementioned general structure changes in the way an auction and the information sets are handled (mapping every held auction at a point in time). The *exists* attributes are generally used for convenient checking whether an entity exists. Upon starting a new auction, if the user's address has not been used to create one before, a new user with said address is generated by calling *addUser()* inside of *startAuction()*. After the creation, the bidding phase for the providers is open. An example of a bid is implemented in the same manner as it is in BRAIN by using *submitBid()*, albeit minimally refactored to not only send the bid on a specific auction, but to also send their chosen displaying name for later authentication and illustration. In the same principle of adding a new User on first opening of an auction, a provider is added per *addProvider()* upon placing the first bid on an auction. Parallel to BRAIN, the auctioneer may choose the point in time when to conclude the auction by either choosing a winner with *endAuction* or cancelling it if no satisfactory bid has been placed. Both actions remove the information for the auction and the auction itself as well so that a new auction with the user's address may be held. If a winner is chosen, the winner is added as a provider to the *usedProviders* list of the auctioneer, which makes the user eligible to rate a provider.

The last two remaining main Solidity functions apart from aiding functions such as *setters* and *getters* are *setRating()* and *getReputation()*. When a user wants to rate a provider with *setRating()*, the system checks their whole eligibility first by calling *getEligibility()*, and if deemed eligible, adds a new rating to the targeted provider. As mentioned before, the reputation of a provider is not directly stored, but calculated each time it is viewed. Currently, the weightings of ratings as seen in Table 4.2 are not applied in the prototype because of the lacking credibility algorithm (i.e., a way to hand out punishments or rewards). One future refactoring method would be to include an additional attribute to *Rating*, which will associate the threshold multiplier of a user to their rating and simplify the calculation in *getReputation()*.

While testing for various functionalities and errors has been done in the Remix-IDE, an additional script is needed for a quicker case study evaluation, where a user starts an auction and multiple providers are bidding on it. For this Python script, a combination of libraries able to deal with blockchains and Smart Contracts in a reliable way are imported. Flask is used to create an application that offers a simpler and quicker interaction in the form of a web-based user interface to a standard participant (user, provider). Codecs [10], solcx [32], and web3 [44] are used for additional functionalities regarding Smart Contracts and interactions with URL's. The Ganache Framework is used to simulate a blockchain that the ReM will be deployed on. The functions shown in Table 5.1 are the functions used in the ReM. Cells marked as blue contain modified functions of BRAIN, while red cells contain get-functions that are not essential to the ReM but are beneficial for depiction and back-end calculation purposes. The red cells are thus necessary for operating the Python script in a simple manner. The most prevalent ones (disregarding the necessary get-functions for both the ReM and the Python script) are shown in Table 5.2. These are the Python Flask functions used for easier interaction with the ReM over a web-interface. It does not include the functions in white and red cells shown in Table 5.1 since all of them are implemented in a rather similar way. When *example_auction.py* and *example_bidder.py* are executed while the ReM is running, a new auction is opened and multiple bids by different providers are placed on said auction. By calling *getAuction* in the browser, a table made with *Google Visualization: Table* [21] is shown. This table was chosen for its simple implementation method and for the ability to easily sort by a value (e.g., from lowest to highest bids or highest to lowest reputation).

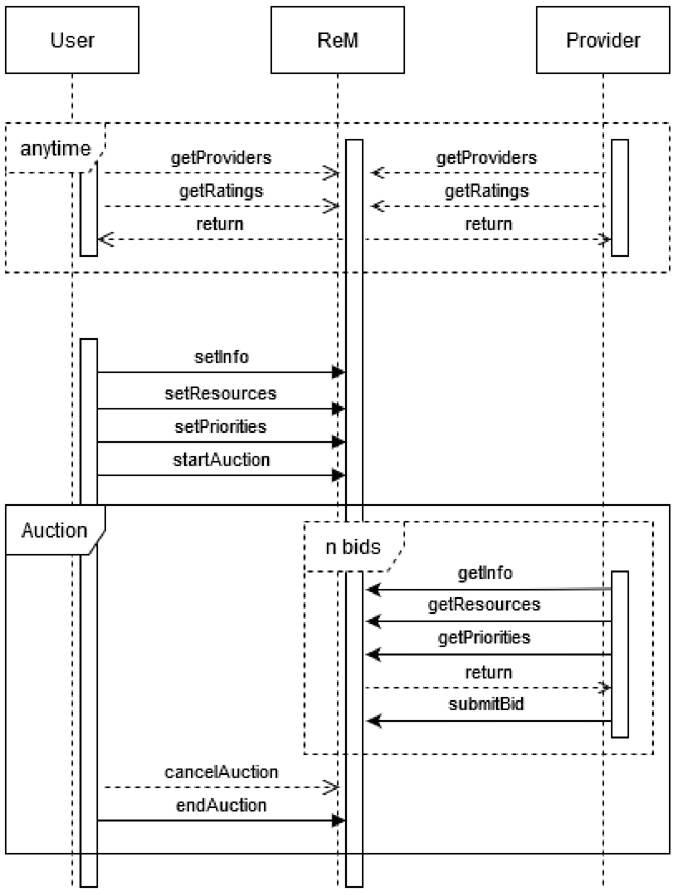


Figure 5.1: Simplified sequence diagram that showcases the procedure of an auction and the possibility of viewing operations at any time.

Additionally, there are hyperlinks on the addresses in the table which lead to the provider's "rating page", where all ratings including their potential hashes (for the database entry) are visible. To illustrate the rating page in a clearer manner, the *example_ratings.py* can be implemented to add ratings to a provider. Note that it is beneficial to achieve this in this simulated workspace by removing both some *private* modifiers, as well as the *getEligibility()* requirements in the *reputation.sol* file. Without these modifications, testing becomes a long and tedious process because of the multiple restrictions posed by the Smart Contracts. In order to demonstrate the functionalities and interface of the ReM, multiple functions in the *reputation.sol* file have to be modified to remove requirements and restrictions that are hindering certain actions (e.g., the most prevalent one is the eligibility check).

Table 5.2: Prevalent Python Flask functions.

Name	Attributes	Definition
getAuction()	auction_address	Returns the auction with the given address in the form of an interactable table. Shows the bidders, their bids, and their reputation (scores).
getProviders()	- - -	Returns a list of all providers registered in ReM in the form of an interactable table, including their average reputation (scores).
getRatings()	provider_address	Returns a list of all ratings for the given provider in the form of an interactable table. Shows the hash code included in a rating.

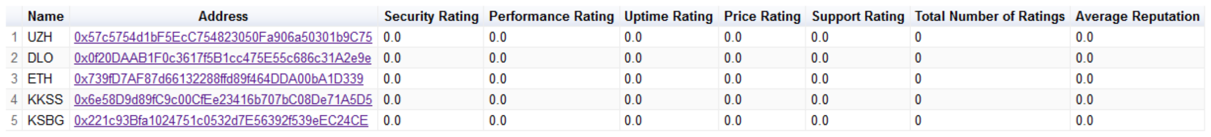
Chapter 6

Evaluation

This chapter will go over multiple use-case scenarios including viewing a list of the providers and showing their ratings, a user starting an auction and choosing a winner, and a user rating a provider. It will explain what steps each participant has to follow to achieve the wished results. Furthermore, it will discuss economic aspects (e.g., gas costs, which are the unit used to measure transaction costs in Ethereum [27]), the practical feasibility of such a solution, and limitations of such a decentralized reputation system.

6.1 Use Case #1 - Viewing the providers and their ratings

Whether a provider wants to view their own reputation and potentially get access to the optionally linked files in the ratings, or a user wants to browse or inspect the complete list of participating providers, the calls are identical and non-discriminating. While connected to the blockchain, the address of the blockchain including the attached function */getProviders* will show a table with an exhaustive list of the providers. For the prototypical implementation, this can be achieved in the following way: Open a Ganache workspace, open the ReM and run *reputation_manager.py*.

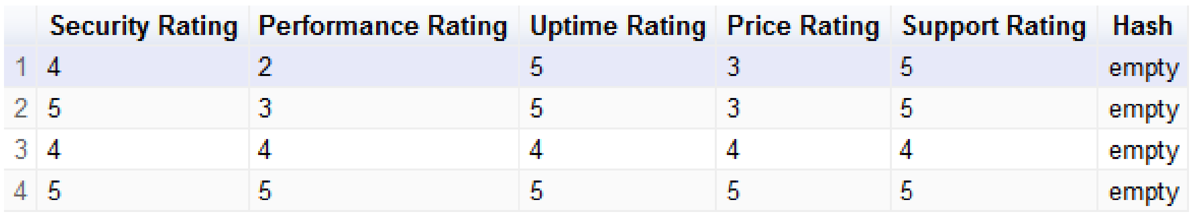


	Name	Address	Security Rating	Performance Rating	Uptime Rating	Price Rating	Support Rating	Total Number of Ratings	Average Reputation
1	UZH	0x57c5754d1bF5EcC754823050Fa906a50301b9C75	0.0	0.0	0.0	0.0	0.0	0	0.0
2	DLO	0x0f20DAAB1F0c3617f5B1cc475E55c686c31A2e9e	0.0	0.0	0.0	0.0	0.0	0	0.0
3	ETH	0x739fD7AF87d66132288ffd89f464DDA00bA1D339	0.0	0.0	0.0	0.0	0.0	0	0.0
4	KKSS	0x6e58D9d89fC9c00CfEe23416b707bC08De71A5D5	0.0	0.0	0.0	0.0	0.0	0	0.0
5	KSBG	0x221c93Bfa1024751c0532d7E56392f539eEC24CE	0.0	0.0	0.0	0.0	0.0	0	0.0

Figure 6.1: A call on */getProviders* while the system has no ratings for any providers.

Since this is a freshly created session, opening *http://localhost:5000/getProviders* in a web-browser will return a table with no entries. To add a number of providers an auction is needed to be held, since new participants get added to the ReM after their first interaction that induces a change in the state of the system. For this, first *example_auction.py* and, after completion, *example_bidder.py* is executed. After execution, five new providers are added to the system, which can be shown by calling http://localhost:5000/getProviders one more time as shown in

Figure 6.1. To show the individual ratings for a provider, a hyperlink on their address opens their ratings page. Since there are once again no ratings in the system, any rating page shall be empty. After running *example_ratings.py*, an example of a ratings page can appear in the form of Figure 6.2. Viewing providers and ratings may occur at any time.



	Security Rating	Performance Rating	Uptime Rating	Price Rating	Support Rating	Hash
1	4	2	5	3	5	empty
2	5	3	5	3	5	empty
3	4	4	4	4	4	empty
4	5	5	5	5	5	empty

Figure 6.2: A call on a provider's ratings with multiple example ratings.

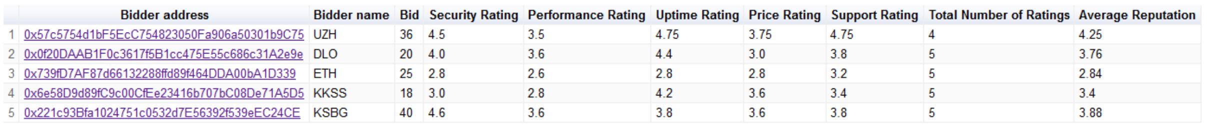
6.2 Use Case #2 - User starts a new auction and chooses a winner

Whether a user chooses to open an auction for the first time, or uses the ReM for the n-th time, starting an auction is always the same. Either the user stores their data (*Info*, *Resources*, *Priorities*) locally and uses a python script (similar as in the ReM's *example_auction.py*) to create an auction, or they call the individual functions manually by opening following calls (preceded by *http://localhost:5000*):

- /setInfo/<vnf_name>/<vnf_type>/<vnf_developer>/<contract_period>
- /setResources/<int:mem>/<int:disk>/<int:cpu>
- /setPriorities/<country>/<addMem>/<addDis>/<addCpu>/<latMax>/<pacPerSec>
- /startAuction

The variables depicted in "< >" can be replaced by their respective values as defined in Chapter 5. Note that while the first three calls can be interchanged, the */startAuction* call has to be done in the end, since it requires the existence of the first three. Upon opening their auction with */getAuction/<address>*, where the *address* depicts their own address in the blockchain, they may find an empty auction if no provider has yet placed their bid. In the case of placed bids, calling the auction may show a table of the bidding providers as shown in Figure 6.3.

The table includes the possibility to sort by highest/lowest score or bid, depending on what the viewer is interested in. For additional information about the provider, there is a hyperlink to the rating page for each respective provider (opening the page as shown in Figure 6.2). Since the ReM may simultaneously record different auctions and ratings, refreshing a */getAuction* page may show different reputation scores than before. If the auctioneer is not content with any of the placed bids, they may cancel the auction by calling */cancelAuction*, otherwise they may choose a winner by opening */endAuction/<chosen_bidder>*, where *chosen_bidder* depicts the address of the chosen provider. After completion, the auction and the information sets are deleted permanently from the ReM and have to be initialized again for further auctions. Note that a user may only hold one auction at a time.

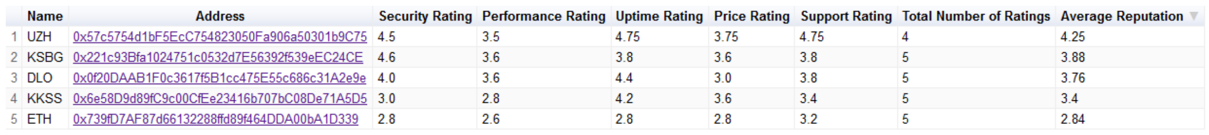


	Bidder address	Bidder name	Bid	Security Rating	Performance Rating	Uptime Rating	Price Rating	Support Rating	Total Number of Ratings	Average Reputation
1	0x57c5754d1bF5EcC754823050Fa906a50301b9C75	UZH	36	4.5	3.5	4.75	3.75	4.75	4	4.25
2	0x0f20DAAB1F0c3617f5B1cc475E55c686c31A2e9e	DLO	20	4.0	3.6	4.4	3.0	3.8	5	3.76
3	0x739fD7AF87d66132288ffd89f464DDA00bA1D339	ETH	25	2.8	2.6	2.8	2.8	3.2	5	2.84
4	0x6e58D9d89fC9c00CfEe23416b707bC08De71A5D5	KKSS	18	3.0	2.8	4.2	3.6	3.4	5	3.4
5	0x221c93Bfa1024751c0532d7E56392f539eEC24CE	KSBG	40	4.6	3.6	3.8	3.6	3.8	5	3.88

Figure 6.3: A call on a specific auction including five bids.

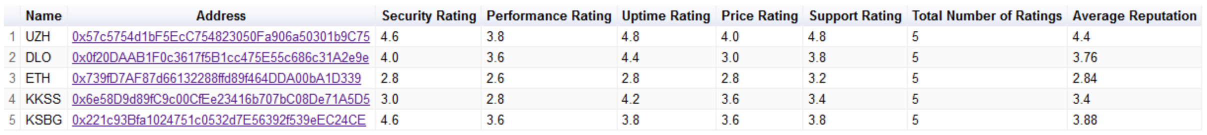
6.3 Use Case #3 - User rating a provider

After successfully holding and ending an auction (with a chosen winner), the user that held the auction is eligible to rate the chosen provider. To rate a provider, the user has to enter */setRating/<provider_address>/<security_rating>/<performance_rating>/<uptime_rating>/<price _rating>/<support_rating>/<hash>*, where the values in "< >" are once again placeholders for the values defined in Section 4.4. If no hash for eternal database access is available, a standard answer such as *null* or *empty* may be used. In this case, the user is extraordinarily satisfied with the service from the provider "UZH" and adds a rating with all values set to 5 and no additional hash. After the rating has been added, the table will change from the state shown in Figure 6.4 to the one depicted in Figure 6.5.



	Name	Address	Security Rating	Performance Rating	Uptime Rating	Price Rating	Support Rating	Total Number of Ratings	Average Reputation
1	UZH	0x57c5754d1bF5EcC754823050Fa906a50301b9C75	4.5	3.5	4.75	3.75	4.75	4	4.25
2	KSBG	0x221c93Bfa1024751c0532d7E56392f539eEC24CE	4.6	3.6	3.8	3.6	3.8	5	3.88
3	DLO	0x0f20DAAB1F0c3617f5B1cc475E55c686c31A2e9e	4.0	3.6	4.4	3.0	3.8	5	3.76
4	KKSS	0x6e58D9d89fC9c00CfEe23416b707bC08De71A5D5	3.0	2.8	4.2	3.6	3.4	5	3.4
5	ETH	0x739fD7AF87d66132288ffd89f464DDA00bA1D339	2.8	2.6	2.8	2.8	3.2	5	2.84

Figure 6.4: A call to show the list of providers.



	Name	Address	Security Rating	Performance Rating	Uptime Rating	Price Rating	Support Rating	Total Number of Ratings	Average Reputation
1	UZH	0x57c5754d1bF5EcC754823050Fa906a50301b9C75	4.6	3.8	4.8	4.0	4.8	5	4.4
2	DLO	0x0f20DAAB1F0c3617f5B1cc475E55c686c31A2e9e	4.0	3.6	4.4	3.0	3.8	5	3.76
3	ETH	0x739fD7AF87d66132288ffd89f464DDA00bA1D339	2.8	2.6	2.8	2.8	3.2	5	2.84
4	KKSS	0x6e58D9d89fC9c00CfEe23416b707bC08De71A5D5	3.0	2.8	4.2	3.6	3.4	5	3.4
5	KSBG	0x221c93Bfa1024751c0532d7E56392f539eEC24CE	4.6	3.6	3.8	3.6	3.8	5	3.88

Figure 6.5: Another call to show the list of providers after an additional rating has been made.

6.4 Economic Analysis

The gas cost estimates are taken from the ReM as they are shown in the Ganache workspace. Discussion about performance and practicality in real-world use will be held in Section 6.5. Table 6.1 shows a list of the callable functions of the ReM as well as their costs in gas and US-Dollars.

Table 6.1: Gas cost estimation for different actions of the ReM.

Action	**Gas Cost**	**Cost in USD (August 31, 2020)**[15, 12]
Deployment	6'541'901 gas	$783.13
getAuction	0 gas (get-function)	- - -
getProviders	0 gas (get-function)	- - -
getRatings	0 gas (get-function)	- - -
startAuction	649'161 gas	$77.71
submitBid	approx. 244'000 gas	approx. $29.21
endAuction	121'850 gas	$14.59
cancelAuction	65'927 gas	$7.89
setRating	approx. 275'000 gas	approx. $32.92

First, the deployment of the ReM has an extraordinarily high gas cost of 6'541'901 gas, which equals (assuming an average gwei conversion rate in [15]) approximately 1.8 ETH, converting to $783.13 USD [12]. Second, since the get-functions that only call information from the blockchain typically have no cost, their gas cost equals 0. Furthermore, the converted costs for starting an auction, submitting a bid, ending/cancelling an auction, and setting a rating are extremely high from a real-world point of view. From a practical and economic perspective, this solution would not be feasible if the cost was paid by the auctioneers and providers, which, with the current prototype, they do. One possibility to make the system more attractive in a public blockchain is to have the manager/deployer of the blockchain pay all of the costs. Multiple works in Section 3.2 mention that the users of a reputation system should not have to pay a price to contribute to the system to achieve a higher participation.

6.5 Discussion and Limitations

During the evaluation being conducted using the developed prototype, it was identified that the gas costs are too high to be a practical solution for the participants to make practical use of it. It is assumed that one of the main reasons for these costs arise due to all of the functionality being assembled in one single Solidity file, having a considerable quantity of complex entities being mapped, as well as having a high amount of embedded function calls (e.g., *getEligibility()* contains multiple functions call in it, while *getEligibility()* itself is called before setting a rating). However, it is possible to refine the code in the future to achieve a better performance in terms of gas usage and implement new features that allow users to interact with the contract (e.g., modify a rating while maintaining its integrity and historic). One possibility would be to split up the Solidity functions into multiple files and try to group them by functionality (e.g., BRAIN effectively split from the ReM, connected using an API) and investigate the fees in the same manner as in Section 6.4. A major problem encountered was that calling the mapped values (and saved values in general) with a function from a different Solidity file did not seem to be possible in a simulated/private environment using Ganache.

Furthermore, while much reviewing and studying have been conducted on reputation systems, decentralization, and security measures, many concepts could not be actively applied in the implementation of the ReM. While much thought has gone into what a rating should comprise, it remained one of the only applicable concepts since the discussions on blockchains, decentralization, and security were almost non-applicable in a pre-defined blockchain technology that includes its own concepts and measures (in this case Ethereum). Topics such as attacks and imbalanced populations (e.g., many providers on a small user base, or the opposite where worse-known providers may not succeed in any auctions because of their lack of ratings) have not been considered for the implementation.

Another issue was posed by the interactions between the Flask framework and Ganache. Multiple challenges have been encountered while testing the ReM deployed on a private blockchain in a Ganache workspace. The implementation of a fleshed-out user interface has been hindered. The most prevalent issue was the seemingly impossible task of using any different Ganache account than the very first one, making testing and simulating the ReM impossible without removing many restrictions from the Solidity file (e.g., multiple *require()* functions as well as the eligibility to rate providers). Note that while the non-modified version of the ReM deploys as intended in both Ganache and the Remix-IDE, it is much more beneficial to use the Remix-IDE for testing functionalities in a simplified way, thus avoiding the issues mentioned above.

In general, the ReM includes the most important functionalities for implementing a reputation system, but it is still missing some aspects that could be required for a real-world scenario. Apart from the high gas costs, the missing algorithm to punish/reward users and providers may currently be the most significant weak spot. Additional security measures against malicious users and providers and general attacks on the blockchain should be studied further. If these issues are eradicated and the system built upon in further works, the ReM and decentralized reputation systems, in general, could become a rather interesting concept in the future.

Chapter 7

Summary and Conclusions

This thesis has discussed multiple sizable concepts such as the blockchain technology, smart contracts, Infrastructure-as-a-Service (and their providers), both centralized and decentralized reputation systems, and security concerns and measures thereof. It explained how state-of-the-art reputation systems are used to ensure trust in a large environment by showing how ratings are made, i.e., what they contain, how they are aggregated, which restrictions they may have, and how they are collected. Furthermore, the BRAIN was introduced and discussed to build up the necessary background knowledge for integrating it into a new decentralized reputation system. To achieve a well-designed solution, multiple works on (decentralized) reputation systems, surveys on (decentralized) reputation systems, and security assessments were studied and discussed.

The solution, the *Reputation Manager* (ReM), includes many design choices that were previously proven to be the closest to a best-practice we currently have. Many requirements can stop falsified information before they are added to the system and in the case of malicious behaviour getting through, a manager has the power to hand out punishments where the data sees fit. Apart from punishing, the manager can reward truthful behaviour as well. Further, the ReM extends the BRAIN with additional functionality by storing (and holding) auctions as individual entities created by users. Instead of accepting the lowest bidder as the auction winner, the auctioneer has the chance to investigate the reputation of providers and include it in their decision of a suitable provider. Security measures are both indirectly present with the decentralization of the application and data, as well as directly implemented in the form of credibility and eligibility.

While the prototype is missing an algorithm to decide and adapt the credibility of a user after performing suspicious actions, it includes a wide set of measures to ensure that participants may only perform the actions they are supposed and allowed to. Both of the main participants have clearly defined restrictions, which are enforced either by direct requirements in the Smart Contract, or by evaluating their eligibility to perform an action. The interactions with the system follow over a web-based user interface, which has the possibilities to show current auctions, a list of providers with their reputation, and the individual ratings of a provider. Multiple use cases and scenarios interacting with the system have been shown, including visualizations of possible auctions, reputations and ratings. Some of the technical limitations such as containing high gas costs, possessing the inability to actively protect against malicious behaviour and attacks, and not having a fully fleshed out interface have been discussed as well.

In conclusion, it is safe to say that this research area still has a lot of potential. Future works including algorithms to ensure trust and to enforce certain rulesets will most likely continue to be the subjects of research and discussion. This may include algorithms that can detect not only suspicious rating values, but algorithms that analyse written reviews as well. Furthermore, a definitive way to rate and classify infrastructure providers may be needed for a more precise reputation calculation. In regard to the ReM, further development may include a more fleshed out user interface, whereas performance optimizations and a potential refactorization may reduce gas costs and the general feasibility of the solution in a real-world scenario.

Bibliography

[1] “Amazon,” https://www.amazon.com/, last visit April, 2020.

[2] “Amazon AWS,” https://aws.amazon.com/de/, last visit August, 2020.

[3] “Amazon EC2,” https://aws.amazon.com/de/ec2/?nc2=h_ql_prod_fs_ec2, last visit August, 2020.

[4] “bbc - Google deletes millions of negative TikTok reviews,” https://www.bbc.com/news/technology-52808177, last visit August, 2020.

[5] “Binance Academy - Turing Complete,” https://academy.binance.com/glossary/turing-complete, last visit August, 2020.

[6] “Bitcoin,” https://bitcoin.org/en/, last visit August 3, 2020.

[7] “Blockchain,” https://www.blockchain.com/, last visit August 3, 2020.

[8] “CHATMETER - 9 Violations that Allow Businesses to Remove Negative Google Reviews,” https://www.chatmeter.com/blog/9-violations-that-allow-businesses-to-remove-negative-google-reviews/, last visit August, 2020.

[9] “Cloud Carib - 5 Important Questions to Ask When Choosing a Cloud Provider,” https://info.cloudcarib.com/blog/5-important-questions-to-ask-when-choosing-a-cloud-provider, last visit August, 2020.

[10] “codecs,” https://docs.python.org/3/library/codecs.html, last visit August, 2020.

[11] “coindesk - Ethereum Classic Suffers Second 51% Attack in a Week,” https://www.coindesk.com/ethereum-classic-suffers-second-51-attack-in-a-week, last visit August, 2020.

[12] “CurrencyRate - 1 ETH Ethereum to USD US Dollar,” https://eth.currencyrate.today/usd, last visit August, 2020.

[13] “eBay,” https://www.ebay.com/, last visit April, 2020.

[14] “edgeuno - How to Choose Your Infrastructure and Connectivity Provider,” https://www.edgeuno.com/how-to-choose-your-infrastructure-connectivity-provider/, last visit August, 2020.

[15] "ETH GAS STATION," https://ethgasstation.info/calculatorTxV.php, last visit August, 2020.

[16] "Ethereum," https://ethereum.org/en/, last visit August, 2020.

[17] "Ethereum Smart Contract Best Practices - Known Attacks," https://consensys.github.io/smart-contract-best-practices/known_attacks/, last visit August, 2020.

[18] "Flask," https://flask.palletsprojects.com/en/1.1.x/, last visit August, 2020.

[19] "Google Drive," https://www.google.ch/drive/about.html, last visit August, 2020.

[20] "Google Play Store," https://play.google.com/store, last visit April, 2020.

[21] "Google Visualization Table," https://developers.google.com/chart/interactive/docs/gallery/table, last visit August, 2020.

[22] "Gryps - IaaS - Infrastructure as a Service," https://www.gryps.ch/produkte/cloud-computing-17/iaas/?gclid=EAIaIQobChMIzuXYg8PM6QIVt4BQBh1degBJEAAYAiAAEgIDuPD_BwE, last visit August, 2020.

[23] "Hyperledger Fabric," https://www.hyperledger.org/use/fabric, last visit August, 2020.

[24] "IBM - 5 key considerations for choosing an IaaS provider," https://www.ibm.com/blogs/cloud-computing/2018/08/21/5-key-considerations-iaas-provider/, last visit August, 2020.

[25] "IBM - Welcome to IBM Blockchain," https://www.ibm.com/blockchain, last visit August, 2020.

[26] "Internet World Stats - Internet Usage Statistics," https://www.internetworldstats.com/stats.htm, last visit August, 2020.

[27] "Investopedia - Gas (Ethereum)," https://www.investopedia.com/terms/g/gas-ethereum.asp, last visit August, 2020.

[28] "Microsoft Azure," https://azure.microsoft.com/en-us/, last visit August, 2020.

[29] "Microsoft Azure - What is Iaas?" https://azure.microsoft.com/en-us/overview/what-is-iaas/, last visit August, 2020.

[30] "Nexellent - Unsere Cloud-Dienstleistungen," https://www.nexellent.ch/en/home.html, last visit August, 2020.

[31] "Procloud - SWISS CLOUD INFRASTRUCTURE (IaaS)," https://www.procloud.ch/en/products/swiss-cloud-infrastructure-iaas/, last visit August, 2020.

[32] "py-solc-x," https://pypi.org/project/py-solc-x/, last visit August, 2020.

[33] "Python," https://www.python.org/, last visit August, 2020.

[34] “Qualium Systems - Solidity â Ethereumâs Programming Language for Smart Contraction,” https://www.qualium-systems.com/blog/useful-it-articles-and-advices/solidity-ethereums-programming-language-for-smart-contraction/, last visit August, 2020.

[35] “Remix - Ethereum IDE,” https://remix.ethereum.org/, last visit August, 2020.

[36] “Solidity,” https://solidity.readthedocs.io/en/v0.6.4/, last visit August, 2020.

[37] “Solidity Types,” https://solidity.readthedocs.io/en/v0.6.4/types.html, last visit August, 2020.

[38] “Swisscom â Purchase IT services flexibly,” https://www.swisscom.ch/en/business/sme/it-cloud/server/dynamic-computing-services.html, last visit August, 2020.

[39] “The Complete Satoshi,” https://satoshi.nakamotoinstitute.org/, last visit April, 2020.

[40] “Truffle Suite - Ganache,” https://www.trufflesuite.com/ganache, last visit August, 2020.

[41] “Trustpilot,” https://www.trustpilot.com/, last visit August, 2020.

[42] “Trustpilot - Analytics Analysis,” https://support.trustpilot.com/hc/en-us/articles/202326718-Analytics-Analysis, last visit August, 2020.

[43] “Trustpilot - Analytics Insights,” https://support.trustpilot.com/hc/en-us/articles/235803867-Analytics-Insights, last visit August, 2020.

[44] “Web3,” https://web3py.readthedocs.io/en/stable/, last visit August, 2020.

[45] “What is Centralized System,” https://bit.ly/2EUVzEQ, last visit April, 2020.

[46] J. Ahn, M. Park, H. Shin, and J. Paek, “A model for deriving trust and reputation on blockchain-based e-payment system,” *Applied Sciences*, vol. 9, no. 24, p. 5362, Dec 2019. [Online]. Available: http://dx.doi.org/10.3390/app9245362

[47] Amazon, “Customer Product Reviews,” 2020, https://sellercentral.amazon.com/gp/help/external/201972140, last visit May 4, 2020.

[48] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on ethereum smart contracts,” Cryptology ePrint Archive, Report 2016/1007, 2016, https://eprint.iacr.org/2016/1007.

[49] E. Bellini, Y. Iraqi, and E. Damiani, “Blockchain-Based Distributed Trust and Reputation Management Systems: A Survey,” *IEEE Access*, vol. 8, pp. 21 127–21 151, 2020.

[50] G. Bhatia, A. Dubey, and P. Kumaraguru, “Workerrep: Building trust on crowdsourcing platform using blockchain,” Ph.D. dissertation, 07 2018.

[51] R. Broida, “cnet - How to spot fake reviews on Amazon, Best Buy, Walmart and other sites,” https://www.cnet.com/how-to/spot-fake-reviews-amazon-best-buy-walmart/, last visit August, 2020.

[52] J. Clement, “statista - Annual net revenue of Amazon from 2004 to 2019,” https://www.statista.com/statistics/266282/annual-net-revenue-of-amazoncom/, last visit August, 2020.

[53] ——, “statista - Annual net revenue of eBay from 2013 to 2019,” https://www.statista.com/statistics/507881/ebays-annual-net-revenue/, last visit August, 2020.

[54] A. Crespigny, D. Khovratovich, F. Blondeau, K. Sok, P. Honigman, N. Alexopoulos, F. Petitcolas, and S. Conway, “Design considerations for decentralized reputation systems,” 06 2017.

[55] S. Daley, “31 Blockchain Companies Paving the Way for the Future,” https://builtin.com/blockchain/blockchain-companies-roundup, 2019, last visit April, 2020.

[56] N. Drake and B. Turner, “Techradar - Best IaaS in 2020: Infrastructure as a Service providers,” https://www.techradar.com/best/best-iaas-providers, last visit August, 2020.

[57] eBay, “Stay Safe from Scammers,” https://pages.ebay.com/securitycenter/stay_safe.html#recognize_scams, last visit April, 2020.

[58] B. Exchange, “Cryptomining: A Beginnerâs Guide to Mining,” 2019, https://medium.com/datadriveninvestor/cryptomining-explained-3ff3716c8f35, last visit August 3, 2020.

[59] D. Fraga, Z. Bankovic, and J. M. Moya, “A taxonomy of trust and reputation system attacks,” in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012, pp. 41–50.

[60] M. F. Franco, E. J. Scheid, L. Z. Granville, and B. Stiller, “BRAIN: Blockchain-based Reverse Auction for Infrastructure Supply in Virtual Network Functions-as-a-Service,” in *IFIP Networking 2019 (Networking 2019)*, Warsaw, Poland, May 2019, pp. 1–9.

[61] J. Frankenfield, “Block (Bitcoin Block),” https://www.investopedia.com/terms/b/block-bitcoin-block.asp, 2020, last visit April, 2020.

[62] T. Guardian, “Amazon blamed as ’iconic’ bookshops announce closure,” https://www.theguardian.com/books/2019/may/30/amazon-blamed-as-iconic-bookshops-announce-closure, 2019, last visit April, 2020.

[63] F. Hendrikx, K. Bubendorfer, and R. Chard, “Reputation systems: A survey and taxonomy,” *Journal of Parallel and Distributed Computing*, vol. 75, 08 2014.

[64] A. Hertig, “Ethereum 101,” https://www.coindesk.com/learn/ethereum-101/what-is-ethereum, 2017, last visit April, 2020.

[65] L.-A. Hîrţan, C. Dobre, and H. Gonzalez-Velez, “Blockchain-based reputation for intelligent transportation systems,” *Sensors*, vol. 20, p. 791, 01 2020.

[66] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan, “Repchain: A reputation-based secure, fast and high incentive blockchain system via sharding,” 2019.

[67] L. M. S. J. and E. Moreira, “An evaluation of reputation with regard to the opportunistic forwarding of messages in VANETs,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, aug 2019. [Online]. Available: https://doi.org/10.1186\%2Fs13638-019-1518-x

[68] E. Koutrouli and A. Tsalgatidou, “Reputation-based trust systems for p2p applications: Design issues and comparison framework,” 09 2006, pp. 152–161.

[69] S. Kumar, C. Diwaker, and A. Chaudhary, “Reputation system in peer-to-peer network: Design and classification,” *Journal of Global Research in Computer Science*, vol. 2, 10 2011.

[70] S. Lee, “Forbes - A Decentralized Reputation System: How Blockchain Can Restore Trust In Online Markets,” https://www.forbes.com/sites/shermanlee/2018/08/13/a-decentralized-reputation-system-how-blockchain-can-restore-trust-in-online-markets/#2b05aa00481a, last visit August, 2020.

[71] C. Majaski, “Investopedia - Distributed Ledgers,” https://www.investopedia.com/terms/d/distributed-ledgers.asp, last visit August, 2020.

[72] Z. Malik and A. Bouguettaya, “Evaluating rater credibility for reputation assessment of web services,” vol. 4831, 12 2007, pp. 38–49.

[73] K. McGabe, “G2 - 50+ Statistics Proving the Power of Customer Reviews,” https://learn.g2.com/customer-reviews-statistics, last visit August, 2020.

[74] A. Y. W. L. Y. Z. L. H. J.-N. L. Y. X. R. H. D. Ming Li, Jian Weng, “CrowdBC: A Blockchain-based Decentralized Framework for Crowdsourcing,” Cryptology ePrint Archive, Report 2017/444, 2017, https://eprint.iacr.org/2017/444.

[75] Miva, “The History Of Ecommerce: How Did It All Begin?” https://www.miva.com/blog/the-history-of-ecommerce-how-did-it-all-begin/, 2011, last visit April, 2020.

[76] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[77] W. Rash and S. Vaughan-Nichols, “PCMag - The Best Infrastructure-as-a-Service Solutions for 2020,” https://uk.pcmag.com/cloud-services/73777/the-best-infrastructure-as-a-service-solutions-for-2020, last visit August, 2020.

[78] N. Reiff, “Blockchain Explained,” https://www.investopedia.com/terms/b/blockchain.asp, 2020, last visit April, 2020.

[79] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, “Reputation systems,” *Communications of the ACM, December 2000, Vol. 43 No. 12, Pages 45-48*.

[80] A. Rosic, “Smart Contracts: The Blockchain Technology That Will Replace Lawyers,” https://blockgeeks.com/guides/smart-contracts/, 2016, last visit April, 2020.

[81] J. S., “Blockchain: What are nodes and masternodes?” https://medium.com/coinmonks/blockchain-what-is-a-node-or-masternode-and-what-does-it-do-4d9a4200938f, 2018, last visit April, 2020.

[82] scamwatch, “Online shopping scams,” https://www.scamwatch.gov.au/types-of-scams/buying-or-selling/online-shopping-scams, last visit April, 2020.

[83] M. Sharples and J. Domingue, “The blockchain and kudos: A distributed system for educational record, reputation and reward,” vol. 9891, 09 2016, pp. 490–496.

[84] D. Won, “Exodus Blog - "Ethereum Proof of Stake Date",” https://www.exodus.io/blog/ethereum-proof-of-stake-date/#head1, last visit August, 2020.

[85] Yang, Zhe and Zheng, Kan and Yang, Kan, “A Blockchain-based Reputation System for Data Credibility Assessment in Vehicular Networks,” in *IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Montreal, Canada, October 2017, pp. 1–5.

[86] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends,” 06 2017.

Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
BRAIN	Blockchain-based Reverse Auction for Infrastructure Supply in VNF-as-a-Service
CPU	Central Processing Unit
ETH	Ether
GUI	Graphical User Interface
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
NFV	Network Function Virtualization
ReM	Reputation Manager
REST	Representational State Transfer
URL	Uniform Resource Locator
USD	United States Dollar
VNF	Virtual Network Function

List of Figures

List of Tables

Appendix A

Installation Guidelines

While the ReM can be tested using the Remix-IDE (by including the ***ReM/Contracts/reputation.sol*** file), this guide will provide instructions to run the ReM on Windows 10 in a Python environment. Please use the latest Windows Updates to ensure that the installation process of following tools goes smoothly. The Windows Defender may try to prevent certain installation steps.

1. Download and install **Python 3.8.0** from ***https://www.python.org/downloads/*** and include it in a Python interpreter of your choice. ***pip*** comes with Python versions above 3.4, so it is assumed that it works correctly.

2. Download and install **Ganache**: ***https://www.trufflesuite.com/docs/ganache/quickstart***

3. This step assumes certain widely used programming tools. In the command line, navigate to the path of the ReM, navigate to ***.../ReM/Documentation*** and execute ***pip install -r requirements.txt***. If it was successful move to step 7. Otherwise, please continue with step 4 to install the necessary tools.

4. To install **solcx**, the dependencies of **solc** have to be solved as shown in
   https://solidity.readthedocs.io/en/latest/installing-solidity.html#building-from-source
   and here: ***https://solidity.readthedocs.io/en/v0.4.24/installing-solidity.html***
   Please follow the steps in the specified order:

   - Download and install **npm** and **Node.js** from ***https://www.npmjs.com/get-npm***
   - Download and install **CMake** (version 3.9+) from ***https://cmake.org/download/***
   - Download and install **Boost** (version 1.65+) from ***https://www.boost.org/***
     In the case of any complications, please refer to
     https://www.boost.org/doc/libs/1_55_0/more/getting_started/windows.html
     It may be beneficial to download and install it from here:
     https://sourceforge.net/projects/boost/files/boost-binaries/1.74.0/
   - Download and install **Git** from ***https://git-scm.com/download***

- Download and install **Visual Studio 2017 Build Tools** C++ Compiler from ***https://visualstudio.microsoft.com/de/downloads/***
  Note that this may require Visual Studio installation if it is not available. Please include the following package in the installation of the Visual Studio 2017 Build Tools
  - Visual Studio C++ core features
  - VC++ 2017 v141 toolset (x86,x64)
  - Windows Universal CRT SDK
  - Windows 8.1 SDK
  - C++/CLI support
- In the command line, execute ***npm install -g solc***
- If everything was done correctly, now executing ***pip install py-solc-x*** should work. Note that in this thesis, py-solc v3.2.0 and py-solc-x v0.8.0 were used. It is expected that the versions do not have to be exactly the ones as stated. You can check versions by executing ***pip freeze*** in the command line.

5. Download and install **Python Flask 1.1.1** by executing ***pip install Flask*** in the command line or by getting it manually from ***https://flask.palletsprojects.com/en/1.1.x/***

6. In the command line, execute ***npm install -g web3*** to install **web3**.

7. If all the steps were followed correctly, the ReM should now be able to run. Make sure everything is installed by executing the process shown in step 3 and then using ***pip freeze*** to confirm that py-solc, py-solc-x, web3, and Flask are installed. Then, please follow the steps below:

   - Run Ganache and select *Quickstart*. Make sure the port for the RPC-Server is set to 7545 (this should be the standard option).
   - Open the Python environment Python 3.8 was included in and open the ReM and configure the interpreter for the project.
   - Run ***reputation_manager.py***
     - The ReM should now be running and interactable per web browser on, e.g., ***http://localhost:5000/getProviders***
     - For the modified version with demonstrative purposes, please (terminate the ***reputation_manager.py*** if it is running) run ***demo_rem.py***, ***example_auction.py***, ***example_bidder.py***, and ***example_ratings.py*** in the specified order.

Appendix B

Contents of the CD

- ReM.zip
- midterm_presentation.pptx
- BA_Dominik_Jurilj.pdf