SCEFSTA: SMART CONTRACT ENABLED FAIR, SECURE, AND TRANSPARENT AUCTION FOR HEALTHCARE TRANSPORTATION

Owen Michael Campbell

A THESIS

Presented to the Faculty of Miami University in partial
fulfillment of the requirements
for the degree of

Master of Science

Department of Computer Science and Software Engineering

The Graduate School
Miami University
Oxford, Ohio

2024

ABSTRACT

The healthcare transport sector has progressed considerably in recent years, yet it continues to face persistent challenges such as payment, staffing, record redundancy, and resource waste. While new technologies are available, these issues remain unresolved. Medical transportation providers face challenges due to uncertainty of payment and fierce competition with peer providers in urban areas. This thesis proposes Smart Contract enabled Fair, Secure, and Transparent Auction (SCeFSTA) to create a transparent and secure auction-based platform that provides secure, immediate payment for services and promotes fair competition. We interviewed domain experts to understand the requirements for the system and designed it according to their feedback. It allows healthcare transportation providers to bid for services, ensuring that only cost-effective providers are selected. The smart contract ensures payments are made upon service completion, enhancing security for patients and providers. The blockchain-based data ledger reduces redundancy and waste while providing enhanced privacy and security. SCeFSTA alleviates staffing shortages, creates a unified data ledger, and reduces waste. We conducted a final demonstration and user survey, which proves the efficacy of the proposed system. SCeFSTA has potential to benefit the healthcare transportation field by creating a fair and efficient system that improves performance.

Table of Contents

List of Tables vii

List of Figures viii

Acknowledgements ix

1 Introduction 1

1.1 Motivation . . . 1

1.2 Proposed System . . . 3

2 Background & Related Work 4

2.1 Healthcare Transport Sector . . . 4

2.1.1 Emergency Medical Services (EMS) . . . 4

2.1.2 Inter-facility Transport . . . 5

2.1.3 Non-Emergency Medical Transport (NEMT) . . . 5

2.1.4 Multi-casualty incident . . . 6

2.2 Shortcomings of Healthcare Transportation . . . 6

2.2.1 Waste of Resources in Healthcare Transportation . . . 6

2.2.2 Staffing Shortage . . . 7

2.2.3 Funding and Billing . . . 7

2.2.4 Communication and Patient Tracking . . . 8

2.2.5 Cost of Transportation for Patients . . . 8

2.2.6 Lack of Access to Transportation . . . 9

2.3 Blockchain Technologies . . . 9

2.3.1 Features of Blockchain . . . 11

2.3.2 Potential Blockchain Drawbacks . . . 11

2.3.3 Types of Blockchain . . . 12

2.3.4 Distributed Blockchain vs. Centralized Database . . . 13

2.3.5 Blockchain Networks . . . 13

2.3.6 Smart Contracts . . . 14

2.3.7 Consensus Algorithm . . . 16

2.3.8 Programming Languages . . . 17

2.3.9 Stablecoin (DAI) . . . 18

2.3.10 Gas Fees/Transaction Fees . . . 18

2.3.11 Decentralized Application (DApp) 19
2.3.12 React 19
2.4 Cryptowallets 20
MetaMask 20
2.5 Market Competition and Fairness 20
2.5.1 Competition Through Auctions 21
2.5.2 Software Fairness 23
2.5.3 Fairness in Smart Contracts 23
2.6 Related Works 24
2.6.1 Emerging Technologies in the Healthcare Transportation Sector 24
2.6.2 Blockchain Based Auction Platforms 25
2.6.3 Blockchain Based Applications in Healthcare 26

3 Preliminary Survey and Analysis 27
3.1 Research Methodology 27
3.1.1 Defining the Application Domain 30
3.1.2 Research Question 30
3.1.3 Requirements 30
3.1.4 Design Principles 32
3.1.5 Features / Tools 34

4 Design and Implementation 36
4.1 SCeFSTA Design 36
4.1.1 Participants 36
4.1.2 Blockchain 37
4.1.3 Cryptocurrency 38
4.1.4 Sealed Bid Auction 38
4.2 Prototype Implementation and Demonstration 41
4.2.1 User Interface (UI) 41
4.2.2 Blockchain Network in Use 43
4.2.3 Crypto Wallet - Metamask 43
4.2.4 Patient Update Database 45
4.3 Technical Challenges for Implementation 45
4.3.1 User Storage 45
4.3.2 Deprecated Blockchain Networks 45
4.3.3 Deprecated React Modules 46
4.3.4 Domain Flexibility 46

5 Evaluation and Validation 47
5.1 Evaluation 47
5.1.1 Estimating Gas Prices Using Foundry Tool-chain 47
5.1.2 Tracking Time Needed for API Calls Using Truffle 51
5.1.3 Overall Requirement Satisfaction 51

5.2 Validation 52
5.2.1 Requirements Validation 53
5.2.2 Concerns and Potential Fixes 54
5.2.3 User Interface (UI) Evaluation 54
5.2.4 Efficiency and Usability Improvements 54
5.2.5 Alterations on Sealed-Bid Auction 55
5.2.6 Privacy Concerns 55

6 Conclusion **57**

7 Appendix A **59**
7.1 Unfair smart contract code 59

8 Appendix B **61**
8.1 Accounts Smart Contract 61
8.2 Auction Smart Contract 64

References **72**

List of Tables

2.1 Challenges in the disaster management system [1] 8
2.2 Pros, cons, and potential uses of the four blockchain types [2] 13
2.3 Differences between Databases & Blockchain [3] 14
2.4 Consensus Algorithm comparisons [4] 16
2.5 Benefits & Shortcomings of the four types of auctions [5] 21

3.1 Description of Interviewees 28
3.2 Potential application scenarios 29

4.1 Ethereum blockchain network vs. Avalanche blockchain network [6] [7] 38

5.1 Description of API smart contract functions in Account smart contract 48
5.2 Average price in USD of SCeFSTA's smart contract API calls (average of 100 API calls) on 3 different blockchain networks – Ethereum, Polygon, and Avalanche. The prices for Ethereum, Polygon, and Avalanche, as of October 8th, 2023 are \$1636.92, \$0.56, and \$10.09 respectively [8]. 49
5.3 Description of API smart contract functions in Auction smart contract 50

List of Figures

1.1 EMS sample use case scenario 2

2.1 Project Domain Examples [9, 10, 11, 12, 13] 4
2.2 Block structure of a blockchain segment [14, 15, 16] 10
2.3 Elements of blockchain technologies [2, 17, 18, 19, 20, 21] 10

3.1 Design cycle diagram for SCeFSTA 27
3.2 Design principle diagram for SCeFSTA 30

4.1 SCeFSTA architecture design diagram 37
4.2 Sequence diagram 40
4.3 Screenshots of the SCeFSTA User Interface 44

5.1 Gas estimate (log-scale) for different API calls (500 each). 48
5.2 Gas estimate (log-scale) with varying load factor. The load factor is the number of calls to a given API function (shown in legend). 50
5.3 Time delay estimate (in ms) for API calls with 10 API calls. 51

Acknowledgements

The author like to acknowledge Dr. Suman Bhunia for his guidance and expertise throughout this process. The author would also like to thank committee members Dr. Arthur Carvalho, Dr. Liudmila Zavolokina, and Dr. Honglu Jiang for participating on the committee and assisting when needed. The author would finally like to thank two senior capstone teams and their members: Matt Blackert, Max Beedy, Jorge Nadjar, Zach Katz, Dalong Wang, and Will Klaus. The author would also like to thank his family and his fiancé Liberty for their love and support throughout this process.

Chapter 1

Introduction

This chapter introduces the thesis and the motivation behind it. This thesis focuses on applying a blockchain based auction system to the healthcare transport sector in order to make improvements to current issues within the field. It outlines the reasoning for changes needed in the healthcare transportation sector, and the need to create a better overall experience for both the transport workers and the clients/patients. Overall, this system aims to eliminate effects of paramedic staffing shortages, create guaranteed reimbursement of services, reduce waste of resources, lower cost of healthcare transportation, create better patient tracking, and provide better access to needed healthcare transportation. These issues occur across several different domains within the healthcare transport sector, including:

- Emergency Medical Services (EMS)
- Inter-facility transportation
- Non-Emergency Medical Transportation (NEMT)

1.1 Motivation

Healthcare transportation services are an integral part of society. Over 18 million patients are given pre-hospital care from Emergency Medical Services (EMS), while roughly 15%, or 2.7 million patients, are transported to hospitals [22]. Without these services, millions of people would not be given lifesaving care and transportation. Another estimated 3.6 million people do not receive medical care due to lack of accessibility to transportation to hospital facilities [23]. There are plenty of people who require medical transportation; however, there is a shortage of competition and a staffing crisis present in the healthcare transport sector, limiting lifesaving care to society [24]. There is also an apparent issue with billing for services and reimbursement of services within the healthcare transportation sector. Often, transportation services are not paid in full, or they do not receive payment at all [25]. This is due to insurance and/or medicare, or simply from a lack of payment by the patient [25]. These issues, among others, will be outlined in the subsections that follow. Many of the issues facing the healthcare transportation field could be solved by new technologies and alterations to the current transportation systems.

There are many shortcomings to the current healthcare transportation system:

- Waste of resources
- Staffing shortages

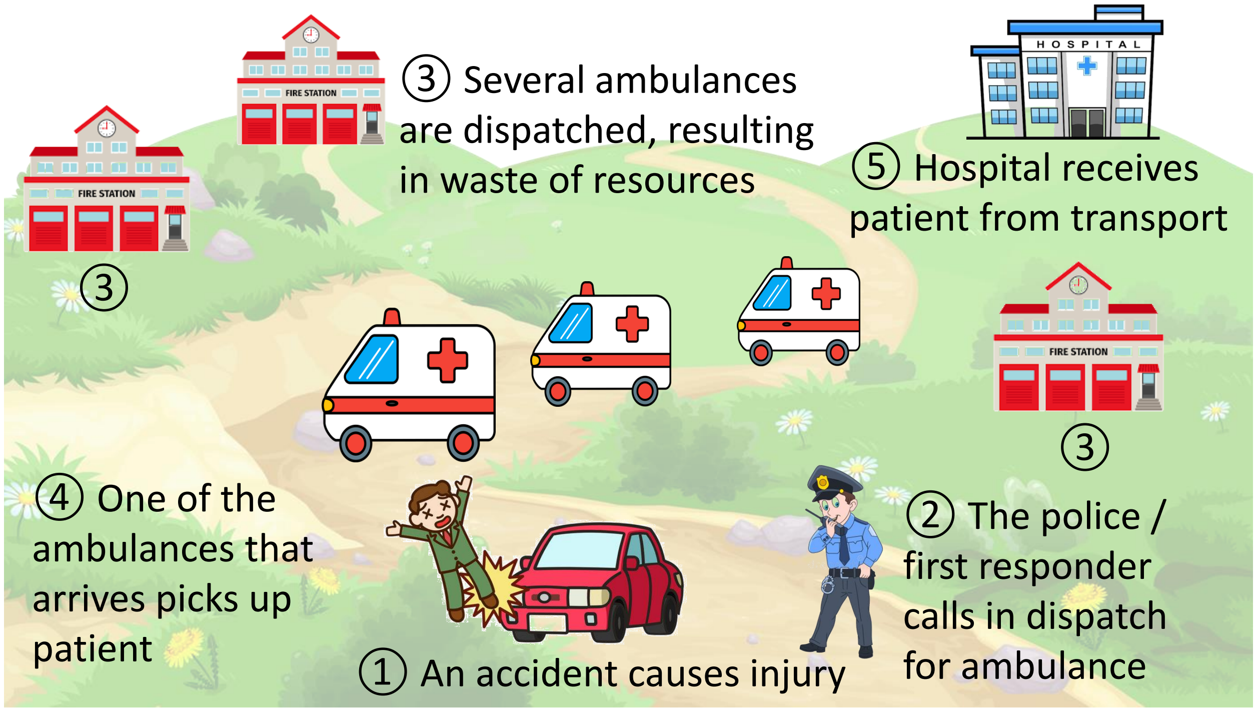

③ Several ambulances are dispatched, resulting in waste of resources
⑤ Hospital receives patient from transport
HOSPITAL
FIRE STATION
③
④ One of the ambulances that arrives picks up patient
① An accident causes injury
② The police / first responder calls in dispatch for ambulance

Figure 1.1: EMS sample use case scenario

- Funding and billing
- Communication and patient tracking
- Cost of transportation for patients
- Lack of access to transportation

Issues like the waste of resources, staffing shortages, payment from insurance, and patient tracking all result in worse care and higher costs for people forced to utilize medical transportation. Figure 1.1 represents a sample EMS use-case scenario. After an accident, police or a first responder calls dispatch for an ambulance. Multiple ambulances are sent out by different agencies. One of these ambulances, usually the first one to arrive at the spot, picks up the patient. The patient is then delivered to the hospital. From the many ambulances dispatched, there is a waste of time and resources from the ambulances who were dispatched but did not transport the patient. Each of these domains has issues that can be solved with novel technology. The system proposed could create a large positive impact on the healthcare transport sector.

Each of these barriers causes troubles for people trying to receive necessary medical care and could be improved by new technologies. Without a proper method of contracting healthcare transportation, it may be impossible for a person to get to a medical facility that can provide them the care that they need for a fair price. A detailed description of each of these barriers is provided in the Section 2.2

This research focuses on creating competition between different transportation companies using a blockchain-based auction system. The use of blockchain technologies would increase security and create guaranteed payment of services for transports, while also solving the effects of staffing shortages due to more transports available across multiple companies. This research also aims to create fairness throughout the application and auction system, so each transportation company has an equal chance of success. This application would aim to fix issues across many domains within the healthcare transport sector to create a better experience for patients and healthcare transportation workers. The overall goal of the software side of this thesis project is to create a working prototype that is fast, has low and stable cryptocurrency prices, creates fair competition, inherently fair software, and guaranteed payments for services. Accomplishing these goals should create a well polished prototype that could be applied to the healthcare transportation field.

1.2 Proposed System

This thesis project aims to make the following contributions:

1. Utilize blockchain technologies to build a secure sealed bid auction platform to be used within the healthcare field.
2. Create fair competition, through auctions, between private transport companies where competition is lacking, or otherwise non-existent.
3. Find a suitable blockchain network that is able to provide stable and reliable prices for users.
4. Find suitable domains for the system through client interviews.
5. Interact with domain experts to assess application effectiveness.

We are proposing a system which allows transport agencies to operate without needing a centralized controller for trust. In this platform, we build a blockchain smart contract through which the transportation agencies and transport hirer interact with each other, transparently, with full trust of payment security. In this architecture, the transport hirer opens a sealed-bid auction for a short duration. During this duration, the transport agencies can bid for price of transportation, and the lowest bid wins the auction. After winning the auction, the transport proceeds with delivery before the due date. The smart contract also enforces penalty of late delivery, or failure to deliver the patient to the verifier. This process eliminates the wastage of transportation resources by enforcing only one medical transport is dispatched for the job.

We generated design requirements and necessary features for the system through interviews with stakeholders of the system. This allowed us to design SCeFSTA with these features and requirements in mind. We then implemented the application through a Decentralized Application (DApp), which can be accessed by users through a deployed website. Once the application and smart contracts were fully developed, we evaluated the smart contract API based on transaction costs and API latency, and finally we validated that we satisfied the design requirements of the application in a user study.

Chapter 2

Background & Related Work

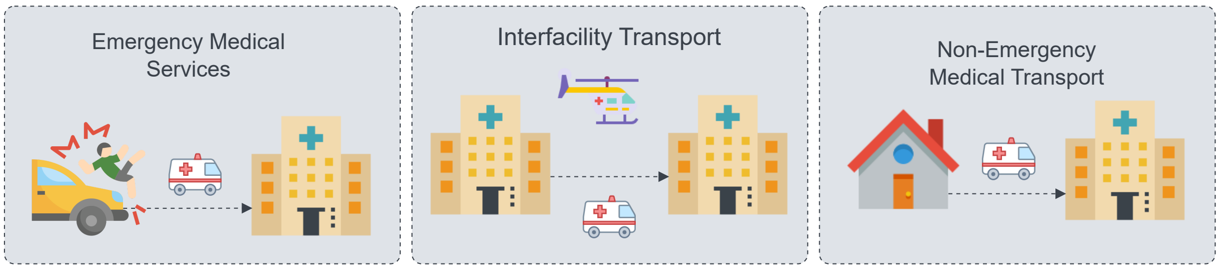
In the previous chapter, the motivation for this thesis project was outlined. In this chapter, the major topics of this thesis project will be covered. These topics include: the healthcare transportation sector and its shortcomings, the blockchain and its major components, market competition and fairness through smart contract auctions, and published research that are related to this thesis project.

2.1 Healthcare Transport Sector

This section of the background outlines the components of the healthcare transport sector. This includes, Emergency Medical Services, inter-facility transport, Non-Emergency Medical Transportation, and multi-casualty/disaster response situations. Figures 2.1a, 2.1b, and 2.1c showcase a simple scenario/ scope of the domains: emergency medical services, inter-facility transport, and NEMT. Each of these have different contributions to the healthcare transport sector and have integral functionalities in healthcare response.

2.1.1 Emergency Medical Services (EMS)

The current Emergency Medical System (EMS) is an intricate system in which each component plays a vital role in providing medical care for those in need. This system is typically deployed after an incident that results in injury or major illness. The EMS is interconnected with other services and is at the intersection of public health, public safety, and health care [26]. There are five phases



(a) Sample EMS Transport scenario (b) Inter-facility Transport scenario (c) Sample NEMT Transport scenario

Figure 2.1: Project Domain Examples [9, 10, 11, 12, 13]

to current EMS software, as outlined by AIM Software and Services. These five phases are as follows: Call to Dispatch: an incident is diagnosed and 911 is contacted, Dispatch to Electronic Patient Care Reporting (ePCR): dispatchers receive the call and dispatch ambulances, ePCR to Quality Assurance/Quality Improvement (QA/QI): ensures all data and supporting documentation, including treatments and medications used, are in the reports, QA/QI to Billing: generate billing information for the transport, Billing to Reimbursement: receiving funds for the transport [27]. The fourth and fifth phases of this system in particular take a long period of time due to Medicare and insurance; resulting in no payment for provided transportation services for months. This system could be revamped to create more competition between companies. The EMS system is a vital system in society, but has deep-rooted flaws that need solutions. A sample EMS scenario in which a person is injured and transported to a hospital is outlined in Figure 2.1a.

2.1.2 Inter-facility Transport

Inter-facility transports are transports of patients between healthcare facilities. These services are typically used to transport a critical patient from an ill-equipped facility to a facility better equipped to handle the patient's needs [28]. Inter-facility transport is a common form of healthcare transport, and it provides business to many healthcare transport companies. It is common for healthcare facilities to use the same private transport companies, with little to no competition between companies.

Air-ambulance transport is a type of inter-facility transport that is common in trauma scenarios. Air-ambulance transport utilizes helicopters to move a patient in between facilities, allowing for much faster transportation than ground transportation. Air-ambulance also allows the patient to receive life-saving care sooner [29]. This is because helicopters are able to travel much faster than ground vehicles, and they are able to avoid traffic on the streets. Opening up inter-facility transport to helicopter travel offers a much higher billing revenue than traditional ground inter-facility transport due to the higher skill required of workers, and the typical higher severity of patient trauma [29]. Utilizing inter-facility transport, both ground and air, is a large aspect of healthcare transportation services that play a large role in patient care. A sample inter-facility transport scenario in which a patient is transported from one hospital to another for both ground and air transportation is outlined in Figure 2.1b.

2.1.3 Non-Emergency Medical Transport (NEMT)

Non-Emergency Medical Transports are often used by nursing homes and senior citizens. These patients are unable to transport themselves from their residence to a doctor's appointment, so they need to independently hire transportation [30]. There is not currently a clear location for senior citizens to find needed healthcare transportation, so they are required to find their own means of contacting a healthcare transportation company in order to receive transport. A sample NEMT scenario is outlined by Figure 2.1c.

2.1.4 Multi-casualty incident

A multi-casualty incident (MCI), also known as mass-casualty incident, is a large scale incident that results in multiple casualties. An MCI can range from an active shooting event with only a few casualties to a natural disaster that results in hundreds to thousands of casualties. These events typically result in depleted resources for injured, a shortage of medical personnel, and poor communication due to the chaos of the incident [31]. A few examples of incidents that could lead to an MCI are [31]:

- **Active Shooter Situations:** Active shootings that result in several casualties.
- **Explosions:** Whether intended or an accident, casualties from explosions can be classified as an MCI.
- **Natural Disasters:** Natural events such as: hurricanes, tornadoes, earthquakes, etc. that results in a large number of casualties.
- **Multi-Vehicle Accidents:** Multi-car pileups, the more cars involved in an incident, the higher likelihood of being classified as MCI.
- **Mass Transit Mishaps:** Plane crashes, train derailments, or any incident with transport that has a large number of people crowded on board.
- **Terrorist Acts:** An umbrella term that could take the form of some of the other examples listed. This is an attack based off of political or ideological views that results in multiple casualties.

There are many more examples of MCIs that were not listed, but the general theme of MCIs are several casualties from a single event.

2.2 Shortcomings of Healthcare Transportation

This section outlines the different shortcomings in the domains of the healthcare transport sector. These shortcomings include: waste of resources, staffing shortages, funding/billing, communication/patient tracking, cost of transportation, and access to transportation. Each of these issues results in worse patient care or worse working conditions for paramedics.

2.2.1 Waste of Resources in Healthcare Transportation

A large obstacle in the healthcare transportation sector is the waste of resources. It is common for multiple healthcare transportation companies to service the same region, leading to a lack of optimization. In large cities, assignment of patients to specific companies is effectively random, with no prioritization for a specific provider [32]. Random assignment of transport providers can lead to waste and delays due to a distant transport being dispatched to an incident, rather than a more nearby transport. This is very common in large cities, since the cities contract both public and private healthcare transportation companies to satisfy the city's EMS transportation needs [32]. In addition to this, it is not uncommon for multiple transports to be dispatched for a single call. Resulting in the first to arrive on the scene to get the patient, and the other transports have wasted time and resources on the call. The larger number of transportation companies that are contracted for EMS services in a city, the greater the possibility there will be inefficiencies, resulting in higher prices and/or worse care for patients. By creating a better coordinated system where only one ambulance is dispatched per call, waste can be reduced and time will be saved.

2.2.2 Staffing Shortage

Staffing shortages are a major issue facing the healthcare transportation sector. A worsening staffing shortage has a negative impact on the quality of patient care. This staffing shortage stems from a lack of qualified workers and poor work environments for paramedics [33]. Without qualified workers, companies are unable to support the number of transports needed; leading to an increased number of transports per transport team. The larger responsibility of workers due to shortages results in greater worker burnout [33]. Providing a solution to the effects of shortages is paramount because the worse the shortage gets, the worse the care patients receive. The shortages also directly correlate to worse work environments for paramedics. This shortage is the difference between having needed transportation or not having transportation that could provide life-saving care [24]. Pay increases have been proposed to attempt to make the healthcare transportation field more enticing for potential workers; however, many companies would not have the funds to support this due to poor medicaid reimbursement [24]. There are many factors that have lead to staffing shortages, and serious changes need to be made to remedy this pressing issue. By creating greater competition between companies, the effects of these staffing shortages can be eliminated due to a larger number of companies available.

2.2.3 Funding and Billing

Currently, healthcare transportation companies are forced to wait a long period of time for payments. In addition to this, not receiving full payment of services results in healthcare transportation companies struggling to earn enough revenue to stay afloat. This stems from insurance and medicaid reducing the amount of reimbursement paid to transportation companies after the delivery of patients. This is forcing ambulance transport providers to try to find alternative funds or cut costs due to a decrease in reimbursement from private and public insurance [34]. One transportation company stated that it costs a transport company over $250 for a typical non-critical transport call; however, the amount of money the company receives for their service is roughly $107 [24]. This reimbursement rate for services is only 43%, a significant reduction in revenue. This result in massive losses for transport companies due to the fact that they are not being paid in full for the vital services they provided. This reduced revenue makes it difficult for healthcare transport companies to pay the paramedics a fair salary for the life-saving expertise they provide, and this directly influences the staffing shortage crisis [24]. A system with more guaranteed and transparent payments could be very beneficial to creating better overall payment for the services that these transportation companies provide.

Table 2.1: Challenges in the disaster management system [1]

Challenges	Potential Solution
Coordination	Create organizational charts, a chain of command, and contact information that can be easily accessed and easily updated
Communication	Have a place to create communications, status boards, and maps with team locations
Information	Create a ledger that is used by all teams to handle all patient information. Have a view similar to Google Maps showcasing nearest hospitals, shelters, and other healthcare resources
Logistics	Use ledger to calculate needed provisions and supplies in shelters. Provide shelters real-time updates on when supplies become available

2.2.4 Communication and Patient Tracking

One of the biggest sources of complexity in handling MCIs or disaster scenarios is communication and patient tracking. The four major challenges in handling MCIs are [1]:

- **Coordination:** Coordination issues arise from a large number of teams from different fields with no standard set of procedures.
- **Communication:** Communication issues are very similar to coordination issues, and stem from a lack of interoperable radios for teams to correspond.
- **Information:** Access to a standard set of information is a challenge to disaster response. Patient data can end up with redundancy in some cases and no data reported in other cases due to a lack of a single ledger for all patient data.
- **Logistics:** Without proper patient data, it is difficult to move numerous people with an appropriate amount of supplies, creating a high level of uncertainty. This includes not having enough transportation to handle those affected by the disaster.

Each of these challenges could be handled by implementing specific disaster response functionalities. Table 2.1 represents potential solutions to the challenges in multi-casualty situations.

The biggest challenges that can be changed by specified software technology are the information and the logistic challenges, as outlined in Table 2.1. Software can be designed to better handle patient information that can be directly translated into the needed logistics for handling people affected by a MCI. Creating a better method of coordination and patient tracking in a disaster scenario would provide a tremendous asset in handling MCIs or disaster responses.

2.2.5 Cost of Transportation for Patients

Healthcare transportation services can be very costly for those who are forced to use them. It is often not possible for patients in critical condition to choose the care they receive, potentially leading to extremely large medical bills. It is not uncommon for people who use healthcare transportation to be hit with “surprise costs” due to specific care they received [35]. Current issues in the healthcare field are driving up costs of transportation for users, creating a lower usage in healthcare transportation services. This is a major issue in trust of healthcare transportation, since people in need of healthcare transportation fear the costs they might incur if they are forced to use healthcare transportation.

2.2.6 Lack of Access to Transportation

In more rural areas, it can be difficult for people to find reliable healthcare transportation. It can also be difficult for people to make it to medical centers based on physical requirements or from having unreliable personal transportation. There are a few different transportation barriers for people in need of medical care [36]:

- **Vehicle Access:** Not every person in the United States has access to a vehicle. People without their own vehicle have to resort to asking others to provide service, or hire a transportation company to deliver them to a medical center.
- **Geographic Barriers:** Those living in more rural areas are forced to travel long distances to get to the nearest available medical center, and this medical center may not even be equipped to safely serve the patient, requiring further transportation to a better facility.
- **Physical Requirements:** People with disabilities, elderly people, those with specific conditions, or people who have sustained severe injuries may require paramedics and specialized transportation in order to safely make it to a medical facility.

Each of these barriers causes troubles for people trying to receive necessary medical care and could be improved by new technologies. Without a proper method of contracting healthcare transportation, it may be impossible for a person to get to a medical facility that can provide them the care that they need for a fair price.

2.3 Blockchain Technologies

A blockchain is an immutable, distributed, and decentralized database/ledger. A blockchain is composed of “blocks” which store information in the blockchain, and each new block is linked to the previous block. Figure 2.2 showcases the structure of a segment of blocks in a blockchain. It highlights the connection between blocks, where each block holds the previous block address, a header, a timestamp, and other accessory information. This block structure is what makes the blockchain such a powerful tool for applications.

The block structure creates an immutable timeline of data that is permanently stored in the blockchain [37]. Blockchain technology is a powerful tool that can be harnessed to create secure, highly organized, and immutable data in a decentralized manner. The makeup of a blockchain is outlined by Figure 2.3. The main components of the blockchain are: blocks, distribution of the blocks, the distributed ledger the blocks form, transactions with the blockchain, and confirmation of transactions [2].

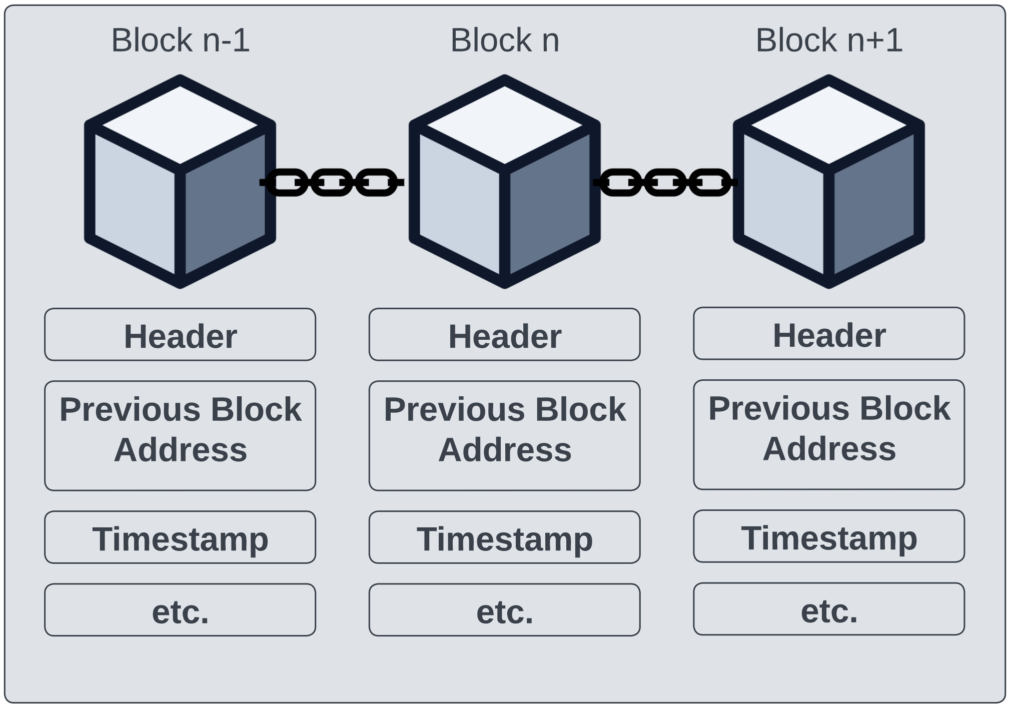


Figure 2.2: Block structure of a blockchain segment [14, 15, 16]

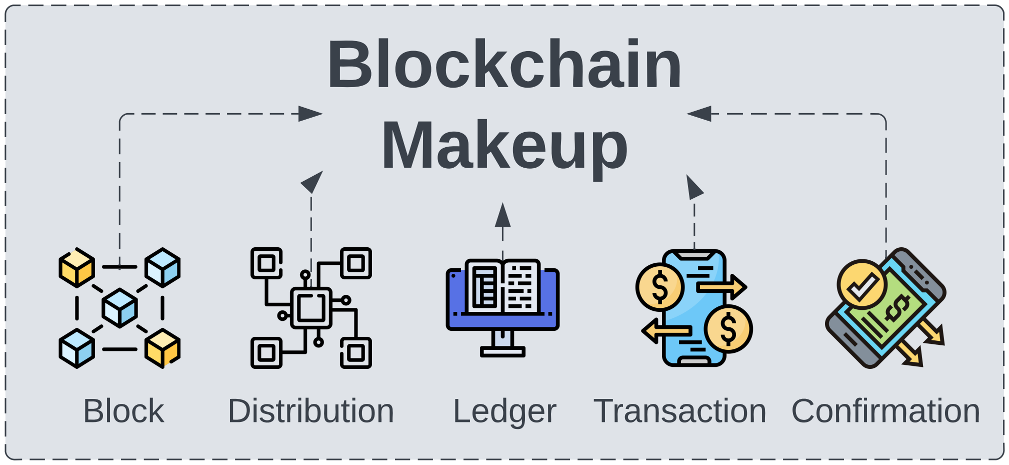


Figure 2.3: Elements of blockchain technologies [2, 17, 18, 19, 20, 21]

Each of these components are integral to the structure and functionality of blockchains. Emerging blockchain technologies have various applications due to their distributed nature and other features of the blockchain [38]. Although blockchain technologies can be an impressive tool, the use of blockchain does have drawbacks; issues like scalability and security are issues that have to be handled to make blockchain technologies viable [39]. There are several features of blockchain that make it a powerful tool that can create transformative applications, if applied correctly. Overall, a blockchain must be handled properly by use the proper safeguards to fix the issues of blockchains, and allow for taking advantage of the features of a blockchain.

2.3.1 Features of Blockchain

The features of blockchain create strong connectivity and quick transactions to the blockchain that is distributed across a network. Blockchain has many key features that enable it to be a powerful tool for developers [38, 37]:

- **Easily Accessible:** A deployed smart contract on the blockchain can be easily accessed through the contract's API calls. Good accessibility fosters strong connectivity for web or mobile applications that are connected to the same network as the blockchain.
- **Open Source:** The ability to develop and deploy individual smart contracts allows for the blockchain to be used for a variety of unique applications. Developers can gear their smart contract to the functionality required for a specific project.
- **Transparent:** All transactions on the blockchain are transparent and visible to anyone who is able to view the network. These transactions also have timestamps that showcase when the transaction was made.
- **Distributed:** The blockchain uses distributed networks, meaning data is shared between nodes of a computer network. Keeping all the data in a decentralized database can keep information from being lost due to equipment failure or a malicious attack.
- **Immutable:** All information posted to a block in the blockchain is irreversible and permanently recorded. This allows for permanent record keeping that can never altered, an important feature when handling sensitive data.

2.3.2 Potential Blockchain Drawbacks

The drawbacks of blockchain come from a lack of regulation in cryptocurrencies, redundancy in data sets due to the block structure, as well as security and scalability. Given the nature of the blockchain, it comes with several potential drawbacks [39]:

- **Lack of Regulation:** Given there is no way to control crowdsourced servers, it is difficult to create regulations for cryptocurrencies. Cryptocurrencies can be very unstable and operate outside the law; therefore, they cannot be regulated by government entities. Without regulation, prices for cryptocurrencies can have significant variations, depending on the provider.

This leads to confusion and lack of predictability of costs. To solve this, “stable” cryptocurrencies have emerged, called Stablecoins [40], to try to create reliably priced cryptocurrency.

- **Redundancy & Scalability:** Every transaction is recorded on the blockchain, resulting in increased costs due to redundant information being stored. These redundancies also result in a larger amount of storage needed for storing data. By not getting rid of old data that has been updated, the blockchain must make storage for all data posted to it. One solution to scalability issues is choosing a good consensus mechanism, a fast consensus mechanism is able to mitigate the effects of a large blockchain structure [41].

- **Privacy Issues:** Since transactions on the blockchain are inherently transparent, keeping data stored on the blockchain can be difficult. Data is also stored on a shared ledger, creating multiple points of attack for potential security breaches. Depending on the application, data security can be very important. Information stored on the blockchain could potentially be legally private data, such as medical or school records. Security issues of blockchain have largely been solved by blockchain networks.

- **Irreversibility:** Although immutability can be viewed as a feature, it can also be a constraint on a program. Not having the ability to revert transactions requires extra steps if issues arise. These extra steps needed to fix a bad transaction can be costly and take a great deal of time to fix, so transactions need to have proper safeguards and error checking, in order to avoid complications.

Although there are many inherent flaws in blockchain, many of the flaws and issues have been solved during developments in the field.

2.3.3 Types of Blockchain

There are four main types of blockchains. *Permissionless blockchains*, also known as public blockchains, are distributed blockchain ledgers that are accessible to any person with an internet connection. These users are also able to access any of the previous transactions, any historical records, or any contemporary records [2]. *Permissioned blockchains*, also known as private blockchains, are typically ran and controlled by a small network through a private firm or organization. This blockchain exists in a private network, which limits the accessibility to only those within the host organization [2]. *Hybrid blockchains* attempt to find a balance between the trade-offs of public and private blockchains. The private side of the hybrid blockchain holds keeps private data secure while giving an organization control over what information can be accessed on the public blockchain. *Consortium Blockchains*, also known as federated blockchains, share similarities with hybrid blockchains. It provides features from both the public and private blockchains, however it incorporates different organizational members working on a decentralized network. Consortium blockchain also utilizes validator nodes that have the role of initiating, receiving, and validating transactions [2]. Each blockchain type has varying functionality, resulting in different benefits and shortcomings accompanied by choosing one over the other. Table 2.2 showcases some of these benefits, shortcomings, and applications of each of the four types of blockchains.

Table 2.2: Pros, cons, and potential uses of the four blockchain types [2]

	Public	**Private**	**Hybrid**	**Consortium**
Pros	Trustable, Secure, Open, Transparent	Speed, Scalability	Secure, Cost-Effective	High Security
Cons	Low transactions per second, Low scalability, High Energy Consumption	Trust Building, Low Security, Centralization	Less Incentive, Lack of Transparency	Lack of Transparency
Potential Uses	Voting, fundraising	Supply Chain Mgt., Asset Ownership, Internal Voting	Real Estate, Retail, Highly Regulated Markets	Banking & Payments, Research, Food Tracking

From Table 2.2 it is evident that public, hybrid, and consortium blockchains have much better security than private blockchains. This is due to the fact that they do not rely on a single network and are distributed across many networks, unlike private blockchains. Private blockchains however succeed in speed and scalability, while public network's shortfall is its low transactions per second and low scalability [2]. There are many pros and cons associated with the different types of blockchains, and the desired functionality of the blockchain can heavily dictate the best type to utilize.

2.3.4 Distributed Blockchain vs. Centralized Database

One of the most common storage ledgers are databases. Databases utilize a centralized ledger to store data in a well-structured way [3]. There are many benefits of using databases and core features that make it one of the best ways to store information; however, the core features of the distributed blockchain technologies makes it a strong competitor for storing data and processing cryptocurrency transactions. The differences between blockchain technologies and databases are outlined in Table 2.3

Ultimately, the best storage solution to use relies heavily on the intended usage. It is also common for blockchain technologies to be utilized along with centralized databases inside an application to get the best features out of both options.

2.3.5 Blockchain Networks

Blockchain networks provide technical infrastructures for processing transactions with the blockchain. Blockchain networks also allow for easy access to the blockchain ledger and the smart contract, creating better connectivity for applications [42]. There are multiple types of blockchain networks and many different networks for each type with differing functionality.

Table 2.3: Differences between Databases & Blockchain [3]

	Blockchain	**Database**
Authority	Decentralized, not controlled by a single user	Centralized by nature and controlled by an administrator
Architecture	Peer-to-Peer	Client-Server
Integrity	Data can not be altered	Data can be altered by malicious users
Cost/Maintenance	Hard to implement & Maintain	Easy to implement & maintain
Performance	Slower due to verification & consensus methods	Extremely fast & highly scalability
Use Cases	Monetary transactions, DApps, and data verification	Store confidential information, relational data, and apps w/ continuous flow of data

There are multiple factors to consider before selecting which network to implement into a project. Speed, cost of transactions, security, and user base are important factors of blockchain networks to consider. Four highly regarded networks are, Ethereum, Avalanche, Ethereum Layer2 Polygon, and Fantom. Each of these options displayed strong connections to each of the valued factors. Ethereum is an expensive network with slow transaction throughput when compared to other options. However, Ethereum has a very large user base and a lot of support from the community and also has its own currency, ETH [43]. To remedy some of these issues, Ethereum released separate blockchains that extend Ethereum called Ethereum Layer2. Layer2 options aim to reduce costs, increase throughput, all while maintaining Ethereum tools. Polygon is an example of a Layer2 extension that has much lower gas price costs and higher throughput, but does have less supporting documentation compared to other networks [44]. Fantom is one of the fastest networks available, but it too is missing strong documentation and a large user-base [45]. Avalanche is an increasingly popular competitor to Ethereum. It is growing rapidly and is stated to be the leading blockchain for the financial services in the web 3.0 economy [45]. Avalanche also has high transaction throughput with low transaction costs when compared to other networks. Avalanche also has its own currency, AVAX, which provides a different option from the highly regarded Ethereum [46]. There is no shortage of blockchain networks to choose from, and each network offers benefits and trade-offs for creating a blockchain that is tailored to a specific purpose.

2.3.6 Smart Contracts

Smart contracts are executable codes created to enforce an agreement between untrustworthy parties, without an intermediary third-party running on top of the blockchain. Smart contracts accomplish this by using automated transactions to codify agreements and create trustworthy relationships [47]. This was derived from Bitcoin [48], created by Satoshi Nakamoto in 2008, which was the first cryptocurrency. Bitcoin introduced blockchain as a peer-to-peer, distributed, infrastructural technology and paved the way for new blockchain technologies. There are three distinct components present in smart contracts [47]:

- A contractual arrangement between parties.
- Preconditions necessary for contractual obligations to occur.
- The execution of the smart contract.

Using defined functions, a smart contract can define constructors for smart contract deployment, a destructor for tearing down the smart contract, and other functions used through transactions [47]. When using a constructor, a new instance of that smart contract is deployed on the blockchain. In this instance of the smart contract, the deployer has become the owner of the smart contract and is now the only person who can utilize the constructors and destructors of that instance. Each smart contract has variables, functions, events, and structures that can potentially be encapsulated in a class [47]. This allows for a strong development environment for creating the desired functionality of any program.

There are four main phases in the life cycle of a smart contract - Create, Freeze, Execute, and Finalize. Each of these phases in the smart contract life cycle is essential to creating a fully functional smart contract that supports transactions [49].

- **Create:** The first phase of the smart contract life cycle is defining the smart contracts objectives. Before writing the code behind the smart contract, participating parties have to derive the contract's overall content and goals. Each participant must also have a cryptowallet for identification and transferring payments in order to participate. Once the contract's goals have been defined, the requirements can be transitioned into code using a programming language, then the contract can be submitted to the blockchain.
- **Freeze:** Once the smart contract is submitted to the blockchain, it is frozen and must be confirmed. The submitter must pay a transaction fee for deploying the blockchain, after payment is received the contract is successfully deployed and open to the public through the public ledger. Before the contract is successfully deployed all transactions will fail, but once payment is received and the contract's pre-conditions for execution are verified, the contract is unfrozen.
- **Execute:** Now that the smart contract is no longer frozen, it is ready to be executed. During this phase the contract's integrity is verified, the code is compiled and interpreted, generating a new state and a new set of transactions for the smart contract. The new information is committed to the distributed ledger and verified using a consensus mechanism.
- **Finalize:** Once the new transactions and updated state are confirmed by the consensus mechanism, the frozen committed digital assets are transferred, and the smart contract is now fully completed and ready to confirm all transactions.

Table 2.4: Consensus Algorithm comparisons [4]

	Proof of Work	**Proof of Stake**	**Hybrid Forms**
Energy Efficient	Low efficiency	High efficiency	Low-Medium efficiency
Block creating speed	Slow	Fast	Slow
Vulnerable to Double Spending attack	High vulnerability	Low vulnerability	Medium vulnerability
Used by	Bitcoin, Ethereum	Avalanche, Cardano, nextcoin	PPcoin, Blackcoin

2.3.7 Consensus Algorithm

Consensus Algorithms in blockchains are used to manage cooperation between blocks in a blockchain. In a peer-to-peer (P2P) network, such as blockchain, there can theoretically be any number of validators. This creates a need for a consensus algorithm in the blockchain that manages cooperation between the peers, or blocks, in the blockchain [41]. A few common forms of consensus algorithms in blockchain are, Proof of Work (PoW), utilized by cryptocurrencies like Bitcoin [48] and Ethereum [43], Proof of Stake (PoS), utilized by Cardano [50], and Ripple Protocol Consensus Algorithm (RPCA) used by Ripple [51] [41]. Other blockchain networks have developed or altered existing algorithms to tailor specifically to their network's needs. Table 2.4 showcases the major differences between the Proof of Work, the Proof of Stake, and the hybrid forms of Pow and PoS consensus algorithms.

From Table 2.4 it is evident that Proof of Stake contracts are more efficient, faster at creating blocks and are less vulnerable to double spending attacks [4]. An attack that results in receiving payment twice, or paying twice due to poor consensus between peers. These benefits of PoS are why large blockchain networks like Ethereum [43] have transitioned from PoW to PoS consensus algorithms [4]. There are many consensus algorithms available to choose from, but that does not mean all were made equal. Some consensus algorithms offer higher speeds and more security than others, creating an overall better product.

Avalanche [46] developed its own consensus mechanism called the Snow consensus protocol. It was developed using Proof of Stake as the foundation to create a balance between transaction speed, network capacity, decentralization, efficiency, and security [52]. This consensus algorithm is a "leaderless" protocol, such that no single blockchain node has to be the leader who control the actions of the group. Instead, a validator node receives a transaction and samples neighboring validators at random to see if they agree. This continues to repeat across other random neighboring validators, until the network has a consensus on the transaction's validity. [52]. By developing their own consensus mechanism, Avalanche [46] was able to create a mechanism that best fits its network.

2.3.8 Programming Languages

Several programming languages for blockchain programming have emerged over the last few years. Solidity, Pact, and Liquidity are prime examples of programming languages designed specifically for developing smart contracts [53].

- **Solidity**: A statically-typed programming language used for implementing smart contracts on Ethereum. This applies to any contract or network running on Ethereum Virtual Machine (EVM). Solidity is a very popular language, largely due to the popularity of Ethereum and its high connectivity.
- **Pact**: A programming language tailored toward Kadena Blockchain. It is immutable, Turing-incomplete, and utilizes a declarative approach instead of a complex control-flow. It was developed to make bugs easy to find and hard to spot.
- **Liquidity**: A fully typed functional language that strictly compiles using Michelson security restrictions. Michelson security restrictions is a formal verification framework that is used to prove correctness of smart contracts.

The code below utilizes the Solidity smart contract programming language. This code showcases a sample smart contract that includes a constructor, a destructor, and many of the core data types within Solidity [54, 55].

```
pragma solidity ^8.0.0
// initalize the smart contract owner
address contractOwner;

contract SampleContract {
    // sample constructor for a solidity smart contract
    constructor() public {
        contractOwner = msg.sender;
    }

    // sample destructor for solidity smart contract
    function done() public {
        selfdestruct(contractOwner);
    }

    // statically typed variables
    int sampleInt = -1; // signed integer, abreviation for int256
    uint256 sampleUint = 1; // unsigned integer
    // ethereum address, can be address
    // or address payable for transactions
    address addr = "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c";
    bool boolean = true; // boolean, two fixed values, true or false
    bytes32 sampleByte = "Hello World!" // 8-bit signed integers, stored
    // in binary
    enum <digits> {1, 2, 3, 4, 5, 6, 7, 8, 9} // user defined datatype

    // reference variables
```

```
        uint[10] sampleArray; // an array of unsigned integers, fixed size
        uint[] sampleArray2; // dynamically sized, unsigned integer array



         // a struct is a custome data type, can be made up of
         // multiple variables
        struct person {
            string name;
            string age;
            uint birthYear;
        }
        // stores data in key-value pairs, retrieve value data using
        // associated key
        mapping (uint => address) Variables;
    }
```

Each of these languages are tailored specifically to programming smart contracts and are applied to specific blockchain networks. More common languages have also evolved to provide tools for developing smart contracts. Languages like C++, JavaScript, Go, Python, and more provide solutions for developing smart contracts without being specifically created for developing smart contracts [56]. This can make development easier, due to the popularity of these common languages and large amounts of documentation on them.

2.3.9 Stablecoin (DAI)

One of the major pitfalls of blockchain mentioned before was the lack of regulation and the inability to predict costs due to changing prices of cryptocurrencies. One solution was the creation of Stablecoins. Stablecoins aim to create significantly more predictably costs when compared to other cryptocurrencies. This is done by pegging the value to another currency, commodity, or financial instrument [40]. One example of a Stablecoin is Dai [57], Dai is a currency that runs on Ethereum (ETH) and is a decentralized Stablecoin. This Stablecoin attempts to keep the price of the currency constant, aiming to maintain a cost of $1.00 USD [57]. This provides a solution to the issue of highly volatile of cryptocurrencies and creates a stable currency for blockchains to take advantage of.

2.3.10 Gas Fees/Transaction Fees

Gas fees, or transaction fees, are the fees required to perform a transaction with a blockchain. These gas fees vary in cost based on the complexity of the operation being executed on the blockchain. A very complex transaction that posts large amounts of data to the blockchain will be more expensive than a transaction this only a single variable from the blockchain. Any contract that interacts with blocks in the blockchain is considered a transaction and will have an associated gas fee [58]. Typically, a transaction cost will be a fraction of a cryptocurrency coin. To make representing transaction costs easier, denominations are commonly used to represent fractions of a cryptocurrency. For example, for ETH on the Ethereum network, small amounts of ETH are represented by wei. A wei is 10^{-18} of one ether. A Gwei is the largest form of wei, which is 10^{-9} of an ether. Other denominations are used for different networks and currencies, i.e. satoshi for bitcoin and nAVAX for Avalanche [59]. Gas Fees are an instrumental part of blockchain and its decentralized nature because it provides compensation for the miners.

2.3.11 Decentralized Application (DApp)

A Decentralized Application (DApp) is a combination of a blockchain smart contract and a user interface to create a full-stack application for users [60]. This combination replaces the traditional centralized database with the decentralized blockchain. The application code has the back-end code running on a decentralized peer-to-peer network, and the front-end code is written in any front-end programming language. There are four types of DApps, as defined by Ethereum.org [60]:

- **Decentralized:** DApps that operate on an open decentralized platform, such as Ethereum.
- **Deterministic:** DApps that perform the same function regardless of the environment they are applied.
- **Turing Complete:** DApps that, given the required resources, can perform any action.
- **Isolated:** DApps executed in a virtual environment. This isolates the smart contract so if there is a bug it will not affect functioning of the blockchain network.

The different types of DApps allow for a variety of application scenarios.

2.3.12 React

React [61], a JavaScript library for building user interfaces, provides DApp frameworks for allowing for easier development and integration with a smart contract. There are a few key features of a React DApp that have to be considered during development [62]:

- **Open-source:** code is open for everyone, so users are able to read and check it.
- **Decentralized:** due to blockchain structure, a React DApp is not concentrated on a central computer.
- **Automation:** since the programs are running based off the smart contract, no third party is responsible for running the DApp.

React can be used as a DApp developing tool that provides several libraries to connect with smart contracts. It can be adapted and developed to an application's specific needs. DApps, overall, provide connectivity with a blockchain smart contract to allow users to have an interface with the blockchain's API.

2.4 Cryptowallets

Users need a secure place to store their private keys for accessing their cryptocurrencies. Losing a private key means losing all the cryptocurrency linked to the key, and poor security could lead to cryptocurrencies being stolen [63]. Cryptocurrencies are typically held in cryptocurrency software wallets, otherwise known as cryptocurrency-wallets. This allows users to securely store their cryptocurrencies and tokens in a single location. There are two types of crypto-wallets. *Hot wallets* are cryptocurrency-wallets that are connected to the internet and are readily accessible to those with an internet connection. *Cold wallets* are not connected to the internet and are less convenient to access, but are more secure than hot wallets [63]. Hot wallets are necessary for applications that have frequent transactions, so transactions can be made instantly and payments can be confirmed upon deliver of a patient. There are several popular hot wallets in the marketplace such as: Crypto.com's Defi Wallet, Guarda, Exodus, Coinbase Wallet, and Metamask are among some of the top hot wallets available in 2023 [64]. Each of these wallets listed boast good reviews and strong functionality. Hot cryptocurrency-wallets can be utilized to perform many transactions including, buying, trading, lending, or earning cryptocurrency.

MetaMask

MetaMask [65] is a popular cryptocurrency hot wallet. MetaMask allows for interactions and monetary transactions with the blockchain. It is a simple google extension which allows for ease of use for checking and sending cryptocurrencies in a user's wallet. MetaMask [65] also offers a mobile option, which allows for easy access from any mobile device with an internet connection. The essential features provided by MetaMask are broadcasting transactions, sending or receiving cryptocurrencies and tokens, securely connect to DApps, and store/manage multiple account keys in a single wallet.

2.5 Market Competition and Fairness

Competition in the economy's markets is fundamental in generating a healthy economy. Markets with firms competing for customers fosters lower prices and better products for consumers [66]. Markets that do not have strong competition are going to result in higher prices for inferior goods than if strong market competition was present. A lack of market competition results in a single entity controlling the majority of a market. Allowing them to keep other competitors from entering the market, creating an unequal playing field [66]. Auctions are a commonly used method for creating competition when trying to sell a good or service. This section will outline many types of auctions as well as their core functionalities. Utilizing blockchain based auctions can lead to a lack of fairness within an auction. Without the proper controls, it is possible to design a smart contract that does not behave as intended, creating a lack of fairness for users. When creating competition in a new market, it is paramount to consider fairness of competition to ensure a single market does not control the market, and that the applications being used to create competition behaves as intended.

Table 2.5: Benefits & Shortcomings of the four types of auctions [5]

Auction Type	Benefits	Shortcoming	Fairness	Winner
Ascending	Allows for truly free market competition for a good	Can lead to monopolistic competition	Wealthy participants can bid 1$ higher than a competitor until they win	Highest bidder
Descending	Allows for truly free market competition for providing a service	Free competition allows for undercutting competitors	Wealthy participants can undercut participants when bidding for services; removing revenue streams for other participants	Lowest bidder
Sealed-first price	Creates more balanced competition due to hidden bids	Reduces free market ecosystem for selling goods	Provides fair results that are not influenced by others	Highest bidder
Sealed-last price	Creates balanced competition for sale of services	Removes free market ecosystem for selling services	Provides fair results that are not influenced by others	Lowest bidder

2.5.1 Competition Through Auctions

Auctions are often utilized to create competition between buyers or competitors and to create better prices for sellers. There are several different aspects of auctions that can be utilized to create a fair outcome for sellers and buyers within an auction ecosystem.

Types of Auctions

There are four main types of auctions [5]:

- **Ascending auctions:** Ascending auctions are the most commonly used auctions. They consist of multiple participants placing higher bids, until none of the participants will bid higher than the current highest bid. In the ascending auction, the highest bidder wins the auction. In ascending auctions, all bids placed by participants are known by all other participants.
- **Descending auctions:** Otherwise known Dutch auctions, descending auctions work inversely from ascending auctions. In a descending auction, the participants bid lower values until no

one is willing to go lower than the lowest bid. In descending auctions, the lowest bidder wins the auction. Descending auctions can be applied to the participants trying to sell a good or service to an auction initiator, where the initiator gets the benefit of the lowest price. In descending auctions, all bids placed by participants are known by all other participants.

- **Sealed-first price auctions:** Sealed first price auctions are close-bid auctions, meaning participants are unable to see rival bids, and each participant is only granted a single bid. In the first-price auction, the highest bidder at the end of the auction period wins.

- **Sealed-last price auctions:** In a sealed-last price auction, the lowest bidder wins the auction, similar to how descending auctions operate where the lowest bidder wins. However, the bidding system works the same as a sealed-first price auction, in which each participant is only granted a single bid and the other bids are hidden.

In each of these auctions, users can only bid during the auction bidding period, but once the bidding period ends, no more bids can be placed. After which, the winning bid is announced [5]. Each of these auction types has different benefits, shortcomings, and structures that heavily influence the auction outcomes, outlined by Table 2.5.

Reserve Prices

A reserve price is the minimum offer a seller will accept as a winning bid, or the maximum offer that will be accepted in a last-price or descending auction. The reserve price will keep what is considered an unacceptable offer to a seller from winning the auction. If the reserve price is not met, the seller has no obligation to sell their product or purchase a service, depending on the auction. Sellers need to use caution when setting reserve prices because expecting too good of a price can cause disinterest in the auction; however, setting too low of a low reserve price can result in lost revenue. Reserve price is often mistaken for being the same as an opening bid, however these two are different. An opening bid is the amount suggested for starting bidding, but the reserve price is the definitive limit for a bid in an auction [67]. Utilizing reserve prices can lead to greater satisfaction and a better outcome for the seller.

Information Disclosure

Information disclosure in auctions is the release of information about previous auctions that can result in buyers' adjusting bids. Disclosing information about previous auctions can lead to more informed buyers on auction tendencies that will directly influence the amounts they will bid. Deciding the optimal amount of information to disclose to auction participants is not a trivial task and is different for every auction environment. The degree to which the seller decides to release information to bidders allows the seller to influence the auction directly [68]. Changing the level of information given to users can increase or decrease the fairness of the auction and will impact the outcome of the auction.

Bidder Collusion

Bidder collusion occurs when two buyers in an auction collude to manipulate the outcome of an auction. This results in an outcome that would not have transpired if the bidding collusion had not taken place. Bidder collusion reduces the fairness of an auction and can result in drastically changed prices within an auction. In an ascending auction, bidder collusion can drive the sale price up by a substantial amount. Bidder collusion is illegal in the United States, but can be a large source of unfairness in an auction system [69]. Bidder collusion can be difficult to locate within an auction system and could have drastic effects on the outcome of the auction.

2.5.2 Software Fairness

There are many definitions of what it means for software to be fair. One definition of software fairness is a property of software that judges whether the software exhibits biases. A bias is a prejudice toward a group that results in an unfair environment. Software fairness results in non-discriminatory behaviors of the system, despite what information is given to the software system [70]. Another definition, called group-fairness, states that the software should provide equal results to different races. By this definition, if one race gets better results than another, the software is biased. The casual fairness definition states that if two participants are the exact same except race, they should receive the exact same results. There are numerous definitions of software fairness, and it is impossible to develop a software system to satisfy every definition [71]. When software is dealing with human subjects, it is important to ensure the results of a software system is not biased toward or against a group of people, but a metric of fairness can be difficult to identify due to a highly ambiguous definition of fairness.

2.5.3 Fairness in Smart Contracts

The importance of smart contracts within a blockchain application has already been discussed. The fairness of smart contracts has not yet attracted much attention in the blockchain community. A smart contract is defined as unfair if there is a contradiction between a participant's expectations and the actual implantation of the contract rules. Fairness is individually decided by each participant, meaning one participant might think the smart contract is fair, but another participant might disagree. An application called FairCon [72] was developed to judge smart contracts and assign them a fairness score. This application focused on four main aspects of components to judge smart contract fairness. The fairness properties scored on were: truth-fullness, efficiency, optimality, and if the smart contract is collusion free. These components are essential to smart contract fairness and should be considered when implementing a smart contract. One of the largest challenges electric auctions face is a lack of trust among participants. The anonymity of the internet increases transaction misbehavior and malicious attacks [73]. By designing a smart contract that is fair, an application can build greater trust among users. Fairness in smart contracts is an often overlooked, yet essential, aspect of smart contract design that can lead to greater trust in an application.

Unfair Smart Contracts

Smart contract fairness is defined as a smart contract's ability to perform the intended functionality without being compromised. There are subtleties in smart contracts that can make them unfair [74]:

- Absence of Logic
- Incorrect Logic
- Logically correct but unfair

A smart contract function that has an absence of logic lacks controls for verifying necessary contract details. This will create an unfair smart contract for users, since the contract is not performing how the users would expect it to. This error leaves the smart contract unsafe due to a lack of safeguards that prevent logic errors [74].

A smart contract function that has incorrect logic can have syntactically correct logic, but have semantic errors that make it unfair. This is unfair because the participant was placed as the highest bidder in the auction, but the highest bid was not properly updated [74]. Incorrect logic will result in an outcome that is unexpected due to an error in the smart contract function.

A smart contract that is logically correct but unfair creates a function that can be exploited by a user to create an unfair outcome for participants. An example of this flaw within an auction system is a seller placing a large bid in an attempt to drive up other auction bids. This creates an issue with fairness among participants who are influenced to place a higher bid due to the seller forcing the bid price up [72, 74].

Sample code snippets using Solidity for these flaws are in the Appendix chapter in Section 7.1. An unfair, smart contract can have disastrous effects on trust in an application. Without proper safeguards, an unfair smart contract can ruin an application, creating an inferior product for users.

2.6 Related Works

There are applications that have been developed that have similarities to this thesis project. The works cited here show the novelty of this thesis project, but provide supporting material for different aspects of the project. The related works covered are encompassed in the following scopes: emerging technologies in the healthcare transport sector, blockchain based auction platforms, and blockchain based applications in healthcare.

2.6.1 Emerging Technologies in the Healthcare Transportation Sector

New technologies aiming to solve issues in emergency response related fields are emerging every year. These technologies provide novel solutions to present issues in the healthcare transportation sector.

One such product is *DistressNet* [75], a cloud based application to help emergency responders respond to natural and man-made disasters. DistressNet collects data, using sensing, networking and data management, to send to first responders and emergency personnel so they can better navigate and handle these disasters. DistressNet showcases how new and emerging technologies that use distributed systems are able to provide security and support in emergency response systems.

Another example is *Secure Priority Vehicle Movement Technology* [76], which is based on blockchain technology. This technology uses blockchain to securely connect autonomous vehicles, and only authorized blockchain nodes are able to make the vehicle move. The prototype outlined has three main cases for Attack Scenarios on priority vehicles, VIP Vehicle Movement, Ambulance Movement, and Special Material Transport. This prototype addresses many security concerns with autonomous vehicles, in various applications, by using blockchain tools; showcasing blockchain's potential applications in applications that need to be secure.

Blockchain-centric Resource Management System [77] is a disaster response and relief system that utilizes blockchain based web/mobile applications. The goal of this application is to reduce the time for goods to be transported to areas affected by disasters. This would allow for those involved in the disasters to be provided with relief much quicker than typical. This system also allows for support for multiple blockchain frameworks, which increases the versatility it can be used. This system showcases the utility and flexibility of blockchain and how it can be applied to systems to increase throughput of objectives.

These applications show the ability of new applications, although not specifically blockchain, to have a positive effect on the healthcare transportation sector. Creating new possibilities for other applications to continue to improve the field.

2.6.2 Blockchain Based Auction Platforms

There are existing auction platforms that utilize blockchain technologies. These applications showcase examples of fully functional blockchain based auction applications for selling a product.

OpenSea [78] is a very popular digital marketplace that utilizes blockchain auction platforms. OpenSea utilizes two different types of auctions: Ascending Auctions, and Descending Auctions. If an auction finishes at or above one ETH OpenSea pays the gas fees for processing, but if the auction finishes below one ETH the seller is not obligated to complete the transaction but is given the chance to accept. OpenSea [78] also allows for sellers to accept an offer at anytime during the auction. Allowing for users to accept auction bids at any time would create shorter auction times and better support the seller.

Rarible [79] is another blockchain based auction system that utilizes timed auctions and open auctions. A timed auction has a set time period where users can bid on an item. Rarible [79] offers selling NFTs with a fixed price as well as using the auction system they have implemented. This application gives users different options for selling their product, a feature that may be beneficial for this thesis project.

Both of these examples of blockchain based auctions showcase great examples of how blockchain technologies can be utilized for auctions; however, both of these applications are applied to NFT trading, which limits their usability in different applications. However, they provide positive aspects of blockchain auctions and have implemented features that could be desirable for this thesis project.

2.6.3 Blockchain Based Applications in Healthcare

Blockchain has many applications in healthcare that have proposed since blockchain began gaining popularity. These applications showcase the benefits of utilizing blockchain technologies in the healthcare field.

A blockchain based record management system is a prime example of an emerging blockchain technology. This application aims to securely store patient information so that it can be quickly accessed to reduce the number of pre-hospital deaths. This system uses blockchain technologies along with secure file transfer to accomplish this goal. Create a continuous footprint of patient data throughout the medical process [80]. Public trust is an important factor to consider when handling medical information. This prototype showcases a blockchain technology that creates secure data and maintains public trust.

Blockchain-based Secure Ambulance-to-Everything Communications [81] aims to create trust and security in a system using blockchain technologies. It consists of an ambulance response to an emergency rescue operation, using blockchain to securely differentiate between an ordinary vehicle and an emergency vehicle. This differentiation between vehicles will allow for safer forged communications with traffic lights and will secure broadcast communications, creating safer transportation during emergencies [81]. This application provides good evidence how blockchain can be applied to emergency response scenarios to create safer and more secure outcomes.

DITrust Chain [82] attempts to utilize blockchain and the Internet of Things (IoT) to establish a trustworthy communication. In this application, the smart contract guarantees authentication of budgets, reducing semantic gaps while still enhancing trust. This is applied to the healthcare systems to create secure communication by applying innovative technology. This technology, overall, will improve the healthcare system by using emerging technology to improve outdated systems. This application showcases how blockchain technology can be applied to other systems to create secure communications and records.

Each of these applications showcase different methods of using blockchain technologies to improve healthcare. Each of these applications has to deal with public trust in order to, create a better environment for patients, and have high security of medical data to be considered viable solutions.

Chapter 3

Preliminary Survey and Analysis

In this chapter we describe the motivation behind SCeFSTA, its key requirements, and suitable features. In the current emergency response system, there is waste and lack of competition between emergency transportation companies. The current Emergency Response System has several flawed characteristics that could be remedied with new technology.

3.1 Research Methodology

The research methodology followed in this thesis takes the form of Figure 3.1. This includes two separate iterations, each having multiple steps.

First Design Cycle Iteration

The first design cycle began by consulting domain exports in the healthcare transport sector. We conducted semi-structured interviews with key stakeholders, consisting of paramedics and EMS coordinators from both rural and metropolitan backgrounds. Table 3.1 summarizes the stakeholders and their roles. The primary focus of these interviews were to understand the application area and who might benefit from an improved healthcare transportation system. The interviews were recorded and transcribed using Zoom video conferencing software.

The responses we received helped in three key ways: 1) to refine our research problem and question; 2) to validate our initial understanding of the healthcare transportation sector and its issues; and 3) to inform our research approach within the design science methodology.

The second step of the first iteration in Figure 3.1 was to formulate the objectives, requirements,

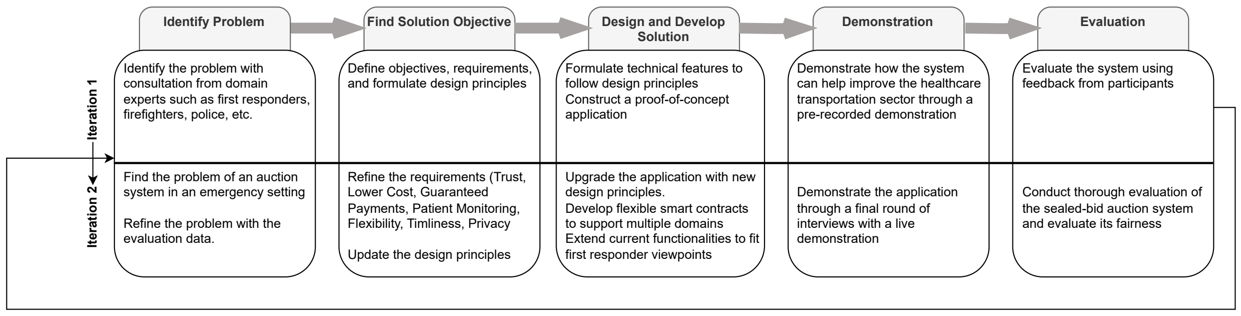

Identify Problem
Find Solution Objective
Design and Develop Solution
Demonstration
Evaluation
Iteration 1
Iteration 2
Identify the problem with consultation from domain experts such as first responders, firefighters, police, etc.
Define objectives, requirements, and formulate design principles
Formulate technical features to follow design principles
Construct a proof-of-concept application
Demonstrate how the system can help improve the healthcare transportation sector through a pre-recorded demonstration
Evaluate the system using feedback from participants
Find the problem of an auction system in an emergency setting
Refine the problem with the evaluation data.
Refine the requirements (Trust, Lower Cost, Guaranteed Payments, Patient Monitoring, Flexibility, Timliness, Privacy
Update the design principles
Upgrade the application with new design principles.
Develop flexible smart contracts to support multiple domains
Extend current functionalities to fit first responder viewpoints
Demonstrate the application through a final round of interviews with a live demonstration
Conduct thorough evaluation of the sealed-bid auction system and evaluate its fairness

Figure 3.1: Design cycle diagram for SCeFSTA

Table 3.1: Description of Interviewees

Interviewee #	Role	Locality type	Year's of practice
#1	Police Sergeant	Metropolitan	25 years
#2	Fire Chief	Rural	32 years
#3	Fire Captain	Rural	20 years
#4	Paramedic	Rural	19 years
#5	Paramedic	Rural	9 years
#6	Part-time EMT	Rural	3 years
#7	Director of Emergency Mgt.	Metropolitan	15 years
#8	Fire Chief	Metropolitan	35 years
#9	Assistant Chief	Metropolitan	41 years
#10	EMS Coordinator	Metropolitan	22 years
#11	System EMS Mgt.	Metropolitan	18 years

and design principles of the research. These design principles define what the platform needs to incorporate in order to be successful.

Utilizing the design principles drafted in step 2, the third step includes taking the design principles and constructing a proof of concept application.

The fourth step is demonstrating the proof-of-concept application to the stakeholders. This includes the same group of people mentioned in step 1; in order to validate the proof-of-concept, as well as receive feedback on how to make improvements. The last step of the first iteration is simply evaluation of the feedback from step 4. Once all the data is compiled, and all feedback is received, the design cycle shifts into the second iteration.

Second Design Cycle Iteration

The first step in the second iteration of this research was to identify why the current auction system was not a good fit for the platform. We used the evaluation from the last step to identify refinements that would need made.

The next step was to refine the requirements specified in step 2, iteration 1. From here, we got our final set of requirements, shown in Figure 3.1. After getting the new list of requirements, the design principles were also updated to reflect the new changes.

Using these new requirements, we updated the proof-of-concept. To do this, we developed flexible smart contracts that were able to handle transportation across several domains of the healthcare transport sector. In addition to the increased flexibility, we also extended other existing functionalities of the proof-of-concept to enhance reliability.

The next step of the design is a second demonstration. We have not been able to complete the demonstration at this point, but are looking to interview stakeholders. After completing the interviews, we will close out the design cycle by evaluating the data from the interviews.

Table 3.2: Potential application scenarios

Application Domain	**Description**	**Benefits over the legacy system**	**Shortcomings**
Emergency Response, not supported as an improvement to current emergency response	First responders or police call the ambulance dispatch centers to get an ambulance that will transport a patient to a medical facility	- Provides secure payments upon delivery of patients - Finds cheapest possible option for patient	- Increased dispatch time compared to existing systems - May not be viable/beneficial in rural area - Billing may be difficult due to Medicare and insurance
Tiered Emergency Response, supported by interviewee #3, #5, #9, and #10	First responders would use a hybrid implementation between the emergency dispatch system and our proposed prototype. Critical/unknown 911 calls will be dealt with by the nearest first responder, but less critical cases will be put up for auction using the prototype	- Provides secure payments upon delivery of patients - Allows government owned first responders to allocate time and resources toward high acuity patients - Gives private ambulances opportunity to respond to low acuity 911 calls	- Potential confusion in which system to use in a fast-paced environment - Delivery of critical patients will not have guaranteed payment for ambulances
Air-ambulance, supported by interviewee #3	The application would be applied to air-ambulance companies for patient transport and med-flights	- Produces fair competition in the air-ambulance market - Secure payments for transportation	- Small market due to limited competitors
Inter-facility Patient Transfer, supported by interviewee #2, #3, #5, #7, #9, and #10	This application would be applied to transporting patients between hospitals/facilities for different care	- Good for hospitals who do not have internal transportation - Provides larger market for private ambulances	- Small scope for the project - Not applicable to hospitals with internal transportation services
Non-Emergency Medical Transport (NEMT), supported by interviewee #1, #2, #3, #7, and #10	This domain would not be used in the case of emergency response but rather for private transportation (i.e. nursing home transportation). Competition would be between private ambulance companies	- Provides competition between private ambulance companies - Secure and guaranteed payments upon patient delivery	- Does not improve the emergency response system - Some private ambulance companies may not want competition
Tragedy Cleanup, supported by interviewee #3, #7, and #11	This application could be altered to cater to tragedy cleanup, this would allow the government to put out auctions for needed contractors to bid on to perform tragedy cleanup	- This would allow for getting the cheapest possible contractors - Contractors would know how much they will be paid ahead of time	- Costs needed to clean up tragedies would be difficult to calculate before cleanup - This would require many alterations to the intended implementation of the application

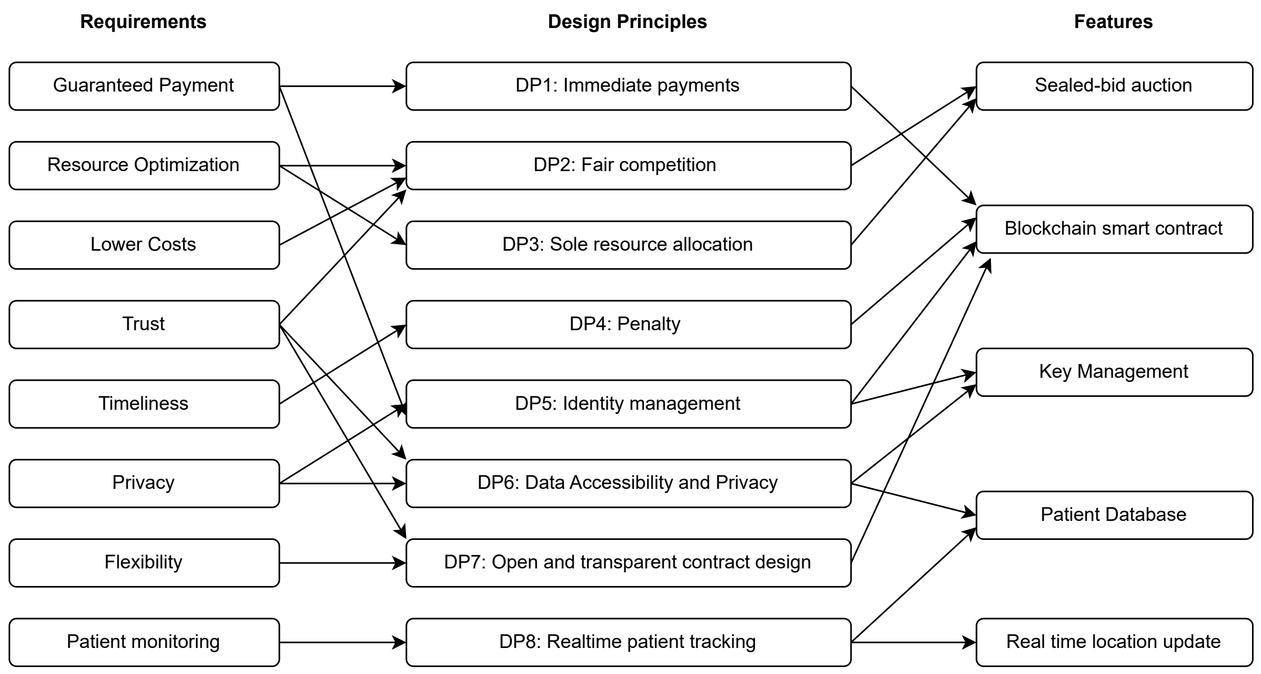


Figure 3.2: Design principle diagram for SCeFSTA

3.1.1 Defining the Application Domain

Table 3.2 summarizes the possible application scenarios that the interviewees suggest. The tiered emergency response domain was recommended by Interviewee #5. They stated that using a different dispatch system for lower acuity patients would free up resources for local EMS. The Air-Ambulance domain was recommended by Interviewee #3, they also denoted air ambulance companies MedFlight, air care, Evac, lifeline, and noted there are many more than this. This opens a new domain beyond ambulance transport with a much higher billing domain. Each of the proposed domains were derived from independent research or ideas from interviewees to consider possible applications of the prototype.

3.1.2 Research Question

How can we design a healthcare transportation system that is trustworthy, reliable, safe, flexible, and cost-effective?

Many design principles were evaluated for the motivation of SCeFSTA. Figure 3.2 Showcases the design principles, requirements, and the features required in this research. These design principles were derived based off of feedback from our interviewees, outlined in Table 3.1.

3.1.3 Requirements

The requirements showcased in Figure 3.2 are outlined below. These requirements were derived from domain knowledge and interviews with potential stakeholders.

Guaranteed Payment

Guaranteed payment in the healthcare transportation sector means that medical transports are paid for one hundred percent of their successful transports. In the current healthcare transportation sector, it is not uncommon to not receive payment for a significant number of transports. Several interviewees mentioned how payment can take months to be paid, and often much of the actual costs are not paid. Interviewee #10 talks about the number of EMS calls their fire station took for the year and billing rates:

> *“ We have billed out this year $718,280... but we don’t collect that, we have a collection rate of approximately 44 to 45%. ”*

Public healthcare transportation companies are not typically paid their full amount for transportation services, they are typically paid less than 50 percent of what they charge. Another interviewee speaks about funding as a major issue for EMS services. Interviewee #9 mentioned:

> *“ I think funding is the number one thing that’s an issue... it’s very difficult. These things (funding for a fire/EMS station) run on razor-thin margins, so the room for error is small... and getting reimbursement is one of the larges challenges. ”*

Running a medical transportation is not any easy task, and this task is made much more difficult by a lack of payment of services.

Resource Optimization

The resource optimization requirement referenced in Figure 3.2 represents a reduction of waste of time and resources. This mainly stems from many medical transports being dispatched to a single incident. Optimizing resources is a big challenge facing public emergency response systems. Interviewee #5 states,

> *“ There is a lot of waste in EMS, being able to clear up some of the low acuity cases for minor injuries, breaks, sprains, strains, would free up time and resources for high acuity cases ”*

When the interviewee mentioned acuity, they are explaining the degree of severity of the injury and the need for an efficient emergency response. Interviewee #5 also mentions that any call made to 911 for medical response must have an ambulance dispatched, regardless of severity of injuries. Having the option to outsource low severity or low acuity cases to private ambulances, local first responders can better cater to those who actually need emergency response.

Lower Costs

High costs are a prevalent issue in the healthcare transportation sector. Lowering the cost of transportation would allow more people to utilize healthcare transportation.

Trust

The trust requirement in Figure 3.2 includes the reliability of SCeFSTA, as well as the reliability of, the users/stakeholders of SCeFSTA. When dealing with medical information and records, data security/ privacy is a must to maintain public trust.

Timeliness

Timeliness refers to the platform being able to operate quickly and efficiently. For instance, the auction system limits the amount of wasted and lost time, and the user interface provides seamless navigation. Thus, timeliness is an integral part of the proposed design.

Privacy

Given the medical application of this research, security of data, privacy for the patients, and trust in the application is paramount. The Health Insurance Portability and Accountability Act (HIPAA) [83] requires that patient data is kept secure and not released to any unauthorized users.

Flexibility

Being flexible and adapting to make improvements is important to the first responders we interviewed. They were asked the following question: "Do you find it viable to change the current EMS system?"

Many of the interviewees thought it could be difficult to make changes. However, the EMS system has been altered countless times for better optimization. Interviewee #11 stated:

> *" I think that we should always be looking for a way to better ourselves (improving EMS services)... however, there may be some challenges in doing that with a lot of different groups (different regions and states). "*

Patient Monitoring

The ability of medical facilities and the delivery verifiers to monitor patient status is important in maximizing patient safety.

3.1.4 Design Principles

The design principles for this research were directly derived based on the requirements outlined above and in Figure 3.2.

Design Principle 1 - Immediate Payments

Given that guaranteed payments is a requirement of SCeFSTA, immediate payments was made a design principle. This principle is regarding paying medical transportation companies in full, as soon as they deliver the patient.

Design Principle 2 - Fair Competition

The second design principle in Figure 3.2 came from the following interview question: Would you support competition between healthcare transportation companies? This interview question was to gauge how interviewees felt about creating competition between healthcare transportation companies through an auction system. The majority of the interviewees were in favor of created competition. Interviewee #2 stated he was in favor of competition because it may solve staffing issues:

> *" Staffing is our biggest issue right now, getting qualified applicants is difficult. I don't think they (private ambulance companies) have the workers either. "*

It is important to note, however, many of the interviewees were skeptical of competition in emergency scenarios due to added time taken to provide vital services to the patients. This repeated note by the interviewees lead to design principle #2 mentioned in Figure 3.2. Fair market competition was derived as a principle from the requirement of trust and is supported by the sealed-bid auction feature. The following question was presented to the interviewees.
Would the use of sealed-bid auctions be fair for all transport companies involved?

This interview question had a wide variety of answers. Some interviewees were more in favor of open bid auctions. Interviewee #2 stated:

> *" I would think it would be better to do open bid auctions, so people can see how low they have to be to stay competitive and can make as much money as possible, if not you may put ambulance services out of business due to not making enough money. "*

However, interviewee #3 believed the type of auction would not have a large impact on the fairness of outcome:

> *" I don't think sealed-bid or open-bid auctions will have a large impact either way. "*

Another interviewee feared the sealed bid auction would produce overbidding by healthcare transportation companies. Interviewee #4 claimed:

> *" With it being the lowest bid, it will incentivize companies to bid on a lot of tenders, so the people on the units may get run ragged from having too many jobs. This may cause worker burnout for the drivers. "*

The fair auction mechanism comes from the privacy and trust requirements, and is supported by the sealed-bid auction feature proposed.

Design Principle 3 - Sole resource allocation

Sole resource allocation in Figure 3.2 references only one transport being dispatched for each patient, helping satisfy the resource optimization requirement, which is delivered through the sealed-bid auctions.

Design Principle 4 - Late penalties

The penalty design principle refers to penalizing transport companies for delivering a patient late, or for not delivering the patient at all. This design principle was designed to increase trust.

Design Principle 5 - Identity Management

Many different users require the use of an identity management system. The identity management system will help keep track of all users of SCeFSTA, ensuring user information is kept secure, and confidentiality of patient data is kept. Identity management is supported by the blockchain smart contract feature.

Design Principle 6 - Data Accessibility and Privacy

Given HIPAA [83] and patient privacy as a requirement, data accessibility and privacy was made a design principle of this platform. This is supported by the patient database to ensure patient data is always accessible to authorized users, but private from those who aren't.

Design Principle 7 - Open and Transparent contract design

An open and transparent contract design, outlined as a design principle in Figure 3.2, helps create trust in the system. This comes from showcasing exactly how the smart contract operates, especially when funding is involved. This also supports flexibility of the platform, because the smart contract is able to be adapted and refined to several fields as stakeholders learn the ins and outs of the system.

Design Principle 8 - Real-time patient tracking

The last design principle drafted was real-time patient tracking. Real-time patient tracking includes live updates on a patient's whereabouts, in order to support the patient monitoring requirement.

3.1.5 Features / Tools

In the proposed system, several of the features are designed to satisfy the design principles mentioned in Figure 3.2.

Sealed-Bid Auctions

A sealed-bid auction allows for fair competition and sole resource allocation. Competition in the economy's markets is fundamental in generating a healthy economy. Markets with firms competing for customers fosters lower prices and better products for consumers [66]. Markets that do not have strong competition are going to result in higher prices for inferior goods than if strong market competition was present. A lack of market competition results in a single entity controlling the majority of a market. Allowing them to keep other competitors from entering the market, creating an unequal playing field [66]. Auctions are a commonly used method for creating competition when trying to sell a good or service. This section will outline many types of auctions as well as their core functionalities. Utilizing blockchain based auctions can lead to a lack of fairness within an auction. Without the proper controls, it is possible to design a smart contract that does not behave as intended, creating a lack of fairness for users. When creating competition in a new market, it is paramount to consider fairness of competition to ensure a single market does not control the market, and that the applications being used to create competition behaves as intended.

Real-time location update

Real-time location updates allow for accurate patient tracking, which will help users track patients easily. This is accomplished through a separate database that tracks each patient's current location and is updated regularly. It helps the users of the system keep track of where patients are at and when they can expect them to arrive.

Patient Database

The patient update database provides data accessibility and privacy principles while simultaneously supporting real-time patient tracking. Security in Distributed Ledger Technology can be an issue for different applications utilizing smart contracts. Given the domain of this research, it is important to consider the security and integrity of patient information under the Health Insurance Portability and Accountability Act (HIPAA) [83]. Due to this, we require a private storage system to keep patient data secure.

Key management

Key management will help support the identity management of users, as well as support data accessibility and privacy. Smart contracts utilize unique wallet addresses, given this we are able to use this to identify users and keep user data secure.

Blockchain Smart Contract

Finally, the blockchain smart contract is the center of most of the design principles. It enables the penalty system, controls identity management, and allows for an open and transparent smart contract design. The use of blockchain technologies brings up many security questions. Due to the decentralized nature of blockchain, a basic use of blockchain technologies would leave patient data exposed. To remedy this, many frameworks have been developed to reap the benefits of blockchain technologies while maintaining privacy of data [84]. These frameworks make blockchain a great tool to host applications while still keeping users data private and secure.

Chapter 4

Design and Implementation

This chapter provides detail on the design and the implementation of Smart Contract enabled Fair, Secure, and Transparent Auction (SCeFSTA).

4.1 SCeFSTA Design

SCeFSTA creates a platform for increasing competition between healthcare transportation companies. This should create fairness and balance in the market for emergency response, inter-facility, and non-emergency medical transport (NEMT).

Figure 4.1 outlines the design of SCeFSTA. In Figure 4.1, there is a database and blockchain smart contract working together as the back-end of the application. The front-end is a NextJS application hosted on a website. SCeFSTA can be accessed through both desktop and mobile browsers, and the interface can be used by any of the four users of the system. The four users showcased in Figure 4.1 are Administrators, Transport Hirers, Transport Providers, and Delivery Verifiers.

4.1.1 Participants

There are many roles in SCeFSTA:

- **Transport Hirer:** this user initiates auctions for the providers to bid on. The hirers control tender creation as well as retracting and reclaiming tenders. More information on this is given in Table 5.3.

- **Transport Provider:** this user is responsible for bidding in auctions and delivery of patients to the verifier for auctions that are won.

- **Delivery Verifier:** the verifier will ensure the patient was delivered on time and safely. E.g. Hospitals could verify the patient delivery.

- **Administrator:** an administrator oversees the users' accounts and has the ability to add and remove accounts.

- **Patient:** the patient is the person in need who is being delivered to the verifier.

These participants have certain smart contract API functions they are able to access. Detailed information about these calls and which users can call them are in Table 5.1 and Table 5.1.

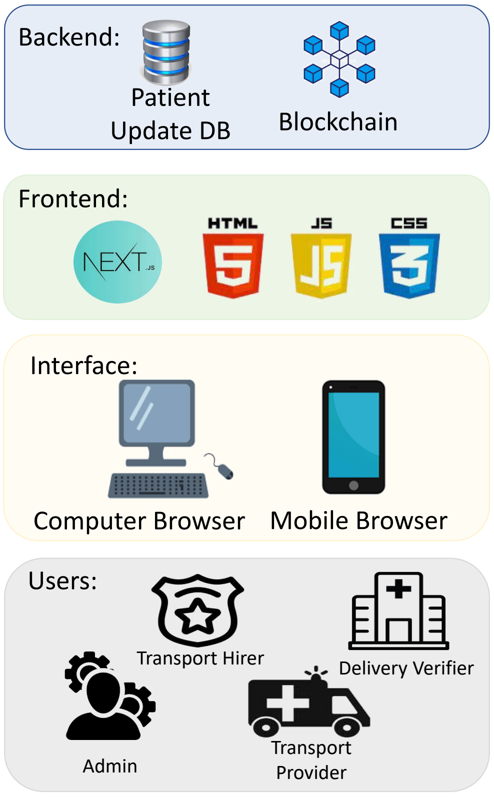


Figure 4.1: SCeFSTA architecture design diagram

4.1.2 Blockchain

Blockchain networks provide technical infrastructures for processing transactions with the blockchain. Blockchain networks also allow for easy access to the blockchain ledger and the smart contract, creating better connectivity for applications [42]. There are multiple types of blockchain networks and many different networks for each type with differing functionality.

There are multiple factors to consider before selecting which network to implement into a project. Speed, cost of transactions, security, and user base are important factors of blockchain networks to consider. Four highly regarded networks are, Ethereum [43], Avalanche [46], Ethereum Layer2 Polygon [44], and Fantom [85]. Each of these options displayed strong connections to each of the valued factors. Ethereum is an expensive network with slow transaction throughput when compared to other options. However, Ethereum [43] has a very large user base and a lot of support from the community and also has its own currency, ETH. To remedy some of these issues, Ethereum released separate blockchains that extend Ethereum called Ethereum Layer2. Layer2 options aim to reduce costs, increase throughput, all while maintaining Ethereum tools. Polygon is an example of a Layer2 extension that has much lower gas price costs and higher throughput, but does have less supporting documentation compared to other networks [44]. Fantom is one of the fastest networks available, but it too is missing strong documentation and a large user-base

Table 4.1: Ethereum blockchain network vs. Avalanche blockchain network [6] [7]

Feature	Ethereum	Avalanche
Scalability	Medium	High
Ecosystem	Well established	Rapidly growing
Fees	High fees and susceptible to network congestion	Lower fees and more stable
Transaction Costs	13 Transactions Per Second (TPS)	4,500 TPS
Transaction Finality	6 minutes	<2 seconds

[45]. Avalanche [46] is an increasingly popular competitor to Ethereum. It is growing rapidly and is stated to be the leading blockchain for the financial services in the web 3.0 economy [45]. Avalanche also has high transaction throughput with low transaction costs when compared to other networks. Avalanche also has its own currency, AVAX, which provides a different option from the highly regarded Ethereum [46]. There is no shortage of blockchain networks to choose from, and each network offers benefits and trade-offs for creating a blockchain that is tailored to a specific purpose.

Transaction Throughput: Avalanche uses a three-blockchain structure to support strong transaction throughput. A quick transaction throughput allows users to wait less and have an overall better user experience. It also reduces the time patients have to wait, which could be drastic in a critical patient setting. Avalanche's average throughput is around $4,500$ transactions per second (TPS) while Ethereum's TPS is around 13 transactions per second, showcased in Table 4.1. Avalanche also has a significantly lower transactional finality than Ethereum, making it a better overall option for this application [6]. This throughput time of Avalanche is very good compared to other networks like Ethereum.

4.1.3 Cryptocurrency

Transparency of transactions is a pressing issue when it comes to blockchain transactions. Cryptocurrencies can be confusing and extremely volatile. One solution to this issue is the use of a Stablecoin. Stablecoins aim to create significantly more predictable costs than other cryptocurrencies. This is done by pegging the value to another currency, commodity, or financial instrument [40]. One example of a Stablecoin is Dai [57], Dai is a currency that runs on Ethereum (ETH) and is a decentralized Stablecoin. This Stablecoin attempts to keep the price of the currency constant, aiming to maintain a cost of $1.00 USD [57]. This provides a solution to the issue of highly volatile of cryptocurrencies and creates a stable currency for blockchains to take advantage of.

4.1.4 Sealed Bid Auction

SCeFSTA utilizes sealed-bid auctions to keep individual bids from the transport companies. The goal of this is to promote fairness between transport companies during the auction. Auctions are initiated by the tender poster, who sets a bidding period length and maximum tender amount for the auction between nearby transportation companies. Once the auction has begun, each company has a limited period of time to submit a single bid with a penalty amount. Once the auction is over, the transportation company can reveal their bid to see if they won the auction. After winning the auction, they proceed with the transport of the patient. A brief description of each of the smart contract API calls as well as who is able to call the API are listed in Table 5.3.

The auction process and overall throughput of SCeFSTA are outlined in the four stages below.

Tender initiation

Tenders are posted by tender issuers. These tender issuers could be first responders, patients, nursing homes, etc. When a tender is posted, it is given the location of the patient, a time limit for the auction, allowed hospitals for transportation, and a penalty amount. Once a tender is posted, there is an auction period for the specified auction time period. Tender initiation is showcased in Figure 4.2 in the beginning of the Secret Bidding Period.

Bidding and winner selection

Once a tender is posted, the auction period begins. During this period, ambulances have the ability to put in one sealed-bid, a bid that is not accessible to other participating competitors. After the auction period, all the ambulances will reveal their bids and the lowest bid that is revealed wins the sealed-bid. Once the reveal period is over, transports can check the tender to see if they were the winner. If so, they can proceed to pick up the patient. Bidding and winner selection is showcased in Figure 4.2 at the end of the Secret Bidding Period.

Algorithm for Auction Mechanism

SCeFSTA has a relatively simple auction mechanism. The algorithm for the auction system is outlined in Algorithm 1. This showcases the entire auction process from tender initiation to payment to the transport for the successful delivery of the patient. This algorithm reinforces the stages of the auction system outlined.

Patient transport and verification

After winning the sealed-bid auction, the winning ambulance will pick up the patient from the specified location. After receiving the patient, the ambulance will proceed to the hospital/designated drop-off area with the patient. Upon delivery, the patient will be accepted and funds will be transported to the ambulances crypto-wallet immediately. Patient transport and verification begins after Patient Delivery Period begins and ends during the Payment Period outlined in Figure 4.2.

Penalty

Each tender is assigned a penalty amount when posted by a tender issuer. This amount is paid to the tender issuer and will be returned if the tender is not won by the ambulance/transport. It is also returned if the ambulance/transport wins the bid and provides an on time delivery. This penalty amount ensures companies are not bidding on tenders they are not able to take, or else they will lose the penalty amount.

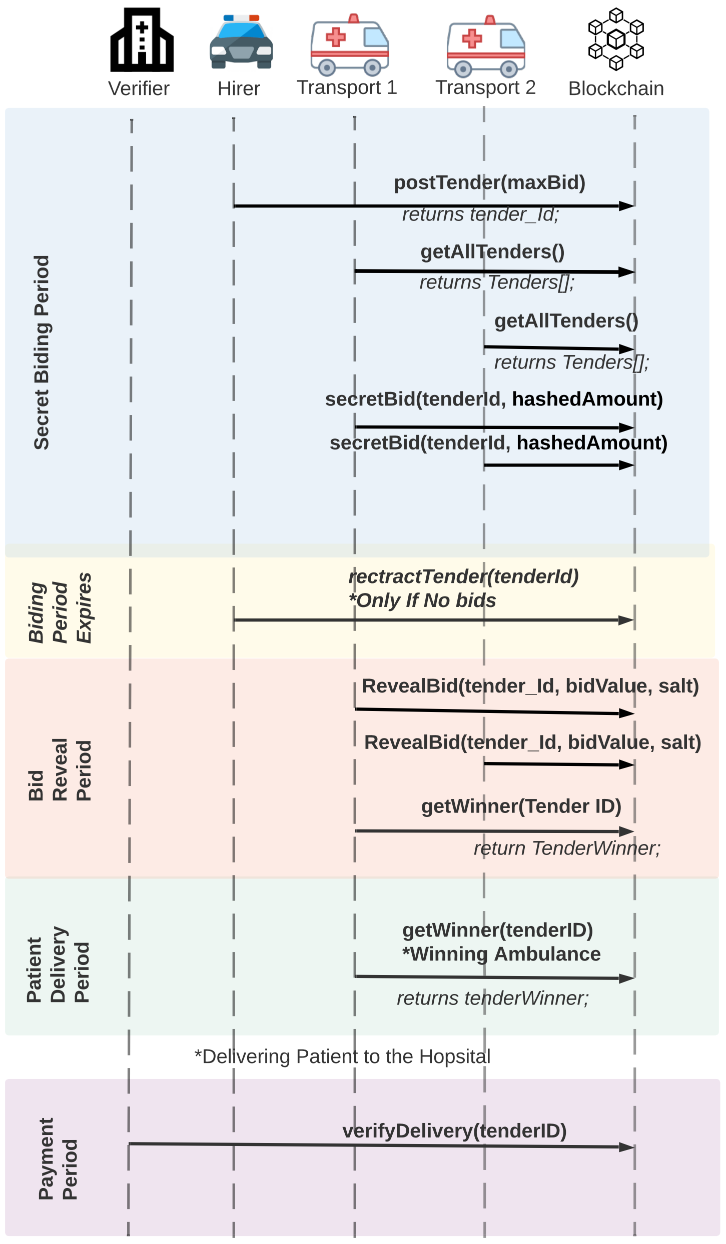


Figure 4.2: Sequence diagram

Algorithm 1: Algorithm for SCeFSTA Auction Mechanism

```
1  Tender Initiator $u_a$ posts a tender $t$ creating an auction $\gamma$ with a due date $t_d$, a penalty $t_p$, a
     max bid $p_b^{max}$, and a final bid $p_b^{final}$
2  Bidder $u_i$ places a bid $p_b^i$ with penalty amount $p_p$
3  if $\gamma$ is open then
4      if $t_d$ has not passed & $u_i$ has not placed a bid & $u_i$ is registered as an ambulance & $p_p$
         matches $t_p$ then
5          if $p_b^i < p_b^{final}$ then
6              $p_b^i \leftarrow p_b^{final}$
7              Lowest Bidder $u_lowestBidder \leftarrow u_i$
8          else
9              Discard $p_b^i$
10     else
11         Close $\gamma$
12 else if $\gamma$ is closed & $t_d$ has passed then
13     Lowest bid amount $p_b^{min} \leftarrow t$
14     Tender Winner pays penalty $t_p$, to be returned on successful transportation of patient
```

If an ambulance/transport does not deliver the patient on time, there will be a penalty amount removed from their payment. This aims to reduce late deliveries and hold ambulances/transports accountable for deliveries they accept. This should reduce the amount of overbidding for fear of losing money on transactions. The penalty amount is specified in the postTender function in Figure 4.2, and a transport must pay this penalty amount if they fail to deliver a patient.

4.2 Prototype Implementation and Demonstration

This section describes the components that were used in implementing SCeFSTA.

4.2.1 User Interface (UI)

The prototype has five main pages in the UI:

Login Page

this page is the landing page for the application that directs users to login through MetaMask [65]. This is shown in Figure 4.3a. If a user does not already have MetaMask account installed on their browser, it will alert them and guide them through installing it before allowing them to log in. If MetaMask is installed the user is able to connect an account through their application, then they are allowed to log in, being routed to their default dashboard.

Admin dashboard

this page is only accessible by administrators and showcases all user account information. In the admin dashboard, administrators are able to add and delete any type of users, except for other admins. Each of these functions are showcased in Table 5.1 under the account smart contract. Only the creator of the smart contract is able to add and delete admin users. The administrators are also able to view any information associated with user accounts such as their walletID, address, name, etc.

Initiator dashboard

this dashboard is accessible to initiator/transport hirers or inter-facility accounts, and showcases all the tenders currently established, and also has a tender form, shown in Figure 4.3c. From this page, authorized users can create tenders, reclaim tenders, or retract tenders. More information about these functions are referenced in Table 5.3. Within the tender form, a hirer selects the allowed delivery verifiers, then has an option whether the tender is for an appointment, or an emergency scenario. If an appointment scenario, the hirer selects a date and time for the appointment, as well as the payment amount. If an emergency scenario, the hirer gives information on the patient's status and injuries, and then sets a delivery time limit for how long transports have to pick up and delivery the patient.

Transport dashboard

transportation providers have access to this page which allows them to view and bid on open tenders, reveal placed bids, and check their status on bids that have been revealed. This page can be seen in Figure 4.3d.

Facility dashboard

the facility dashboard is accessible by facility and inter-facility accounts and displays all pending, incoming, and accepted patients for that facility, shown in Figure 4.3e. From this page, verifiers are able to verify when a patient is delivered to an allowed facility.

Patient update map

this page showcases the patients who are currently in transport to a delivery verifier, showcased in Figure 4.3f. This update system helps users visualize where all of the users and patients are currently located.

The user interface of SCeFSTA is designed to be a simple and efficient as possible. The screenshots in Figures 4.3a-4.3e showcase some of the main functionality of the system.

In order to create an easy and efficient user interface, the front-end was developed using the Next.js [86] framework and React.js libraries. Next.js is a very well-supported framework which promises longevity of the prototype. Next.js also provides many components and tools that aid in creating a positive and intuitive user experience. The prototype itself was hosted on an EC2 instance through Amazon [87]. This provides a secure and reliable host for the prototype to eliminate any downtime that may come from a less reliable source. Having continuous connectivity is of the upmost importance when working in emergency situations, so a reliable host is a necessity for this prototype.

Security - SSL

OpenSSl communication is paired with blockchain transactions to create secure communications between all components of the prototype. OpenSSl is a widely known open library used for secure communications [88]. Security is one of the most important factors in public trust and must be considered when working with confidential data.

Mobile App

Through the MetaMask mobile application [65], users of SCeFSTA can confirm transactions from our platform. This allows for increased portability, allowing users to easily access transactions.

4.2.2 Blockchain Network in Use

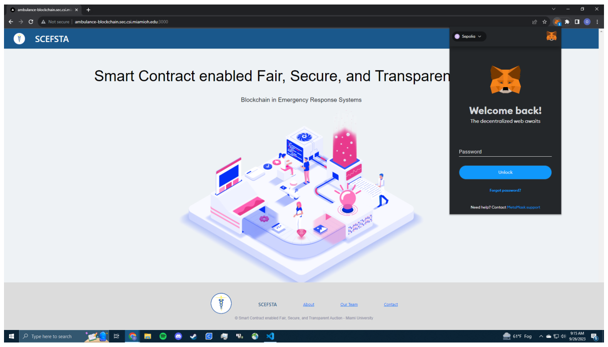
The use of Avalanche provides the necessary transaction throughput, transaction security, and immutability of data necessary for a prototype in the healthcare domain.

The smart contracts for this prototype were deployed on Avalanche Fuji C-chain [46], a test chain for Avalanche applications. The smart contracts were built using Solidity programming language [89]. Smart contracts are applications that control the functionality of accounts within the Ethereum state. Solidity is a high-level object-oriented programming language, and it was designed to target the Ethereum Virtual Machine (EVM) [89].

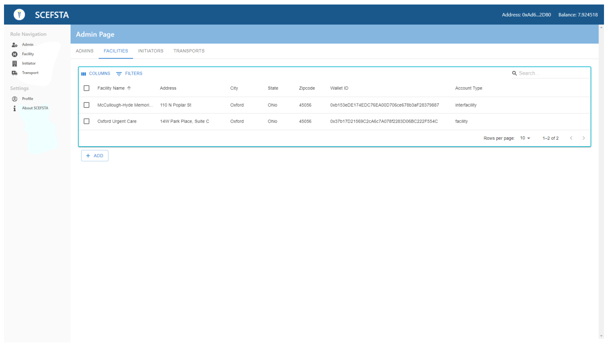
4.2.3 Crypto Wallet - Metamask

Cryptocurrency is typically held in cryptocurrency software wallets, otherwise known as crypto-wallets. This allows users to securely store their cryptocurrencies and tokens in a single location that is readily accessible. There are two types of crypto-wallets, *Hot wallets* are crypto-wallets that are connected to the internet and readily accessible. *Cold wallets* are not connected to the internet and are less convenient to access but more secure [63]. This project requires the use of hot wallets, so transactions can be made instantly and payments can be confirmed upon deliver of a patient. Crypto-wallets can be utilized to perform many transactions including, buying, trading, lending, or earning cryptocurrency.

Given the secure nature of this research, selecting a strong cryptowallet for users to use is paramount. The transactions completed must be secure, fast, and easily-accessible. The crypto-wallet selected for this project was MetaMask [65]. MetaMask allows for interactions and transactions between the blockchain. It is a simple google extension which allows for ease of use for checking and sending cryptocurrencies in a user's wallet. MetaMask also offers a mobile option,


SCEFSTA
Smart Contract enabled Fair, Secure, and Transparen
Blockchain in Emergency Response Systems
Welcome back!
The decentralized web awaits
Password
Unlock

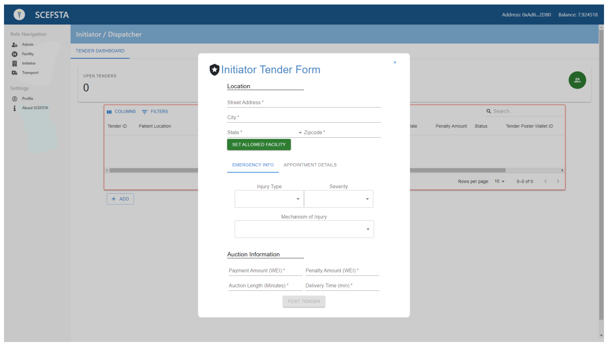
(a) Login dashboard, the first page accessed by users that provides MetaMask login



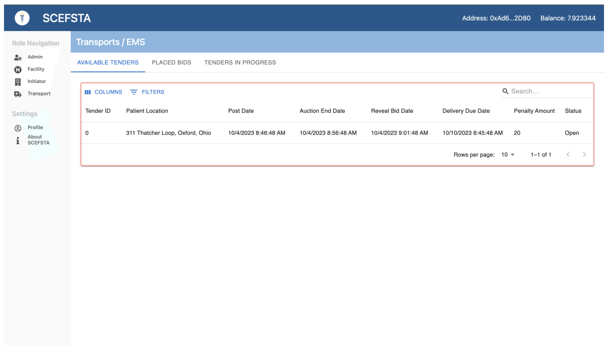
Facility Name	Address	City	State	Zipcode	Wallet ID	Account Type
McCullough-Hyde Memori...	110 N Poplar St	Oxford	Ohio	45056	0xb153eDE174EDC76EA00D706ce678b3aF28379887	interfacility
Oxford Urgent Care	14W Park Place, Suite C	Oxford	Ohio	45056	0x37b17D21569C2cA6c7A078f2283D06BC222F554C	facility

Rows per page: 10 1–2 of 2

(b) Admin dashboard, allows administrators to see all users. As well as adjust their permissions



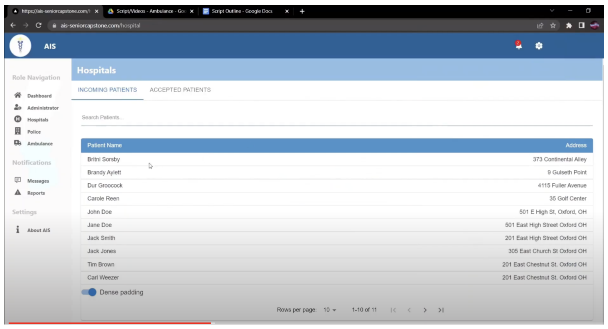
(c) Tender form, provides a physical interface for the postTender API call



Tender ID	Patient Location	Post Date	Auction End Date	Reveal Bid Date	Delivery Due Date	Penalty Amount	Status
0	311 Thatcher Loop, Oxford, Ohio	10/4/2023 8:46:48 AM	10/4/2023 8:56:48 AM	10/4/2023 9:01:48 AM	10/10/2023 8:45:48 AM	20	Open

Rows per page: 10 1–1 of 1

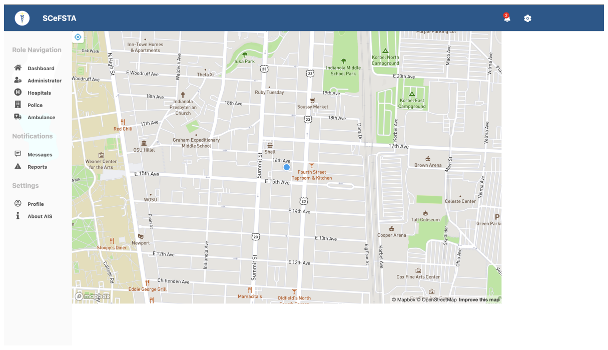
(d) Ambulance dashboard, allows transporters to see tenders under auction, their placed bids, as well as in-progress tenders



Patient Name	Address
Britni Sorsby	373 Continental Alley
Brandy Aylett	9 Gulseth Point
Dur Groocock	4115 Fuller Avenue
Carole Reen	35 Golf Center
John Doe	501 E High St, Oxford, OH
Jane Doe	501 East High Street Oxford OH
Jack Smith	201 East High Street Oxford OH
Jack Jones	305 East Church St Oxford OH
Tim Brown	201 East Chestnut St. Oxford OH
Carl Weezer	201 East Chestnut St. Oxford OH

Rows per page: 10 1-10 of 11

(e) Facility dashboard, allows delivery verifiers to see incoming patients, and already accepted patients



(f) Patient update map, provides real-time updates of patient locations

Figure 4.3: Screenshots of the SCeFSTA User Interface

which allows for easy access from any mobile device. The essential features provided by MetaMask are broadcasting transactions, send/receive cryptocurrencies and tokens, securely connect to Decentralized Applications (DApps), and store/manage multiple account keys in a single wallet.

4.2.4 Patient Update Database

A centralized database such as a SQL database is very secure, with the proper safety protocols, and allows for safely storing patient information to ensure there are not any HIPAA violations. A simple MySQL [90] server was used for the database. This database is used to support the blockchain and store non-critical, personal information off-chain. This reduces the amount of information stored in the blockchain, creating lower overall gas costs and waiting time for information to be retrieved. The database has two tables which hold all the user and patient information. The Users table holds the user's walletID as the primary key, the user's first and last name, email, location, username, and type of account (Admin, Initiator, Transport, or Hospital). The second table named Patients holds all the patient information necessary for the transporters to move the patient from one location to the end destination.

4.3 Technical Challenges for Implementation

We faced many issues during the design and development period of this project. Given that blockchain technologies are relatively new we faced issues with React modules becoming deprecated and being forced to switch to new modules. We also had to make a major adjustment and transfer our user/patient storage from MongoDB[91] to a MySQL[90] database. We also had to make several large changes in the scope of the project, following our initial interviews. Detailed information about the issues we faced are listed in this section.

We faced several obstacles throughout design and development of SCeFSTA that caused setbacks and required major application changes. In the end, these changes made our application easy to customize and made the project code easier to navigate.

4.3.1 User Storage

Early in the development of this thesis, we decided to change our user storage system from MongoDB to MySQL. This switch was because we thought MongoDB was not as portable and user-friendly as MySQL; however, switching this required starting the database from scratch. This also required creating a host for the new API and reworking the interfaces within the NextJS application.

4.3.2 Deprecated Blockchain Networks

Another major issue faced throughout development was the Ethereum testnet originally implemented became deprecated, and we were no longer able to use this. Originally, the testnet information was hard coded into a configuration file and individually into each React function that calls the smart contract API. With how the testnet and smart contracts were implemented, it was very difficult to change the testnet used, requiring refactoring each of the smart contract API callers by hand and extensive configuration. Instead of re-implementing everything by hand, we decided to refactor our configurations to allow to easily alter the blockchain network and currency utilized. The new implementation has a single configuration point that changes the implementation of each smart contract call and the interface with the smart contract. This allows owners of the smart contracts to decide which blockchain network they want to use for their system and streamlines any changes made with the blockchain network.

4.3.3 Deprecated React Modules

Wallet and transaction transparency was an important consideration in this project. Users need to be able to quickly see how much currency they have in order to ensure they have the funds required to post a tender or pay the initial penalty fee for a tender. To help keep wallet balances transparent, we needed to implement a feature that showed users how much cryptocurrency they had in their MetaMask wallet. The React module we originally implemented for MetaMask did not have a feature to get the wallet balance, so we were forced to either not implement the feature or refactor our wallet system in order to implement this feature. We ultimately decided to implement a new React module, which required refactoring our login system and ultimately provided better features than we had access to before.

4.3.4 Domain Flexibility

Another requirement we derived from our interviews was that our application needs to be flexible and should ultimately be able to support several domains of the healthcare transport sector. In order to implement this, we decided it was better to create two separate smart contracts for the application. One smart contract contained all the API functions for creating, deleting, and viewing user permissions. The other smart contract has any code pertinent for the sealed-bid auction system. Following this, we had to refactor how the users work so we could support several types of transport hirers, and support delivery verifiers that can also serve as transport hirers for inter-facility transportation. These changes also required reworking the tender form so that the new types of transport hirers only have access to certain types of tenders.

Chapter 5

Evaluation and Validation

This chapter provides evaluation on costs and API times of the application. In addition to this, the chapter also provides validation that the design requirements of the application are satisfied.

5.1 Evaluation

We evaluate the prototype in two different scenarios. In the first method, we used a blockchain tool-chain called Foundry[92] in order to test the gas prices of our smart contract. In the second method of evaluation, we evaluate the solidity contract using the Truffle test suit in order to get the API latency of API calls from JavaScript to the blockchain.

In addition to estimating the gas price of the smart contracts, we also utilized Foundry for unit testing. These unit tests ensured correct functionality for the program API calls throughout the development cycle. We evaluated the system using two methods: 1) estimating the gas price for each API call and 2) estimating API latency, as described in this section.

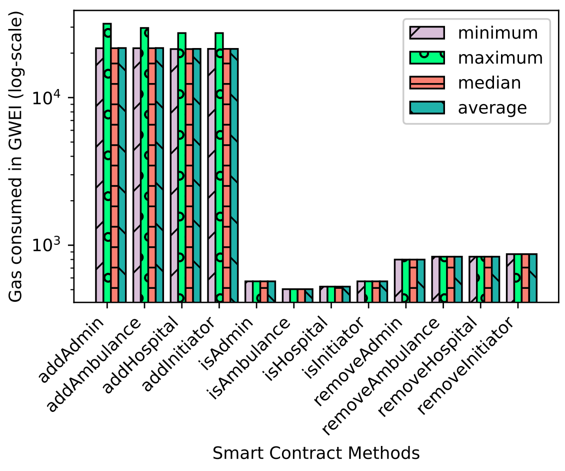
5.1.1 Estimating Gas Prices Using Foundry Tool-chain

Foundry Tool-chain [92] was also used to generate estimated gas prices for the prototype API calls. This allowed for tweaking/optimizing contract API calls for optimization.

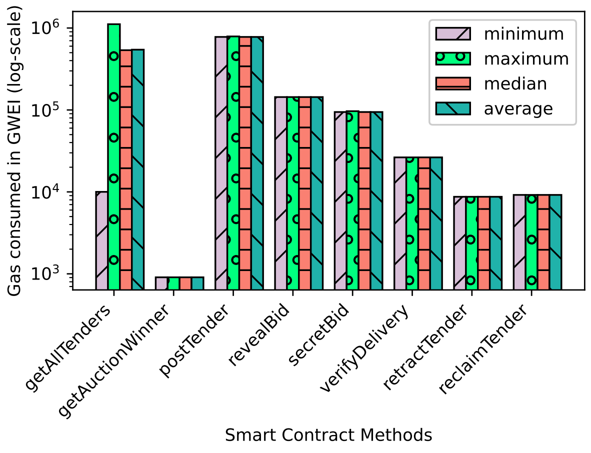
Observing gas price for different API calls with static load

Gas prices for each smart contract API call were tracked in the SCeFSTA prototype. Figure 5.1a showcases the minimum, maximum, median, and average projected gas prices (log-scale) of the Account smart contract with 500 calls made to each API. Figure 5.1b showcases the minimum, maximum, median, and average projected gas prices (log-scale) of the Auction smart contract with 100 calls made to each API.

From Figure 5.1a it is apparent there is not much variability between the minimum, maximum, median, and average for the Account smart contract functions. This low variability showcases that the program could be scaled up and down, but still have consistent gas prices among the API calls. There are similar results when viewing Figure 5.1b, except for the getAllTenders function, which seems to have some variability.



(a) Account management related API cost with static load



(b) Auction system related APIs cost with static load factor

Figure 5.1: Gas estimate (log-scale) for different API calls (500 each).

Table 5.1: Description of API smart contract functions in Account smart contract

API Name	**User Invoked By**	**Description**
addAdmin	superAdmin	Creates a user with administrator permissions for given crypto address
addAmbulance	Admin	Creates a user with transport provider permissions for given crypto address
addHospital	Admin	Creates a user with delivery verifier permissions for given crypto address
addInitiator	Admin	Creates a user with transport hirer permissions for given crypto address
isAdmin	Admin	Queries to see if given crypto address is an administrator
isAmbulance	Admin	Queries to see if given crypto address is a transport provider
isHospital	Admin	Queries to see if given crypto address is a delivery verifier
isInitiator	Admin	Queries to see if given crypto address is a transport hirer
removeAdmin	superAdmin	Removes users (Admin) access from given address (if applicable)
removeAmbulance	Admin	Removes users (Transport Provider) access from given address (if applicable)
removeHospital	Admin	Removes users (Delivery Verifier) access from given address (if applicable)
removeInitiator	Admin	Removes users (Transport Hirer) access from given address (if applicable)

Table 5.2: Average price in USD of SCeFSTA's smart contract API calls (average of 100 API calls) on 3 different blockchain networks – Ethereum, Polygon, and Avalanche. The prices for Ethereum, Polygon, and Avalanche, as of October 8th, 2023 are \$1636.92, \$0.56, and \$10.09 respectively [8].

API	GWEI	Ethereum(\$)	Polygon(\$)	Avalanche(\$)
addAdmin	21655	0.0035447503	0.0000012127	0.0000218499
addAmbulance	21647	0.0035434407	0.0000012122	0.0000218418
addHospital	21368	0.0034977707	0.0000011966	0.0000215603
addInitiator	21390	0.0035013719	0.0000011978	0.0000215825
getWinner	1110	0.0001816981	0.0000000622	0.0000011200
isAdmin	570	0.0000933044	0.0000000319	0.0000005751
isAmbulance	504	0.0000825008	0.0000000282	0.0000005085
isHospital	525	0.0000859383	0.0000000294	0.0000005297
isInitiator	569	0.0000931407	0.0000000319	0.0000005741
removeAdmin	800	0.0001309536	0.0000000448	0.0000008072
removeAmbulance	836	0.0001368465	0.0000000468	0.0000008435
removeHospital	836	0.0001368465	0.0000000468	0.0000008435
removeInitiator	871	0.0001425757	0.0000000488	0.0000008788
revealBid	143693	0.0235213946	0.0000080468	0.0001449862
postTender	778738	0.1274731807	0.0000436093	0.0007857466
secretBid	94674	0.0154973764	0.0000053017	0.0000955261
verifyDelivery	26485	0.0043353826	0.0000014832	0.0000267234
reclaimTender	9210	0.0015076033	0.0000005158	0.0000092929
retractTender	8718	0.0014270669	0.0000004882	0.0000087965

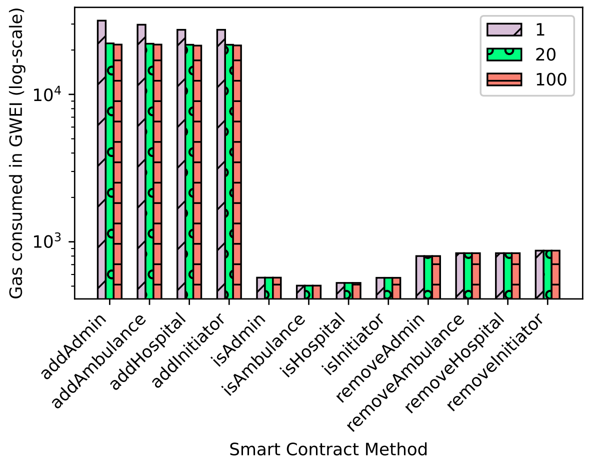
Scalability Effects on API Gas Usage with Varying Load

Figure 5.2a and Figure 5.2b showcases how gas prices (log-scale) are impacted when different load factors (number of calls) to different API functions that occur in the smart contracts. It is also apparent that the load factor of each API call is very consistent for all the load factors applied. Figure 5.2b for the account smart contract shows that all the functions are stable in cost. Figure 5.2b showcases the smart contract functions utilized in the auction process. The postTender API call is initiated by the tender initiator, which begins the auction period for a given tender. The secretBid API call is utilized by registered ambulances, in which they place bids unknown to other transport companies on an open auction. The low variance in secret bid allows for the SCeFSTA prototype to handle many bids from different transports at once. revealBid is the API called by a transport vehicle to accept a winning bid. The getAuctionWinner API call returns the lowest bidder in the auction. The only function that showcased varying gas prices with increasing load factor was the getAllTenders function. Each of these functions would be called numerous times throughout daily use of the prototype, and low variance in gas prices is a strong attribute of the API calls.

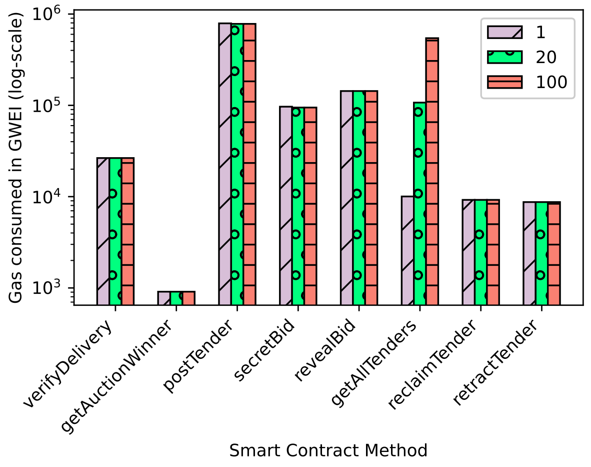
Evaluation of Gas Estimates with Current Crypto Prices

Table 5.2 showcases sample gas prices in GWEI for each of the smart contract API calls. This is based off static load with 100 API calls, similar to Figure 5.1a and Figure 5.1b. The price of Ethereum, Polygon, and Avalanche are all as of October 8th, 2023. Implementing the prices with our gas estimates gives us insights into which APIs have the highest cost.

Table 5.2 showcases that using a blockchain network like Avalanche [46] provides very low costs for our transactions. Due to this, transaction cost should not be a concern for those who utilize the application properly, and the result of transaction fees should be negligent for users.



(a) Account management related API cost with varying load

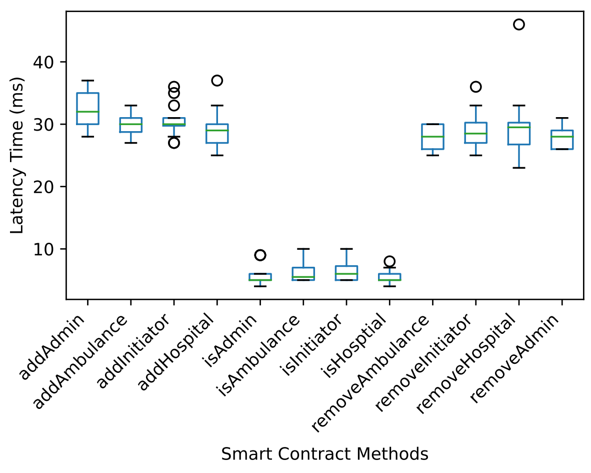


(b) Auction related API cost under varying load factor

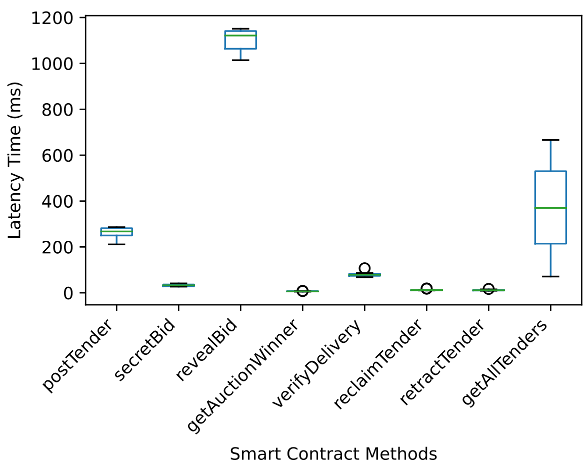
Figure 5.2: Gas estimate (log-scale) with varying load factor. The load factor is the number of calls to a given API function (shown in legend).

Table 5.3: Description of API smart contract functions in Auction smart contract

API Name	**User Invoked By**	**Description**
postTender	Transport Hirer	Creates a tender for transport providers to bid on with either a time limit delivery period, or a set delivery date. This API call starts the sealed-bid auction. The hirer gives information such as, patient location, severity of injuries, etc. to help transporters respond best to the situation
secretBid	Transport Provider	This API call places the transporters sealed bid in a given tender. This API can only be used on a Tender that is under auction
revealBid	Transport Provider	Invoked by transport providers, this API call occurs after the auction has ended and is within the reveal period. A transporter must have a bid placed for the given tender. Once revealed, the transporter will have to check the getWinner API after the reveal period ends
getWinner	Transport Provider	getWinner tells transporters whether they won the auction and need to go pick up the patient at the specified location
verifyDelivery	Delivery Verifier	This API is invoked once a patient is successfully delivered to an allowed verifier. Once the patient is verified, payment funds are transmitted to the transport provider
reclaimTender	Transport Hirer	reclaimTender is used on tenders that are past the due date of the auction. The purpose of this is to allow the transport hirers to reclaim their funds for a failed job
retractTender	Transport Hirer	This API is used on tenders that went through the auction period, but did not get successfully claimed by a transport provider.



(a) account management related APIs



(b) auction related APIs

Figure 5.3: Time delay estimate (in ms) for API calls with 10 API calls.

5.1.2 Tracking Time Needed for API Calls Using Truffle

Truffle [93] as Web3 development environment was used as the platform to deploy the SCeFSTA prototype. Truffle made it possible to compile, develop, and visually test the prototype.The API call's time for execution was also tracked using Truffle [93]. This allowed for seeing the trade-off between time (retrieved using Truffle and cost (gas prices retrieved using Foundry) to allow for optimizing the program to be as efficient as possible.

The time for execution of each of the functions was tracked. Figure 5.3a and Figure 5.3b showcase the smart contract API latency (in ms) for the Account and Auction smart contracts, respectively. In Figure 5.3a the box plots showcase a relatively consistent latency delay of around 30ms for the add account functions and the remove account functions, and the isAccount functions are very quick around 5ms.

When looking at Figure 5.3b for the Auction contract, it is evident that the majority of the API functions are very short, somewhere between 50ms-100ms, except for getAllTenders and revealBid. getAllTenders shows a large variance between about 200ms and 600ms. revealBid has a lower variance, but it has a much higher delay of a median around 1100ms. These lower times for these two functions would not cause any issues within the auction process, but are worth noting. These variances in latency are due to the fact that increasing the amount of tenders in the system increases the latency for API calls such as revealBid and getAllTenders.

5.1.3 Overall Requirement Satisfaction

There are several features implemented in SCeFSTA; these features aim to increase the efficiency, usability, and operation of the SCeFSTA application. Each of these features is discussed in more detail below.

Multiple Domain Support

Much of the feedback in the first iteration of the design cycle demonstration step, showcased in Figure 3.1, was focused on providing support for many domains within the healthcare transport sector. Given this feedback, SCeFSTA it was adjusted to include the domains of Emergency Response, Inter-facility Patient Transfer, and Non-Emergency Medical Transports. This required adding in multiple types of initiators, Private, Facility, and Emergency. Each of these roles has different options when creating a tender. The addition of these roles creates the ability of SCeFSTA to be easily adjusted to many domains, increasing the flexibility.

Emergency and Appointment Tenders

Part of having flexible domains requires a flexible auction system for posting tenders. This requires the use of two types of tenders, emergency and appointment. Emergency tenders allow an emergency or a facility initiator to create a critical tender with a set auction period and delivery time (i.e. transports have ten minutes to move from one facility to another). Appointment tenders are accessible to facility and private initiators; they allow the user to schedule an appointment with a facility at a specific time, rather than after a certain period of time, such as the emergency tenders. The appointment tenders also limit the number of facilities the patient can be delivered at to a single facility; meanwhile, an emergency tender can have several potential hospitals for delivery.

Allowing Specific Delivery Facilities

Another important feature in SCeFSTA is the ability for initiators to select options for the delivery location. The initiator is given options for registered facilities within the system and are able to set designated facilities that are allowed to accept the patient. Along with this list, each facility has a distance to the location from the incident, as well as the time it takes to get from the incident to each facility by utilizing the Google API. This feature helps give the patients options to ensure they are taken to a facility that suits them best.

Wallet Transparency

In order to reduce accounting errors inside MetaMask, SCeFSTA showcases MetaMask information, so the user is able to see their current account address, as well as the balance in the supported blockchain network. This allows users to easily see if they have enough for a transaction and help prevent errors of logging into the incorrect account.

5.2 Validation

The requirements for this project were outlined in Figure 3.2; these requirements were derived from our first round of interviews in the early stages of our research. After designing, implementing, and evaluating the prototype, we validated the prototype through a user study of eleven Miami University students as potential users. These students were presented with background information on our initial interviews, the requirements derived from the interviews, and background information on the auction system/application as a whole. After going through the background information of the project, the students watched a short demo of the system in action. Once the students had seen the demo, they were asked questions to validate whether we satisfied all the requirements outlined, as well elicit for any major issues our application may have. In essence, we sampled eleven Miami University students with a series of quantitative questions to validate we satisfied the requirements defined in the thesis, and then we asked them qualitative questions pertaining to what improvements could be made. The qualitative questions were open-ended questions for the participants in the user study. These questions aim to find any issues or improvements that could be made. Details on the feedback are listed in this section. The results from the user study for each requirement will be discussed in more detail in this section.

Our initial goal for validating our study was to interview stakeholders of the system, similar to our preliminary interviews. However, we spent extensive time trying to contact and conduct interviews with these stakeholders. Despite trying to utilize personal connections, connections through our previous interviews, and connections through the university, we failed to generate the interest in our project and make the necessary connections to perform these interviews. Ultimately, this was a learning point, showcasing that people in the medical field and healthcare transportation field can be difficult to track down and schedule a meeting with. Not being able to create a good line of contact with these stakeholders required that we adjust our approach and instead survey students.

5.2.1 Requirements Validation

The students were asked simple questions that pertained to requirements outlined in our first round of interviews in Figure 3.2. These questions were simple yes or no questions so we could get straightforward responses about our requirements. The requirements we sampled based on were guaranteed payment, resource optimization, lower costs, trust, timeliness, flexibility, and fairness. The quantitative questions we asked pertaining to these requirements are as follows:

- Do you think the system adheres to the promise of guaranteed payment? Does it implement a penalty for failed/late delivery?
- Are we reducing the wastage of resources?
- Do you envision this to lower costs to customers/patients?
- Do you think this auction system will promote fair competition?
- Do you expect it to enhance trust among all the involved parties?
- Do you think it will encourage the timely delivery of patients?
- Do you think the system is flexible enough for the users and service providers? Is it applicable to the EMS, NEMT, and Inter-facility domains?

Each participant answered “yes” for all seven of the questions that correlated to the design requirements we outlined in this thesis. Ultimately, each of the requirements we sampled the user-study participants on were deemed as satisfied by each participant.

5.2.2 Concerns and Potential Fixes

We sampled participants on if they had any major concerns with our system and if they had any fixes for these issues with the question, “What are your major concerns with this proposed prototype? What could be changed about the prototype to fix these issues?”

The majority of the study participants did not have major concerns with our application; however, we did receive valuable feedback on features that could be implemented to improve any concerns the remainder of the participants had.

There were several concerns presented along with potential fixes for issues presented:

- Supply a backup vendor if a transport provider fails to pick up a patient. This would keep a tender from being forgotten and a patient not receiving service
- Provide a response if a tender does not receive any bids due to a user input error or for some other reason. This is partially handled by the reclaim and retract tender option, but this feature could still be further improved for better patient support
- Another concern was about the load factor of multiple people placing bids at once. The effects of multiple bids at once is showcased in the Evaluation section within this chapter

5.2.3 User Interface (UI) Evaluation

We asked the participants: “What did you think of the user interface? Are there any user interface changes that would make the prototype faster or easier to use?” One participant stated:

> I thought the user interface was really well organized. It was definitely easy to follow, even as somebody who doesn’t know a lot about health care.

The responses pertaining to the user-interface were very positive, and several users thought the UI was straightforward and well organized.

5.2.4 Efficiency and Usability Improvements

There were several recommendations and features that were given in the user-study to improve the efficiency and usability of our application. This feedback was based on the question, “Are there any features that you think could make an improvement in the efficiency or usability of the proposed system?”

We wanted to attempt to get ideas for new features and gaps in the application that would improve usability of the system. We got the following feedback from the participants:

- Create a notification system/pop-up countdown for bids and revealing bids to alert transports when they need to act

- Provide an alert system for transportation providers so they can alert others they are not able to proceed with a transport
- Refine the penalty system to also include longer-lasting effects, transports who consistently perform poorly are no longer able to participate in auctions
- Provide a means of automation/API integration, so people can set up preferences and automatically bid on open tenders
- Implement an autofill address section to streamline creating tenders and inputting patient locations

5.2.5 Alterations on Sealed-Bid Auction

To sample the participants on the sealed-bid auction, we asked them, "How do you think the sealed-bid system could be improved?" One response recommended,

> I think for emergencies, allowing the emergency initiators to have more control over who gets the bid instead of just going to the lowest bidder so that life is best preserved

This could potentially be a good implementation, but based off the other feedback, is not an essential implementation of our sealed-bid auction system. The feedback from this question was largely positive, and each of the recommendations were relatively minor. One member liked the concept of the sealed-bid, but thought it would be helpful to list the number of bids on each tender to gauge interest from other companies. Another mentioned creating an incentive system for high performing transports, and another participant recommended showcasing prices in USD instead of cryptocurrency for better transparency of cost.

5.2.6 Privacy Concerns

We asked participants, "What are your opinions and concerns about privacy?" to sample if the participants have any concerns about patient and user privacy from our application.

A few of the participants had concerns about privacy and HIPAA violations, but as mentioned prior, measures are implemented to ensure that there is no sensitive patient information stored. In addition to this, any information is stored on a private database, and we do not store any patient data on the blockchain smart contract.

One participant mentioned this about privacy concerns relating to HIPAA violations,

> Medical emergencies are covered by HIPAA laws. I believe this system would need to take information security incredibly seriously. Some information might need to be hidden from bidders to better provide privacy to patients. The system should operate on a need-to-know basis

As discussed in previous sections of this thesis, HIPAA violations were thoroughly considered in this application. We made sure to not store any defining information about each patient, and any information that is stored is not on the blockchain network and is only accessible in the private database to authorized users. Another participant reinforces this by stating,

> From what I was able to see, there was not too much privacy concerns, there was not too much information listed about the patient other than their injury

These recommendations from the user-study participants showcase features that could be implemented in the future to provide further improvements on SCeFSTA. The open-ended responses were largely positive, and the majority of the concerns people have are satisfied by features of SCeFSTA that were not mentioned in the video demonstration.

Chapter 6

Conclusion

Given the limitations and issues within the current healthcare transportation sector and the emergence of new blockchain technologies, improvements can be made to the healthcare transportation sector using an auction system on a blockchain network. This will provide benefits to workers and patients within the healthcare field sector while providing more instantaneous transactions for transportation providers. This thesis proposes Smart Contract enabled Fair, Secure, and Transparent Auction (SCeFSTA), which promotes competition between healthcare transportation companies across several different healthcare transportation domains. This thesis will also delve into the fairness aspect of using blockchain based auctions through smart contracts to create fair competition among transportation companies. Fairness and competition will help solve staffing issues within the healthcare transport field by giving companies the option to not bid beyond their means for patients.

This thesis contributes SCeFSTA, a web-hosted NextJS [86] website providing an interface for users to access the blockchain auction system. The auction system provides a sealed-bid first price auction, in which the lowest bidder when the auction ends will be the winner. The blockchain auction system has several users. A transport hirer who starts the auction with patient injury and location information. A transport provider who bids on the individual tenders created by the hirers, the winner picking up the patient. A delivery verifier who validates that a patient was delivered by the transport provider. There is also an administrator who handles all the other users and has the ability to add and remove other users in the system.

The limitations of this thesis are that the application is not guaranteed to provide a solution to the issues within the healthcare transportation sector; however, we interviewed stakeholders of the system to limit the possibility to this. In addition to this, the application can always be improved and add new features that could ultimately provide a stronger user experience, but also might create confusion and clutter within the application.

This application is highly scalable and has a multitude of features that could be implemented that, while not necessary, might improve user's quality of life. User Interface features such as autofilling addresses, a pop-up countdown timer for auction endings, and a notification system for auction results were all potential features recommended in our user study. These recommendations showcase that there are almost always improvements that can be made to the UI to improve user experience for a fully-developed application. There were also larger features that were recommended such as creating a backup transport provider, so a patient is never left without service, create a more extensive penalty system to automatically remove users from the application who fail to delivery patients on multiple occasions, and an automation system, so companies can automatically place bids based off certain criteria. These changes would provide extensive alterations to the application.

SCeFSTA was designed with the feedback of stakeholders within the healthcare transport sector. Directly creating benefits within the domain, and creating a fair and efficient system that improves patient experience and costs. The application was validated through a user-study based directly on the requirements derived from the stakeholders interviewed. Showcasing SCeFSTA successfully satisfies all the requirements to provide improvements on the existing healthcare transportation system.

Chapter 7

Appendix A

7.1 Unfair smart contract code

The following code snippet showcases a sample smart contract function that has an absence of logic. The code does not verify that the auction is still open, that the bid placed is the highest, or if the expiration date of the auction has passed [74].

```
pragma solidity ^8.0.0

contract SampleContract {

    // function for user to place bid in auction w/ auctionId
    function placeBid(uint auctionId) payable {
        // get the auction information based on given auction id
        Auction memory auct = AuctionArr[auctionId];

        // assign the higest bidder and highest bid to the bidder
        auct[auctionId].highestBidder = payable(msg.sender);
        auct[auctionId].highestBid = msg.value;
    }
}
```

The code example below highlights a contract function that properly accepts an auction bid as the new highest bid, but fails to set the new highest bid [74].

```
pragma solidity ^8.0.0

contract SampleContract {

    // function for user to place bid in auction w/ auctionId
    function placeBid(uint auctionId) payable {
        // get the auction information based on given auction id
        Auction memory auct = AuctionArr[auctionId];

        // ensure the auction is still open, the auction is not
        // expired, and the bid is larger than the highest bid
        require(auct.status == 'open', "auction is not open");
        require(block.timestamp < auct.experationTime);
        require(msg.value > auct.highestBid);

        // assign the higest bidder and highest bid to the bidder
        auct[auctionId].highestBidder = payable(msg.sender);
        }
    }
```

The code below has no logic errors but does not account for outside influences that can result in an unfair smart contract [74].

```
    pragma solidity ^8.0.0

    contract SampleContract {

        // function for user to place bid in auction w/ auctionId
        function placeBid(uint auctionId) payable {
            // get the auction information based on given auction id
            Auction memory auct = AuctionArr[auctionId];

            // ensure the auction is still open, the auction is not
            // expired, and the bid is larger than the highest bid
            require(auct.status == 'open', "auction is not open");
            require(block.timestamp < auct.experationTime);
            require(msg.value > auct.highestBid);

            // assign the higest bidder and highest bid to the bidder
            auct[auctionId].highestBidder = payable(msg.sender);
            auct[auctionId].highestBid = msg.value;
        }
    }
```

Chapter 8

Appendix B

Appendix B showcases the smart contracts utilized in SCeFSTA. The full code repository is public and available on GitHub. The link to the GitHub repository is in the footnote on this page [1]

8.1 Accounts Smart Contract

The code outlined in 8.1 holds all the account information for the users in SCeFSTA. It holds information on each user and is assisted by the User database. Details about specific API calls of the Account smart contract can be found in Table 5.1.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

contract Accounts {
    // super admin who has access to all features
    address public superAdmin;

    // four user accounts
    mapping(address=> bool) private admins;
    mapping(address => bool) private initiators;
    mapping(address => bool) private ambulances;
    mapping(address => bool) private hospitals;

    // set the superAdmin and initalize a few accounts
    constructor() {
        // add superAdmin to all account types (for demo purposes only)
        superAdmin = msg.sender;
        admins[superAdmin] = true;
        initiators[superAdmin] = true;
        ambulances[superAdmin] = true;
        hospitals[superAdmin] = true;

        // setup default accounts
        address emergencyInitiator = 0
           xcdF98E3f41A0160360884f67BF8FfF35D92d4E2f;
        address hospital = 0x37b17D21569C2cA6c7A078f2283D06BC222F554C;
        address ambulance = 0x9f8d25e9e3261d328e1Bef34CdbadB9310E451Fc;
        address admin = 0xEE8fb1E70B2Cd462cC0eE0ABb12B36db6D0932B2;
        address privateInitiator = 0xC53762A6D1E4557Ab363eE38042828fcfBF064bE;
```

[1] GitHub Repo: https://github.com/sbhunia/scefsta

```
        address interfacility = 0xb153eDE174EDC76EA00D706ce678b3aF28379887;

        // allocate accounts
        admins[admin] = true;
        hospitals[hospital] = true;
        hospitals[interfacility] = true;
        initiators[emergencyInitiator] = true;
        initiators[privateInitiator] = true;
        initiators[interfacility] = true;
        ambulances[ambulance] = true;


    }

     // returns true if address is an ambulance, false if not
    function isAmbulance(address addr) public view returns (bool) {
        return ambulances[addr];
    }

     // Allows the admin to add verified ambulances
    function addAmbulance(address addr) public {
        // verify account does not already have another role
        require(hospitals[addr] == false, "Already facility");
        require(ambulances[addr] == false, "Already a transport");
        require(initiators[addr] == false, "Already initiator");
        require(admins[addr] == false, "Already admin");

        // verify the sender is an admin
        require(admins[msg.sender] == true, "Sender must be admin");

        // if passes all requires add account to ambulances
        ambulances[addr] = true;
    }

    // Allows the admin to remove verified ambulances
    function removeAmbulance(address addr) public {
        require(admins[msg.sender] == true, "Must be admin");
        require(ambulances[addr] == true, "Must be a transport");
        ambulances[addr] = false;
    }

    // returns true if address is an poice, false if not
    function isInitiator(address addr) public view returns (bool) {
        return initiators[addr];
    }

    // Allows the admin to add verified police stations
    function addInitiator(address addr) public {
        // verify account does not already have another role
        require(ambulances[addr] == false, "User already transport");
        require(initiators[addr] == false, "User already initiator");
        require(admins[addr] == false, "User already admin");

        // verify sender is an admin
        require(admins[msg.sender] == true, "Sender must be an admin");

        // if passes all requires add the address as police
        initiators[addr] = true;
    }

    // Allows the admin to remove verified police stations
    function removeInitiator(address addr) public {
        require(admins[msg.sender] == true, "Sender must be an admin");
        require(initiators[addr] != false, "Must be initator");
        initiators[addr] = false;
    }

    // returns true if address is a hospital, false if not
    function isHospital(address addr) public view returns (bool) {
        return hospitals[addr];
    }

    // Allows the admin to add verified hospitals
    function addHospital(address addr) public {
        // verify account does not already have another role
        require(hospitals[addr] == false, "Already facility");
        require(ambulances[addr] == false, "Already transport");
        require(admins[addr] == false, "Already admin");

        // verify the sender is an admin
        require(admins[msg.sender] == true, "Sender must be admin");

        // if passes all requires add account to hospitals
        hospitals[addr] = true;
    }

    // Allows the admin to remove verified hospitals
    function removeHospital(address addr) public {
        require(admins[msg.sender] == true, "Sender must be an admin");
        require(hospitals[addr] != false, "Not already a facility");
        hospitals[addr] = false;
    }

    // returns true if address is an admin, false if not
    function isAdmin(address addr) public view returns (bool) {
        return admins[addr];
    }

    // add a new admin
    function addAdmin(address addr) public {
        require(ambulances[addr] == false, "Already transport");
        require(hospitals[addr] == false, "Already facility");
        require(initiators[addr] == false, "Already initiator");
        require(admins[addr] == false, "Already an admin");
        require(addr != superAdmin, "superAdmin is already an admin");
        require(msg.sender == superAdmin, "Sender must be superAdmin");
        admins[addr] = true;
    }

    // remove an existing admin
    function removeAdmin(address addr) public {
        require(msg.sender == superAdmin, "Sender must be a superAdmin");
        require(admins[addr] == true, "Not an admin");
        require(addr != msg.sender, "Cannot remove yourself as admin");
        admins[addr] = false;
    }
}
```

8.2 Auction Smart Contract

The code in section 8.2 showcases the auction smart contract code utilized in this thesis. This code holds each smart contract API required to run the sealed-bid auction based system. More information about these API calls can be found in Table 5.3.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import "./Accounts.sol";

contract Auctions {
    // initialize accounts contract
    Accounts accountsContract;

    // constants
    uint256 constant MAX_INT = 2**256 - 1;
    uint256 constant REVEAL_PERIOD = 60;

    // enum for various possible tender status's, closed is default
    enum TenderStatus {
        Closed,
        InProgress,
        Open,
        Retracted,
        Reclaimed
    }

    // helper struct for tender details
     struct TenderDetails {
        address payable tenderPoster;
        address payable tenderAccepter;
        uint256 postDate;
        uint256 auctionDate;
        uint256 revealDate;
        uint256 dueDate;
        string addr;
        string city;
        string state;
        string zipcode;
        address[] allowedHospitals;
        uint256 maxBid;
        uint256 penalty;
        // Maps an address to its bid. This enforces one bid per address,
        //so that ambulances do not just bid as many prices as they can,
        //and then reveal lower and lower bids during the reveal period.
        // We use 2 arrays because we can't have nested mappings in Solidity.
        address[] bidders;
        uint[] bidHashArray;
        uint256 finalBid;
        string severity;
    }

    // struct to hold tender information
    struct Tender {
        TenderDetails details;
        TenderStatus status;
        uint256 tenderId;
    }

    // data members for tenders
    uint256[] public tenderIds; // list of all tenders
    mapping(uint256 => Tender) public tenderMapping; // mapping for faster
       tender lookup
    Tender[] public tenders;
    uint256 tenderIdCounter = 0;

    // constructor, references the Accounts contract address
    constructor(address contractAddress) {
        accountsContract = Accounts(contractAddress);
    }

    /*
    * Function posts new tender from initiator and begins the auction for the
       tender
    *
    * @param timeLimit - length of the auction period
    * @param deliveryTime - how long the ambulance has to transport the
       patient after auction and reveal
    * @param addr - address of the tender location
    * @param city - city of the tender location
    * @param state - state of the tender location
    * @param penalty - penalty amount charged for lack of delivery for
       accepted tender
```

```
 * @param severity - severity of the tender patient
 * @param allowedHospitals - list of valid hospitals to be transported to
 * @returns - returns the id of the new tender posted
 * Note: msg.value is the max bid
 */
function postTender(
    uint256 timeLimit,
    uint256 deliveryTime,
    string memory addr,
    string memory city,
    string memory state,
    string memory zipcode,
    uint256 penalty,
    string memory severity,
    address[] memory allowedHospitals
) public payable returns (uint256) {
    require(accountsContract.isInitiator(msg.sender), "Sender not
       initiator");
    require(msg.value > 0, "Max Value < 0");

    // validate allowed hospitals given are valid hospitals
    for (uint256 i = 0; i < allowedHospitals.length; i++) {
        require(accountsContract.isHospital(allowedHospitals[i]), "
           Hospital(s) invalid");
    }

    // initalize new tender information
    Tender memory tender;
    tender.tenderId = tenderIdCounter;
    tender.details.tenderPoster = payable(msg.sender);
    tender.status = TenderStatus.Open; // open tender

    // set important dates for tender
    tender.details.postDate = block.timestamp; // set post date to current
        time
    tender.details.auctionDate = block.timestamp + timeLimit; // sets
       auction end date w/ time limit given
    tender.details.revealDate = block.timestamp + timeLimit +
       REVEAL_PERIOD; // sets reveal peiod end date
    tender.details.dueDate = block.timestamp + timeLimit + REVEAL_PERIOD +
        deliveryTime; // set due date for tender

    // set tender information
    tender.details.finalBid = MAX_INT; // set current bid to max possible
       integer
    tender.details.maxBid = msg.value; // set maximum bid to msg.value
    tender.details.penalty = penalty;
    tender.details.allowedHospitals = allowedHospitals;

    // patient location and details
    tender.details.addr = addr;
```

```
        tender.details.city = city;
        tender.details.state = state;
        tender.details.zipcode = zipcode;
        tender.details.severity = severity;

        // push new tender and adjust mappings accordingly
        tenders.push(tender);
        tenderMapping[tenderIdCounter] = tender;
        tenderIdCounter++;
        return tender.tenderId;
    }

    event BidPlaced(uint256 tenderId, uint256 bidId);

    /*
    * Function places a secret bid by an ambulance. Each ambulance receives
       only 1 bid
    *
    * @param tenderId - the ID of the tender being bid on
    * @param bidHashedAmount - the hash value of the bid being places (bid +
       random salt value)
    * @returns the ID of the ambulances bid, used for revealing
    */
    function secretBid(uint256 tenderId, uint256 bidHashedAmount) public
       payable returns(uint256 index) {
        Tender memory tender = tenderMapping[tenderId];
        require(accountsContract.isAmbulance(msg.sender), "Sender must be an
           ambulance");
        require(tender.status == TenderStatus.Open, "Tender not open");
        require (block.timestamp < tender.details.auctionDate, "Auction has
           passed");
        require(msg.value == tender.details.penalty, "Incorrect penalty amount
           ");
        require(!contains(tender.details.bidders, msg.sender), "Can only bid
           once");

        // add bid to array
        tenderMapping[tenderId].details.bidders.push(msg.sender);
        tenderMapping[tenderId].details.bidHashArray.push(bidHashedAmount);
        emit BidPlaced(tenderId, tenderMapping[tenderId].details.bidders.
           length - 1);
        return tenderMapping[tenderId].details.bidders.length - 1;
    }

    /*
    * Function is used to reveal winning bids after the auction period ends
    *
    * @param tenderId - ID of tender being revealed
    * @param bidVal - amount bid by the ambulance
    * @param saltVal - the random salt integer used in secretBid
    * @param index - index of placed secret bid
    */
    function revealBid(
        uint256 tenderId,
        uint256 bidVal,
        uint256 salt,
        uint256 index
    ) public payable {
        Tender storage tender = tenderMapping[tenderId];
        require(accountsContract.isAmbulance(msg.sender), "Sender must be
           ambulance");
        require(block.timestamp > tender.details.auctionDate, "Tender under
           auction");
        require(block.timestamp < tender.details.revealDate, "Tender past
           reveal period");
        require(tender.status == TenderStatus.Open, "Tender not open");
        require(bidVal < tender.details.maxBid, "Bid was not below max bid
           amount");
        require(msg.sender == tender.details.bidders[index], "Wrong bid ID");
        require(tender.details.penalty == msg.value, "Incorrect penalty amount
           ");
        require(tender.details.bidHashArray[index] == uint256(keccak256(abi.
           encodePacked(bidVal + salt))), "Bid value != Hash Value");

        // if job was already assigned, refund
        if (tender.details.tenderAccepter != address(0)) {
             tender.details.tenderAccepter.transfer(
                 tender.details.penalty
             );
        }

        // update tender information
        tender.status = TenderStatus.InProgress;
        tender.details.tenderAccepter = payable(msg.sender);
        tender.details.finalBid = bidVal;

        // set new info to tender mapping
        tenderMapping[tenderId] = tender;
        tenders[tenderId] = tender;
    }

    /*
     * get the winner of an auction
     *
     * @param tenderId - given tender ID
     */
    function getAuctionWinner(uint256 tenderId) public view returns (address
       tenderWinner) {
        require(
            block.timestamp >
                tenderMapping[tenderId].details.revealDate,
                "Tender not past Reveal"
        );

        require(
            tenderMapping[tenderId].status != TenderStatus.Open, "Tender is
                open"
        );

        return tenderMapping[tenderId].details.tenderAccepter;
    }

    /*
    * this function is called upon delivery of a patient where the ambulance
        is then paid for delivery
    *
    * @param tenderId - id of the tender where delivery is being verified
    */
    function verifyDelivery(uint256 tenderId) public {
        require(accountsContract.isHospital(msg.sender), "Sender must be
            hospital");
        require(
            tenderMapping[tenderId].status == TenderStatus.InProgress,
            "Tender not in progress"
        );
        require(
            block.timestamp >
                tenderMapping[tenderId].details.revealDate,
            "Tender is not past reveal date"
        );

        require(contains(tenderMapping[tenderId].details.allowedHospitals, msg
            .sender), "Sender valid hospital");

        // if the tender was not delivered on time, do not send penalty amount
            back, else return all funds
        if ( block.timestamp < tenderMapping[tenderId].details.dueDate) {
            tenderMapping[tenderId].details.tenderAccepter.transfer(
            tenderMapping[tenderId].details.finalBid);
        } else {
            // transfer agreed upon funds to the tender accepter
            tenderMapping[tenderId].details.tenderAccepter.transfer(
            tenderMapping[tenderId].details.finalBid + tenderMapping[tenderId
                ].details.penalty
            );
        }

        // transfer funds back to the tender poster
        if (tenderMapping[tenderId].details.maxBid - tenderMapping[tenderId].
            details.finalBid > 0) {
            tenderMapping[tenderId].details.tenderPoster.transfer(
                tenderMapping[tenderId].details.maxBid -
                    tenderMapping[tenderId].details.finalBid
            );
        }

        // close the tender
        Tender storage referencedTender = tenderMapping[tenderId];
        referencedTender.status = TenderStatus.Closed;
        tenderMapping[tenderId] = referencedTender;
        tenders[tenderId] = referencedTender;
    }

    /*
     * get all the tenders present in tenders array
     *
     * @returns all tenders on the blockchain
     */
    function getAllTenders() public view returns (Tender[] memory) {
        return tenders;
    }

    // /*
    //  * Get a tender from a given tenderID
    //  *
    //  * @param tenderId - given tenderID
    //  *
    //  * @returns a single tender with given tenderId
    //  */
    //  function getTender(uint256 tenderId) public view returns (Tender
       memory) {
    //      return tenderMapping[tenderId];
    //  }

    /*
     * allows police stations to reclaim their funds + the penalty for failed
        jobs
     *
     * @param tenderId - ID of tender being retracted
     */
    function reclaimTender(uint256 tenderId) public {
        require(
            tenderMapping[tenderId].details.tenderPoster == msg.sender,
            "Sender is not the tender poster"
        );
        require(
            tenderMapping[tenderId].status == TenderStatus.InProgress,
            "Tender not in progress"
        );
        require(
            tenderMapping[tenderId].details.tenderAccepter != address(0),
            "Tender accepter is not the address set"
        );
        require(block.timestamp < tenderMapping[tenderId].details.dueDate, "
            Auction is over");

        tenderMapping[tenderId].details.tenderPoster.transfer(
            tenderMapping[tenderId].details.maxBid + tenderMapping[tenderId].
                details.penalty
        );

        // update status of the tender
        tenderMapping[tenderId].status = TenderStatus.Reclaimed;
        tenders[tenderId].status = TenderStatus.Reclaimed;
    }

    /*
     * remove a tender with auction in progress from auction period
     *
     * @param tenderId - ID of tender being retracted
     */
    function retractTender(uint256 tenderId) public {
        require(
            msg.sender == tenderMapping[tenderId].details.tenderPoster,
            "Sender is not the tender poster"
        );
        require(
            tenderMapping[tenderId].status == TenderStatus.Open,
            "Tender not open"
        );

        // can only retract a tender if bid period is over (need to add)
        tenderMapping[tenderId].details.tenderPoster.transfer(
            tenderMapping[tenderId].details.maxBid
        );

        // update status of the tender
        tenderMapping[tenderId].status = TenderStatus.Retracted;
        tenders[tenderId].status = TenderStatus.Retracted;
    }

    // This sucks, but it's one of the consequences of storing everything on
        the blockchain
    function contains(address[] memory addresses, address addressToFind)
       private pure returns (bool doesContain) {
        for (uint256 i = 0; i < addresses.length; i++) {
            if (addresses[i] == addressToFind) {
                return true;
            }
        }
        return false;
    }
}
```

References

[1] Youngcheoul Kang, Nakbum Choi, and Seoyong Kim. Searching for new model of digital informatics for human–computer interaction: testing the institution-based technology acceptance model (itam). *International journal of environmental research and public health*, 18(11):5593, 2021.

[2] Shyamli Jha. The Complete Guide for Types of Blockchain `https://www.simplilearn.com/tutorials/blockchain-tutorial/types-of-blockchain`.

[3] Gwyneth Iredale. Blockchain Vs. Database: Understanding the Difference. `https://101blockchains.com/blockchain-vs-database-the-difference/`.

[4] Giang-Truong Nguyen and Kyungbaek Kim. A survey about consensus algorithms used in blockchain. *Journal of Information processing systems*, 14(1):101–128, 2018.

[5] R. Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738, 1987.

[6] Aron Abraham. Avalanche (AVAX), the platform for fast smart contracts in the spotlight. `https://blog.bitpanda.com/en/avalanche-avax-the-platform-for-fast-smart-contracts-in-the-spotlight`.

[7] Avalanche vs Ethereum — Which Is Better? AVAX and ETH Comparison. `https://stealthex-io.medium.com/avalanche-vs-ethereum-which-is-better-avax-and-eth-comparison-9bb97841b84d`.

[8] Crypto Currency Price. `https://goldprice.org/cryptocurrency-price`.

[9] Surang. `https://www.flaticon.com/authors/surang`.

[10] Fasil. `https://freeicons.io/profile/722`.

[11] Becris. `https://becrisdesign.com/`.

[12] Chattapat. `https://freeicons.io/profile/101237`.

[13] Anu Rocks. `https://freeicons.io/regular-life-icons/house-icon-17847`.

[14] GeeksForGeeks. Blockchain Strucutre. `https://www.geeksforgeeks.org/blockchain-structure/`.

[15] Royyan Wijaya. https://www.flaticon.com/free-icon/cube_7560704?term=blockchain+block&page=1&position=8&origin=search&related_id=7560704.

[16] Freepik. https://www.flaticon.com/free-icon/chains_419121?term=chain+link&page=1&position=12&origin=search&related_id=419121.

[17] Freepik. https://www.flaticon.com/free-icons/blockchain.

[18] Ultimatearm. https://www.flaticon.com/free-icons/general.

[19] Freepik. https://www.flaticon.com/free-icons/technology.

[20] Surang. https://www.flaticon.com/free-icons/transaction.

[21] Eucalyp. https://www.flaticon.com/free-icons/confirmation.

[22] Zachary F Meisel, Jesse M Pines, Daniel Polsky, Joshua P Metlay, Mark D Neuman, and Charles C Branas. Variations in ambulance use in the united states: the role of health insurance. *Acad. Emerg. Med.*, 18(10):1036–1044, October 2011.

[23] American Hospital Association. Transportation and the Role of Hospitals. https://www.aha.org/ahahret-guides/2017-11-15-social-determinants-health-series-transportation-and-role-hospitals.

[24] Carter Evans and Simon Bouie. U.S. faces shortage of EMTs, nearly one-third quit in 2021. https://www.cbsnews.com/news/emt-shortage-quit-ambulance/.

[25] Victor Cheung. 9 Problems Your EMS Billing Software Should've Solved Yesterday. https://traumasoft.com/ems-software/9-problems-your-ems-billing-software-shouldve-solved-yesterday/.

[26] EMS-gov. What is ems? https://www.ems.gov/whatisems.html.

[27] AIM EMS software & services. The ems workflow: Five critical phases of ems software. https://www.aim-system.com/blog/the-ems-workflow-five-critical-phases-of-ems-software.

[28] Joseph Heaton and Melissa D Kohn. Ems inter-facility transport. In *StatPearls [Internet]*. StatPearls Publishing, 2021.

[29] Eric Sy and Terrance Ross. Air ambulance transport. *CMAJ*, 193(37):E1462–E1462, 2021.

[30] Sandra Rosenbloom. Transportation needs of the elderly population. *Clinics in Geriatric Medicine*, 9(2):297–310, 1993. Medical Consideration in the Older Driver.

[31] DataTech911. An Overview Of MCIs (Multi-Casualty Incidents). https://www.datatech911.com/resources/ems-guide-mass-casualty-incidents/.

[32] Joseph J Doyle, Jr, John A Graves, and Jonathan Gruber. Uncovering waste in US healthcare. *J. Health Econ.*, 54:25–39, July 2017.

[33] James Boyd Eubanks. The ems deficit: A study on the excessive staffing shortages of paramedics and its impact on ems performance in the states of south carolina and north carolina and interventions for organizational improvements. *Liberty University*, 2022.

[34] Julie Haslam. Emergency medical services: Decreasing revenue and the regulated healthcare environment will ambulance transport providers survive? *Journal of Health Care Finance*, 42(2), 2015.

[35] Amy-Xiaoshi DePaola. `https://businessjournalism.org/2020/01/the-costs-of-emergency-medical-transportation/`.

[36] Josh Komenda. `https://www.forbes.com/sites/forbestechcouncil/2022/03/01/how-the-tech-industry-can-help-overcome-the-hidden-hurdle-to-healthcare-transportat` `?sh=4fba590a7825`.

[37] Adam Hayes. Blockchain facts: What is it, how it works, and how it can be used. `https://www.investopedia.com/terms/b/blockchain.asp`, September 2022.

[38] Richard Lee Twesige. A simple explanation of bitcoin and blockchain technology. *Comput. Sci*, 1:1–5, 2015.

[39] Saifedean Ammous. Blockchain technology: What is it good for? *Available at SSRN 2832751*, 2016.

[40] Adam Hayes. Stablecoins: Definition, How They Work, and Types. `https://www.investopedia.com/terms/s/stablecoin.asp`.

[41] L. M. Bach, B. Mihaljevic, and M. Zagar. Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1545–1550, 2018.

[42] Cointelegraph. A beginner's guide to the different types of blockchain networks. `https://cointelegraph.com/blockchain-for-beginners/a-beginners-guide-to-the-different-types-of-blockchain-networks#:~:text=A%20blockchain%20network%20is%20a,their%20copy%20of%20the%20ledger`.

[43] Ethereum. Ethereum. `https://ethereum.org/en/`.

[44] Ethereum. Ethereum layer2. `https://ethereum.org/en/layer-2/`.

[45] ByBit Learn. Network comparisons. `https://learn.bybit.com/blockchain/fastest-cryptocurrencies-high-tps/`.

[46] Avalanche. Avalanche. https://www.avax.network/.

[47] Shafaq Naheed Khan, Faiza Loukil, Chirine Ghedira-Guegan, Elhadj Benkhelifa, and Anoud Bani-Hani. Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-peer Networking and Applications*, 14:2901–2925, 2021.

[48] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[49] jagjit Singh. the life cycle of smart contracts in the blockchain ecosystem. https://cointelegraph.com/explained/the-life-cycle-of-smart-contracts-in-the-blockchain-ecosystem.

[50] Cardano. Cardano, making the world better for all. https://cardano.org/.

[51] Ripple. Ripple, business impact powered by crypto. https://ripple.com/?c1=GAW_SE_NW&source=US_BRND&cr2=search__-__us__-__brand--general__-__exm&kw=ripple_exm&cr5=611959309622&cr7=c&utm_source=GAW_SE_NW_US_BRND&utm_medium=cpc&utm_campaign=search__-__us__-__brand--general__-__exm&gclid=Cj0KCQiA6LyfBhC3ARIsAG4gkF-gy3FyDJNYD09x5npYb55Lb0msLNT0UhL0chnE817i_IJ99HVMAaQaAgUcEALw_wcB.

[52] BitStamp. What is Avalanche (AVAX)? https://www.bitstamp.net/learn/cryptocurrency-guide/what-is-avalanche-avax/.

[53] Reza M. Parizi, Amritraj, and Ali Dehghantanha. Smart contract programming languages on blockchains: An empirical evaluation of usability and security. In Shiping Chen, Harry Wang, and Liang-Jie Zhang, editors, *Blockchain – ICBC 2018*, pages 75–91, Cham, 2018. Springer International Publishing.

[54] Rob Hitchens. How can i call contract method that suicide and continue execution? https://ethereum.stackexchange.com/questions/64877/how-can-i-call-contract-method-that-suicide-and-continue-execution.

[55] LogRocket. The ultimate guide to data types in Solidity. https://blog.logrocket.com/ultimate-guide-data-types-solidity/#fixed-size-arrays.

[56] Cory Stieg. The Top Programming Languages Used For Blockchain Development. https://www.codecademy.com/resources/blog/programming-languages-blockchain-development/.

[57] Coinbase. Dai. https://www.coinbase.com/price/dai.

[58] Nichholas Rossolillo. What are Crypto Gas Fees? https://www.fool.com/investing/stock-market/market-sectors/financials/cryptocurrency-stocks/crypto-gas-fees/, date=10 June 2022.

[59] Jake Frankenfield. Wei: Definition in cryptocurrency, how it works, and history. https://www.investopedia.com/terms/w/wei.asp.

[60] wackerow. Introduction To DApps. https://ethereum.org/en/developers/docs/dapps/.

[61] React. React. https://reactjs.org/.

[62] ProCoders. How to Develop a DApp with React? https://procoders.tech/blog/how-to-build-a-dapp-react/#:~:text=dApps.

[63] Bitpay. Different Types of Crypto Wallets Explained. https://bitpay.com/blog/types-of-crypto-wallets/#:~:text=A%20crypto%20wallet%20securely%20stores,wallets%2C%20which%20function%20primarily%20offline.

[64] Andy Rosen. 11 Best Crypto Wallets of February 2023. https://www.nerdwallet.com/article/investing/best-bitcoin-cryptocurrency-wallet.

[65] MetaMask. Metamask. https://metamask.io/.

[66] Heather Boushey and Helen Knudsen. The Importance of Competition for the American Economy. https://www.whitehouse.gov/cea/written-materials/2021/07/09/the-importance-of-competition-for-the-american-economy/.

[67] Adam Hayes. https://www.investopedia.com/terms/r/reserve-price.asp.

[68] Dirk Bergemann, Tibor Heumann, Stephen Morris, Constantine Sorokin, and Eyal Winter. Optimal information disclosure in classic auctions. *American Economic Review: Insights*, 4(3):371–388, 2022.

[69] Cornell Law School. https://www.law.cornell.edu/wex/collusive_bidding#:~:text=Collusive.

[70] Ezekiel Soremekun, Mike Papadakis, Maxime Cordy, and Yves Le Traon. Software fairness: An analysis and survey. *arXiv preprint arXiv:2205.08809*, 2022.

[71] Yuriy Brun and Alexandra Meliou. Software fairness. In *Proceedings of the 2018 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 754–759, 2018.

[72] Ye Liu, Yi Li, Shang-Wei Lin, and Rong Zhao. Towards automated verification of smart contract fairness. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 666–677, 2020.

[73] Shuangke Wu, Yanjiao Chen, Qian Wang, Minghui Li, Cong Wang, and Xiangyang Luo. Cream: A smart contract enabled collusion-resistant e-auction. *IEEE Transactions on Information Forensics and Security*, 14(7):1687–1701, 2018.

[74] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: analyzing safety of smart contracts. In *Ndss*, pages 1–12, 2018.

[75] Stephen M. George, Wei Zhou, Harshavardhan Chenji, Myounggyu Won, Yong Oh Lee, Andria Pazarloglou, Radu Stoleru, and Prabir Barooah. DistressNet: a wireless ad hoc and sensor network architecture for situation management in disaster response. *IEEE Communications Magazine*, 48(3):128–136, 2010.

[76] Anil Saini, Shreyansh Sharma, Palash Jain, Vikash Sharma, and Arvind Kumar Khandelwal. A secure priority vehicle movement based on blockchain technology in connected vehicles. In *Proceedings of the 12th International Conference on Security of Information and Networks*, pages 1–8, 2019.

[77] Vinayak G Bhat, HP Pranaav, S Mini, and Deepak Tosh. Blockchain-centric resource management system for disaster response and relief. In *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 540–545. IEEE, 2021.

[78] Opensea. How do timed auctions work? `https://support.opensea.io/hc/en-us/articles/1500003246082-How-do-timed-auctions-work-`.

[79] Rarible. How do i sell my nft on rarible? `https://rarible.com/how-it-works/using-rarible/how-do-i-sell-my-nft-on-rarible`.

[80] Shirin Hasavari and Yeong Tae Song. A secure and scalable data source for emergency medical care using blockchain technology. In *2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 71–75. IEEE, 2019.

[81] Chakkaphong Suthaputchakun and Yue Cao. Blockchain-based secure ambulance-to-everything communications in emergency rescue operations. In *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 174–179. IEEE, 2022.

[82] Eman M Abou-Nassar, Abdullah M Iliyasu, Passent M El-Kafrawy, Oh-Young Song, Ali Kashif Bashir, and Ahmed A Abd El-Latif. Ditrust chain: towards blockchain-based trust models for sustainable healthcare iot systems. *IEEE Access*, 8:111223–111238, 2020.

[83] HIPAA for Professionals. `https://www.hhs.gov/hipaa/for-professionals/index.html`.

[84] Faiza Loukil, Chirine Ghedira-Guegan, Khouloud Boukadi, and Aïcha Nabila Benharkat. Towards an end-to-end IoT data privacy-preserving framework using blockchain technology. In *International Conference on Web Information Systems Engineering*, pages 68–78. Springer, 2018.

[85] Fantom, Innovation Unleashed. `https://fantom.foundation/`.

[86] The React Framework for Production. `https://nextjs.org/`.

[87] Amazon EC2: Secure and Resizeable Compute Capacity to Support Virtually Any Workload. https://aws.amazon.com/pm/ec2/?trk=36c6da98-7b20-48fa-8225-4784bced9843&sc_channel=ps&s_kwcid=AL!4422!3!467723097970!e!!g!!amazon%20ec2&ef_id=Cj0KCQiA4aacBhCUARIsAI55maETQzxRUosUoa_TglJ0qAUwyNNQRaDoZfJMNwrYr-ctK5zrn0zHUwEaAv0OEALw_wcB:G:s&s_kwcid=AL!4422!3!467723097970!e!!g!!amazon%20ec2.

[88] Kenneth Ballard. Secure programming with the OpenSSL API. https://developer.ibm.com/tutorials/l-openssl/, 2004.

[89] Solidity. https://docs.soliditylang.org/en/v0.8.17/.

[90] MySQL. https://www.mysql.com/.

[91] MongoDB: The Developer Data Platform. https://www.mongodb.com/.

[92] Foundry Book. https://book.getfoundry.sh/.

[93] Truffle: The Most Comprehensive Suite of Tools for Smart Contract Development. https://trufflesuite.com/.