Development of a Trust-enhanced Data Sharing System

Arian Fotouhi

A Thesis

In the Department of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Master of Applied Science (Electrical and Computer Engineering)

at Concordia University

Montréal, Québec, Canada

December 2022

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Arian Fotouhi**

Entitled: **Development of a Trust-enhanced Data Sharing System**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________________ Chair
Dr. Paula Lago

______________________________ Examiner
Dr. Yang Wang

______________________________ Supervisor
Dr. Jun Cai

Approved by ______________________________
Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

____________ 2023 ______________________________
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Development of a Trust-enhanced Data Sharing System

Arian Fotouhi

Despite importance of data sharing, there are several challenges yet to be tackled, especially security and privacy in which the users roughly may trust to utilize a platform that is insecure to protect privacy violation. Recently, blockchain technology as an immutable and decentralized ledger has emerged to be a promising solution to security and privacy issues. To improve the trust in data sharing systems, reputation systems are deemed advantageous. In fact, the users receive reputation based on their collaboration. It is worth noting, the blockchain-based data sharing and reputation systems impose complexities to the systems as well. This work presents an improved data sharing scheme in terms of privacy, security and latency. The idea features enhancement in performance of blockchain technology and reputation system. Besides, we have implemented the proposed idea as a web application to provide facilitated access for data sharing users. Most of the challenges of development are originated from the fact that many parts of this implementation is from scratch, unlike most of research works on data sharing implementations. We opted for this type of development because there exist common demerits in developments of data sharing frameworks and we believe the reason is to utilize the external and built-in tools imposing many constraints, e.g. communication costs. Besides, the development steps and their corresponding challenges are demonstrated as well. As the final stage, we evaluated the performance and depicted the results to compare and assess the quality of implemented data sharing framework.

Acknowledgments

I would like to express my sincere gratitude to Dr. Jun Cai for his guidances and supports that assisted me to overcome hurdles of my graduate studies path.

My appreciation extends to my parents as well who unconditionally supported me through all my life steps and nurtured my curiosity and ambition to follow my goals.

Contribution of Authors

This dissertation is the research work of Arian Fotouhi and it is submitted under supervision of Dr. Jun Cai for the degree of Master of Applied Science at Concordia University. The dissertation is original and unpublished.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides the requisite background information to grasp the concept and significance of data sharing.

1.1 Motivation

As the volume of data in our daily applications is surging, we have to seek new solutions to manage and distribute our information efficiently. As figure 1.1 shows, interests towards e-health, Internet of Things (IoT) and wearable devices have motivated researchers to pursue and devel op data sharing area with aim of producing privacy preserving and secure environment sharing interactions. In spite of prosperous performance in terms of data exchange and management in the proposed data sharing systems, the new world expects beyond that.

As a matter of fact, nowadays users can trust a system only if they are assured that their information is safe, thereby, cyber attackers cannot manipulate and misuse it. To earn users' trust, the research works of data sharing have found a recent direction towards security and privacy. However, meeting these two requirements is quite challenging. The reason is there are diverse types of cyber threats and preventing them one by one is a complicated solution. Therefore, finding a new technology that provides a package to improve cyber security has been craved for a long time.

Satoshi Nakamoto [1] brought blockchain technology to attention of public and a solid framework for the concept of cryptocurrencies, more specifically Bitcoin, was introduced. Interests to

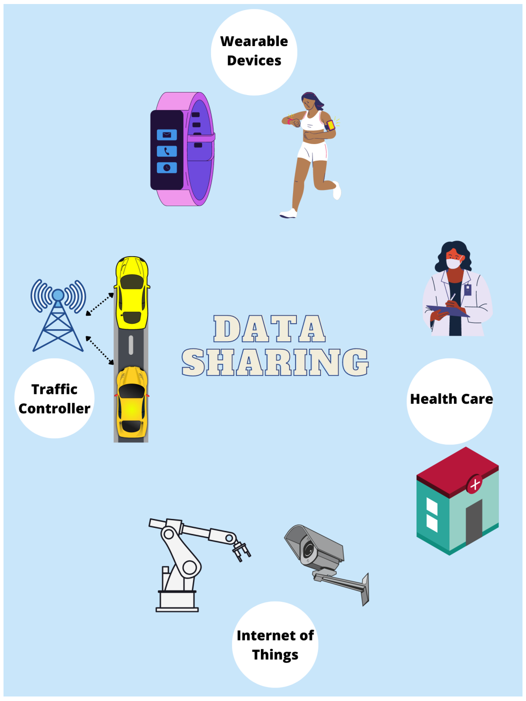


Figure 1.1: Data sharing and applications

blockchain experienced a large growth while due to deficiencies of Bitcoin, most importantly its massive power usage, many users could not afford engaging. Following that, Ethereum emerged by lower power requirements and also it established many regulations like constraints on the used gas, to encourage the users to apply power efficient programming algorithms [2]. In spite of decrease in power consumption, still high power dependency existed due to utilizing power-based consensus algorithm. Following that, blockchain became adopted into different network and cyber security applications. To come under close scrutiny, blockchain has several advantageous attributes where the most important ones are tamperproofness, decentralization and third party independence. Tamperproofness refers to the fact that data in blockchain are immutable. To find out the reason, firstly we have to investigate the architecture of blockchain network demonstrated in figure 1.2. Each block of blockchain is composed of three sections namely, hash, hash of previous block and data. Regarding the hash, it is a series of digits and letters generated based on the holding data of a block. Also, as figure 1.3 depicts, it is irreversible and unpredictable. The unpredictability stems from avalanche effect of hashing function wherein a slight change in input results in huge changes of hash. Besides, hash of previous block in block N of the network expects the same value compared to hash

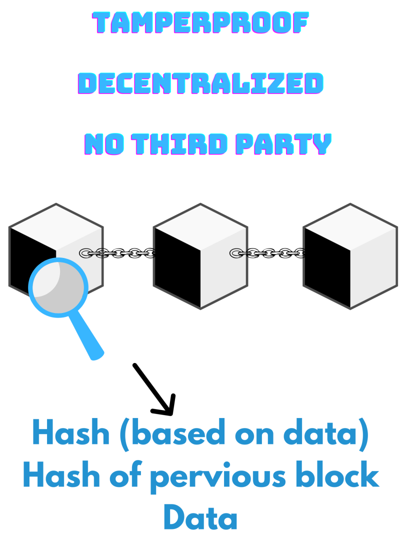


Figure 1.2: Blockchain network structure

of block *N-1*. If a malicious user manipulates the data of block *N-1*, that results in change of hash, the hash of block *N-1* and hash of previous block of block *N* are no longer the same and the network becomes notified. That is the reason why any data tampering and manipulation is detected in blockchain. Furthermore, decentralization of blockchain yields participants with diverse locations and also avoiding the types of attack that rely on centralization. The most important one is single point of failure in which the malicious users attack a point that leads to dysfunctional performance of the network. An efficient solution for security against single point of failure is relying on decentralized technologies. As for third party, blockchain is an autonomous network in which there is no need to be governed by a third party organization. As a matter of fact, presence of third party plummets the trust in a system especially in the ones dealing with sensitive information, e.g. medical data. Hence, blockchain self-governance is deemed valuable to assure security of information. Despite all aforementioned merits, blockchain brings drawbacks as well. As an illustration, due to hurdles imposed by consensus algorithm of blockchain, it requires to have powerful processors like GPU, ASIC, and even mining farm in large scale cases. In consequence of this problem, concerns over limited applications of blockchain and environmental repercussions rose. Besides, small electronic devices of IoT and wearable devices applications cannot be integrated into these types of data sharing networks.

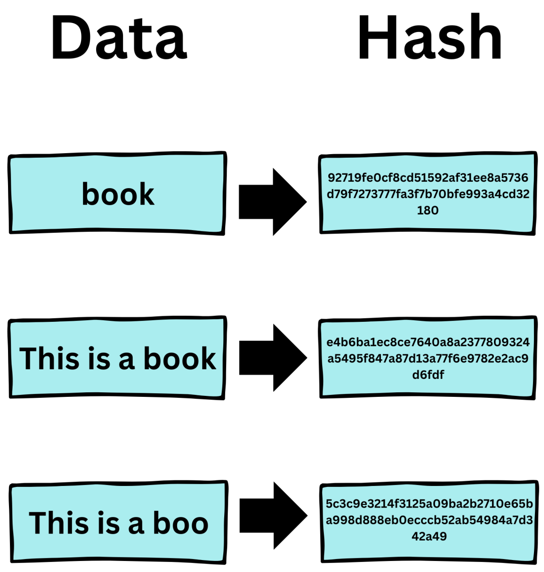

Data
Hash
book
92719fe0cf8cd51592af31ee8a5736d79f7273777fa3f7b70bfe993a4cd32180
This is a book
e4b6ba1ec8ce7640a8a2377809324a5495f847a87d13a77f6e9782e2ac9d6fdf
This is a boo
5c3c9e3214f3125a09ba2b2710e65ba998d888eb0ecccb52ab54984a7d342a49

Figure 1.3: Data and hash

As a matter of fact, the large power consumption requirement is originated from consensus algorithm of blockchain, that is proof of work (PoW) where the miners have to solve a complicated mathematical puzzle. To solve power usage, staked-based consensus algorithms were introduced. At the beginning, proof of stake (PoS) was unveiled that substituted mining of PoW with voting process [3]. Voting occurs based on stakes of users, the user with more stake has more influence in voting. Following PoS, delegated proof of stake was introduced [4]. It was able to decrease the communication costs and latency of PoS due to the fact that, only delegates could engage in voting while in PoS all users could do so. Despite power consumption reduction, stake-based algorithms accompany an unfair voting process [5] where a selfish user who has plenty of stakes may threaten the network. Also, rich users will get richer and richer since they always vote in favour of themselves.

Incentive mechanisms play a key role to encourage participation of the members. As figure 1.4 depicts, there are different solutions to achieve that goal, one popular method is based on monetary rewards, e.g. Bitcoin [1], wherein the users receive coins after participating in mining process. As to

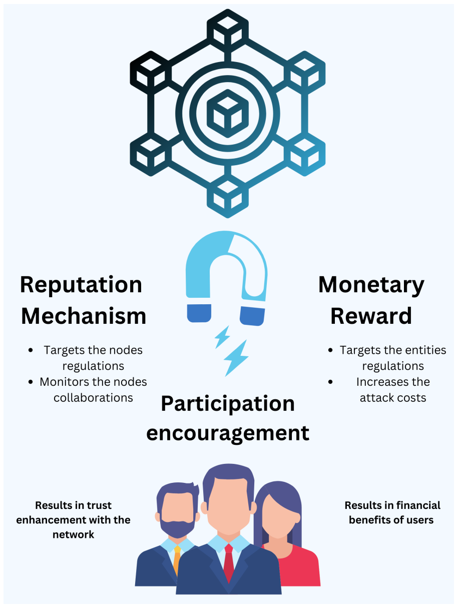

Reputation Mechanism
Targets the nodes regulations
Monitors the nodes collaborations
Monetary Reward
Targets the entities regulations
Increases the attack costs
Participation encouragement
Results in trust enhancement with the network
Results in financial benefits of users

Figure 1.4: Incentive mechanisms

other solutions, reputation mechanism is a beneficial method to reward the collaboration of the users as well. In this mechanism, the participants who act honestly are received incentive with reputation and the ones who are dishonest are penalized by reputation reduction. To compare money based and reputation based mechanisms, as Han et al. discuss in [6], monetary incentives aim to produce regulations for performance of system entities from an economic view. They grow the utility of entities when they engage in the system activities by rewarding them monetarily. Moreover, these sorts of incentive mechanisms target to surge the cost of attack. Compared to money based incentive mechanisms, reputation based ones peruse the aim of encouraging the nodes to collaborate. They confer reputation to regulate node behaviors to build an honest and cooperative network. Therefore, to enhance trust among the network users, reputation mechanism outperforms monetary one.

On the other hand, current reputation systems suffer from deficiencies as well. First and foremost, selfish users can manipulate system. They can gain reputation by forging unnecessary transactions where data owner and data requester are showing themselves as active collaboratives of the framework. As a result, higher possibility of attacks that require the malicious users to obtain reputation, e.g. sybil attack, appears in the system.

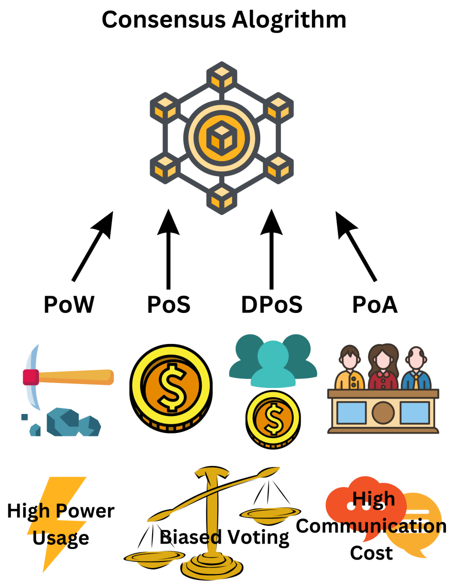


Figure 1.5: Consensus algorithms analysis

1.2 Objectives

This work aims to attain several different goals to supply a secure and efficient blockchain based data sharing network. As a matter of fact, blockchain is adopted into data sharing to enhance this framework while, as figure 1.5 shows, consensus algorithms of conventional blockchain designs impose either high power consumption or weighted voting or high communication costs. In terms of consensus algorithm, the objective is to propose a power and communication costs efficient yet secure algorithm able to manage the transactions.

Besides, our research work targets to employ a reputation system as the incentive mechanism to enhance the security and trust, and also reward the honest collaborators. This step firstly requires to propose a solution to solve the current problems of conventional reputation systems that are composed of susceptibility against manipulation of selfish users and malicious activities of attackers.

As the last goal, the work seeks a method to decrease the latency of current data sharing frameworks. Latency reduction is a requisite since the framework can become an efficient choice for different IoT and low latency applications.

1.3 Contribution

This work presents a new proposed data sharing framework that tackles the problems of high power usage, biased voting and high communication costs, with the introduced new consensus algorithm. Also, we provide incentive mechanism based suggestions to enhance the trust in the network of users. Moreover, to decrease latency and communication costs, the framework utilizes a cloud-centric encryption.

The principal portion of effort in this work is allocated to the implementation of our data sharing framework. In the development steps, we underwent various hurdles that are explained in this work. As a matter of fact, one essential reason that impacted the complexity of our implementation is we developed this framework by minimum number of built-in and external tools. In other words, we implemented a considerable portion of this work from scratch to reduce the necessitated communication costs between external tools, and also to increase the control over entire the network functions.

It is worth mentioning, this work has resulted in publication of the below research paper:

- Fotouhi et al. ”Trust-enhanced blockchain-enabled framework for secure and privacy-preserving data sharing systems”. In: *IEEE Future Networks World Forum* (2022)

1.4 Thesis Structure

The content of thesis is organized as chapter 2 represents the literature review. Followed by that, chapter 3 demonstrates the system model wherein the network design is elaborated. Besides, the mathematical expressions introduced in this work, are shown and explained within the chapter. Moreover, the data sharing framework working steps and also the consensus algorithm are elucidated by the attached algorithms.

Chapter 4 explains the development challenges. In this chapter the codes, utilized tools and involved programming frameworks are depicted. Chapter 5 represents the evaluation of the performance of proposed idea by the test results that are attached. Also it discusses the outcome of measurements to illuminate the reasons that the proposed idea confronted either outperformance or underperformance. As the last chapter, chapter 6 concludes this work and its contributions following that, it discusses the potential future works to enhance the performance of the proposed work.

Chapter 2

Literature Review

In [7], Zhang et al. propose a blockchain based framework for personal health information (PHI) that is reliable in terms of security and privacy. As to the introduced architecture, it is built upon two sorts of blockchain networks namely consortium blockchain and private blockchain. Private blockchain, in which higher control over the stored data is provided, protects PHI records. Also, consortium blockchain, stores the indexes of the PHI records. The system is composed of three entities: system manager, medical service providers (it is assumed the registered hospitals) and users (the patients). System manager monitors the interactions and all the doctors and users, are required to register to the system manager. This work is developed based on cryptographic keys concepts. Considering the users, each patient receives a token after registration. The user should show the token to the doctor, who is a client in the medical service provider (hospital). The token works as the evidence demonstrating the interaction of doctor with the specific patient and authorizes the doctor to generate a new PHI for the patient in the private blockchain. Regarding the process of transaction generation, when the doctor encrypts the PHI with patient's cryptographic keys, a new transaction is emerged. The new transaction would be sent to the private blockchain of the hospital.

Liang et al. in [8], introduced a blockchain based user-centric health data sharing that mostly focuses on wearable devices data and features the concept of channel formation. As to implementation, this work is deployed as a mobile application to depict the performance in practice. The data sharing framework is implemented on Hyperledger Fabric [9] permissioned blockchain. The proposed architecture is composed of six entities: user, wearable devices, healthcare provider, insurance company, cloud database and the blockchain network. To briefly explain each one, the user generates data by her wearable device then the health data are uploaded to the network by the device. Following that, the user is considered as the data owner who is responsible for grant, refusal and revocation of a request.

Respecting wearable device entity, it serves as the tool to produce the health data then it should transform original health data into a human readable format. Also, it should be connected to the network to dispatch the acquired information. Healthcare provider entity is comprised of doctors assigned by the users to render them access for conducting medical test and treatments. Health insurance company is the entity utilizing the network to make an appropriate health insurance plan based on user health condition and daily exercises. Cloud database is the storage to protect health information of users and provides a traceable data access. The last entity is blockchain network that carries health data of users and also contains data access policies for each piece of information dedicated by the wearable devices.

Fan et al. in [10] explained the proposed medical data sharing system to solve data management and sharing policy tasks for electronic medical records (EMRs). As a matter of fact, they indicate the system architecture relying on Certificate Authority (CA) to handle authority management parts and act as system administrator to suspend malicious users. Also, cryptographic keys generation is based on CA as well. Moreover, User layer is the section responsible for user's data management and also making the users able to encrypt their data for user privacy. As the last layer, it has Processing layer containing databases and accountable for reaching consensus and data processing. Finally, according to the demonstrated evaluation, the cryptographic key based solution for encryption has imposed considerable delay to provide service.

Xia et al. in [11] portrays a medical data sharing that is built on four layers. The layers are composed of user layer, data query layer, data structuring and provenance layer and lastly, data structuring and provenance layer. Besides, each layer includes multiple sub-layers. Xie el al. indicate the user layer holds diverse types of users who all want to access data for research purposes. The instances of users are healthcare organizations and universities. The second layer, data query layer, manages the data queries that can be the requests to access the data from existing database infrastructure. This layer interacts with data structuring and provenance layer to interpret and translate proposed actions between the third layer and outside entities like a request receiving from outside of the framework. In fact, the second layer is consisted of querying system and trigger. Querying system has two main roles firstly, it is tasked to process the requests and change them into a format understandable by the third layer. Secondly, it responds to the requesters for their requests.

As mentioned in [11], the third layer of the model architecture, manages the data requests and retrieves data from database layer. The layer controls the actions and sends them to the database to be indexed and stored. As a matter of fact, results of actions are kept in the blockchain network due to its immutability that provides a reliable future auditing for the data sharing system. As Xia et al. mention, this layer has more responsibilities as well that are performed by the subsections namely, authenticator, processing and consensus node, smart contracts, smart contract permissioned database and blockchain network. In terms of authenticator, it evaluates the integrity of requests that are sent to data owners. The actions are encrypted using the contract key generated by authenticator. Authenticator encrypts the data requests as well. Processing and consensus node handles data request form that might be further developed into information in the blockchain, and also it delivers the requested data and their corresponding smart contract to the requester. In respect of smart contracts, receiving a new action leads to activation of smart contracts. The smart contracts duty is to manage the required actions on the dispatched data and also alleviate the storing process of data. Moreover, smart contracts are able to revoke access to the data in case of identifying policy violation and this is the part that connects smart contracts to the smart contract permissioned database section. In fact, this section holds a storage that receives any report regarding rule violations, following that, it suggests further actions according to the desire of the data owner. The blockchain network of this work represents an immutable and chronically distributed data respecting actions on the data package delivery. The other task of blockchain is to hold a side-block for actions of reported data by the smart contract. As the last component of system model, the database layer hosts the medical records, therefore, it is only accessible to a certain authorized participants. Also, it retrieves the valid requested data upon received requests from the third layer.

Shen et al. in [12] introduced a decentralized network based on blockchain network and peer to peer storage network for healthcare data. In fact, the work has two types of nodes namely, super nodes and edge nodes. Super nodes are servers from large healthcare institutions and hospitals, and they usually have large storage and computing power. The super nodes are responsible for main infrastructures of data sharing network. The edge nodes are the servers from smaller health centers like clinics that are tasked to store the actual data of data owners. As to resources of super peers, they are made up of three parts: blockchain service, directory service and healthcare database. Blockchain server manages the blockchain network that verifies and audits the data integrity. The directory server holds the inventory of user for medical data, and manages the sessions of the data sharing framework. Also, healthcare database protects the actual medical data of patients. There exist two sorts of events on the blockchain namely: data genration event and session creation event. Considering data generation event, it is produced when a piece of medical information is contributed. It accompanies, patient public key, healthcare provider public key and signature, data digest and hash of the event on the blockchain network. On the other hand, session creation event is created once a patient accepts the access requests for her medical data. The event includes infromation like the start and end time for the data access, public key of patient and the requester as well as the signature of the patient.

To ehance the security in [12], the work widely uses cryptographic keys for different tasks. For initialization, a patient generates and store her public key and private key. In the first step, the user sends her public key to the communicating super peer for future message verification. Then the public key is signed by the certificate authority entity. The patients and the healthcare institutions exchange the public keys for inventory generation. Also, each time a patient can access an inventory by decrypting the encrypted inventory using her keys. As another application of the keys in this work, when a user chooses the data from her inventory for the sharing, the one makes a session with the data descriptions then encrypts them using her cryptographic keys. Besides, the patient uses her public and private keys to encrypt the content of the session separately as well. Upon receiving the data access request, the healthcare provider verifies the session status. If the session is available, it assesses the request by message signature. Meanwhile the requester can find and access the session and by her public and private keys, the ones can decrypt the session. If all occurred validations are positive, the requester can receive access to the sent data. As a result, the requester starts decrypting the data package and using the requested data. To sum up, the cryptographic keys in this work are employed in data encryption, data decryption and verification processes. Although, the cryptographic keys-based methods are widely utilized, they increase the latency and complexity of network.

In [13], a blockchain-based data sharing framework is introduced where it prioritizes the security of data. The system is built upon a private blockchain and it allows only the invited members to utilize the framework. The processes of adding new members to the network and making them able to request to receive data, require verification of their identity. The work is implemented by the virtue of the concept of cryptographic keys. Membership issuing keys, are produced and dispatched to the users to permit them joining the data sharing framework. Moreover, Membership private key is used to generate a data request that further might be developed into a new block. Transaction private key is employed to digitally sign a request and Transaction public key is for verification of signatures on the blocks.

As explained in [13], the framework is comprised of three entities, namely the user, system management and storage. The users are the ones who want to access or contribute data, like hospitals, institutions, etc. Also system management is composed of a multiple interconnected sections, including issuer, verifier and consensus nodes. As for issuer, it is responsible for authentication process of new requested users (to become a new member). Verifier has to evaluate the integrity of user's public and private keys when the goal is to add new blocks. Considering consensus nodes, they assess the proposed blocks. Xia et al. indicate when a user wants to join the permissioned members, the user should send request to the issuer. If the member is authorized, the issuing key will be sent to the user to commence utilizing this framework. Also, to request a new block, the user has to use the generated public key and private key that are used to sign the blocks. The consensus node is required to fetch the unprocessed blocks then assure their authenticity for security purposes. Lastly, the processed block is broadcast into the blockchain network by the consensus node.

Yu et al. in [14], illuminate a proposed data sharing framework to deal with Industrial IoT data. The prominent problems that are indicated to be solved are lack of secure storage, access control, tracing and revocation of adversaries. As a matter of fact, a secure and traceable blockchain based data sharing framework for Industrial IoT and smart factories is introduced. The system architecture composes domain, blockchain, system manage server, edge sever and cloud sever provider.

In fact, domains are management sections of each smart factory and they are consisted of users, devices, sensors and domain administrators. This entity produces Industrial IoT data. In this work, this section is assumed as a trusted role. As to blockchain network of this research work, it conducts identity authentication to guarantee the integrity of the registered users. Blockchain also stores all public keys related to system manager, domain manager and users. Besides, system management sever is responsible to manage smart factory services for the users. It also generates cryptographic keys for all the sections of the system including the users. As to edge server entity, with aim of producing policy matching and supply data services for the users, this entity nonpermanently stores data of factories. Respecting cloud service provider, it stores historical encrypted data of factories.

As explained by Yu et al., in the first phase the blockchain authenticates all the identities and stores the entire public keys. Followed by that, the system administrator calculates the system parameters and assigns the users their private keys. Meanwhile, domain administrator handles formulating domain security, privacy protection policies and encryption tasks. In case of meeting access policies plus having a user who is not found in the revocation list, the user receives intermediate decryption parameters from either edge server or cloud server to access data [14].

In [15], Li et al. build a blockchain based and privacy preserving data sharing to solve a specific sharing problem. The scenario is a user who wants to sell her IoT data to medical research institutions. As declared by Li et al., due to openness of IoT environment, adversaries may harm the security and privacy of users. To avoid threats, the work defines six criteria that are required to be met by the data sharing framework namely, anonymity, data authenticity, rewarding, behavior policy of data user, access control, nonframeability.

As for anonymity, the data of data owner includes health information and divulgence of identity of users violates user privacy and also yields unwillingness of them to further participations. By data authenticity, it refers to the fact that forging IoT data by attackers reaches the researchers to the wrong trained models and conclusions, thereby, data sharing framework requires to assess authenticity of data. As the next criterion, rewarding is a useful tool to increase collaborations between users. Also, behavior privacy of data user should be assured since the adversaries can construct behavior profile of data users based on pattern of user accesses to the information. In terms of access control, users need a trustworthy environment to set sales rules and access policies. Lastly, nonframeability means in case of malicious framing of data user, it may be a denial procedure to defend user's rights and the framed data user can submit a proof to trusted entities in this regard.

As for system model, Li et al. propose five entities: data owner, data user, management center, cloud server and blockchain [15]. Data owner and data user are the involved individuals on the both side of a data transaction. In this work, management center is the entity tasked to verify licence message issued by the data user. Besides, it provides the stealth address and access token for authenticated data users. Also, the paper explains, when a data user is framed management center investigates the case. Cloud server has considerable computational power and high storage level in which data is encrypted using cryptographic keys. Blockchain is the entity that protects privacy of transactions. Besides, blockchain assists to provide an anonymous area for the involved data owners and data users. Finally, data owners set sale rules and access policies using blockchain network.

Kumar et al. in [16], combined blockchain technology with federated learning to reach a deep learning model that is trained in a distributed manner to detect Covid-19 positive cases using their CT scans. In fact, Kumar et al. state urgency of quick Covid-19 diagnosis plus privacy preserving and global share of data among hospitals have led us to propose this approach. Blockchain technology is tasked to authenticate the contributed data and also federated learning focuses on secure global model training. To train the model properly, the input data have to be normalized to remove data heterogeneity. To normalize the image data, there are two stages: spatial normalization and signal normalization. Spatial normalization handles the dimension and resolution issues while signal normalization considers intensity of each voxel of CT scanners.

As to data sharing process in [16], due to high costs and infeasibility of storing data on blockchain, an off-chain method is employed. In other words, the CT scan data of each hospital is kept by themselves and blockchain intermediates in case of data retrieval. Besides, when a hospital provides the information, a transaction in the blockchain is appended to declare the data owner. This method of data sharing not only reduces blockchain network costs, but also improves data privacy wherein the actual data is held only by the data owner.

To reach consensus for data sharing and also perform collaborative model training, PoW concept is exploited. To enhance privacy, the whole received data is encrypted and singed by public and private keys. In this work instead of sharing the CT scan data, hospitals share their learned models with the requesters. Moreover, the transactions are composed of weights of a local trained model proposed by a hospital and also these local models are aggregated by the network to create a global model as well.

In [17], Shayan et al. elucidate a blockchain and federated learning based system for privacy preserving communication between clients. The main goals followed by this framework are to converge to the optimal global model, prevent manipulation and avoid attacks like poisoning attack. In this work, stochastic gradient descent (SGD) is the optimization algorithm and within each iteration, clients locally compute SGD also for data privacy reasons the users mask their SGD updates. It should be noted, the updates are signed by the creator then the system assesses their signatures using their public key. As the next step, the masked updates are verified by a verification committee to avoid poisoning attack. All peers who have contributed to the applied updates of global model, are rewarded with stakes. Also, the aggregated updates form the new block that appends to the blockchain network.

The consensus algorithm of this work is proof of federation (PoF). As Shayan explains in [17], PoF is quite similar to PoS. In the introduced PoF algorithm, stake reflects the contribution of a user to the network. In fact, clients may increase their stake by contributing advantageous model updates to the system. Respecting poisoning attack prevention, the SGD updates are evaluated. Verifiers perform Multi-Krum method on the received updates to filter out malicious ones. As a matter of fact, the verifier assures the newly received update is consistent with previous ones. By this method unusual updates can be removed.

Lu et al. in [18], propose a data sharing architecture consisting of two modules: permissioned blockchain and federated learning module. In the blockchain network, there exist two kinds of transactions namely, retrieval transactions and data sharing transactions. Besides, federated learning is a distributed version of machine learning algorithms in which a global model is trained and improved using the local models contributed by each node. One of the main reasons of integrating federated learning into this data sharing framework is for consensus algorithm efficiency. The introduced consensus algorithm of this work is proof of training quality (PoQ) wherein the power consumption of consensus algorithm unlike, e.g. Bitcoin [1], is tried to become useful. In other words, the power usage from each node is employed to train the learning models. Despite steering the power usage to become beneficial, it is still deemed a power consumption dependent consensus algorithm. Moreover, the committee node members are tasked to collect the local models from their nearby users to process the consensus and assist feeding the global model to learn. Also, the trained models can be accessible to the users in response to their data sharing request.

As Lu et al. mention [18], when a data provider joins the framework, the one is ought to send her public key and data profile to the nearby super node for verification. As to security tools, the participants have to encrypt the messages and the encryption is occurred using cryptographic keys. Subsequently, the encrypted transactions are received by the committee nodes. Each one is responsible to collect all the received model transactions and evaluate their training quality using mean absolute error (MAE). The committee node with the lowest MAE will be selected as the committee leader. The committee leader task is to propose a new block, then the block is either granted or declined based on the votes of the committee nodes.

To assess the integrity of transactions in the blockchain network, there exist different consensus algorithms [19]. Azaria et al. in [20] proposed a medical data sharing framework however, it was based on PoW. In other words, it suffered from power consumption issue as a result, this became an infeasible option for many network especially low power devices. In addition, the framework utilizes Ethereum blockchain and its smart contracts to manage the permissions and data retrieval from database. Also, to incentivize the users to render their computational resources for PoW, the system has set policies. In fact, transactions require Ether to be processed by the network and the solution to earn Ehter is mining, wherein the users has to solve a mathematical puzzle by expending their power resources. Moreover, the patients who want to contribute data to the network, are required to pay the costs. As a result, they have to gain Ether using mining. Regarding the transactions, the data requesters, e.g. researchers, involved in this work can freely choose which transaction to validate and mine. Azaria et al. believe leveraging PoW mining and a monetary incentive mechanism yields a data sharing framework in which data economics, supply and matching of data between data owners and requesters are valued.

With aim of solving power problem, Tosh et al. [21], proposed a PoS based framework that exploits votes of users to validate the transactions. As explained by the authors, in PoS a user's ability to broadcast a new block to the blockchain is proportional to the rate of stakes that the one holds. In this work, there are several fully connected virtual nodes in the cloud computing architecture that are called validators to engage in the consensus process. As to the consensus algorithm procedure, the validators receive raw transactions and they produce their own proposing block. Then the leader is elected to broadcast a new block to the blockchain network. Following this stage, the validators verify the authenticity of the proposed block. As to leader selection, the chance of a validator to become a leader is related to the possessed stake. Besides, the structure of blockchain in this work has slight differences. Apart from the attributes that a conventional block has, in this work a block includes the stake of the leader who has proposed the block that is utilized in block verification process by the validators. To incentivize the users to participate, the staple part of reward in each round belongs to the leader and the remaining is split among the validators.

To decrease the generated latency of voting of all users, [22] and [23] introduced a system featuring DPoS consensus algorithm. In fact, in [22], a privacy preserving data sharing for healthcare records is shown. In this work, the medical records are stored in the cloud and the indexes of them can be found in the tamper-proof consortium blockchain. Also, smart contracts are employed to control user access policies. This work features DPoS consensus algorithm in which the nodes on the blockchain network choose 101 delegates. As stated by Lie et al., DPoS in comparison with PoW and PoS is quicker, more decentralized and more efficient in terms of power consumption. It is noteworthy, the election solution used in conventional DPoS [24] has modified in this work to enhance the reliability. In the proposed node election method, the selection is based on the rank of institutions' credit score. As elaborated by the authors, the top 30 institutions are designated as the representative users to generate blocks. Besides, the following 20 institutions are designated as the audit users to investigate the blocks. Considering rewarding, the nodes that contribute to the data sharing framework gain reward for their collaboration. If a representative loses signing a block or the nodes tasked to audit the blocks, falsely audit the information, it leads to lose of credit score and as the credit score of a node comes below the required rate, it might lose the assigned role, e.g. representative, and be replaced by another high score node.

In [23], a lightweight medical sharing system is proposed. It utilizes proxy re-encryption to assist the doctors accessing healthcare records of users. Moreover, it improves security due to query of encrypted version of health data using cryptographic keys. Liu et al. explain, the consensus algorithm of this work is DPoS. We could chose the conventional DPoS in which there are several chosen nodes to vote for validation in behalf of network users. In terms of delegates, they all have equal rights compared to each other and they are tasked to broadcast new blocks to the network. Also, when a delegate misses to participate and collaborate, the system will choose a substitute for the delegate and the one will be penalized. Liu et al. continue by mentioning, the conventional DPoS is not efficient in regards to communication costs of voting. Therefore, this work introduces an improved version of DPoS where, without voting, every doctor is known as a delegate who is tasked to generate the new blocks. Besides, the server of hospital is deemed as the verifier. The system manager section and server of hospital may monitor the performance of doctors to assure their quality of performance and when they are found to be incompetent, they lose stake. In case of reaching the stake of a doctor below the threshold, the doctor loses the delegate role. On the other hand, hospitals are monitored by the system manager and doctors as well and their dishonest behavior may lead to issue of penalty.

Proof of authority (PoA) was the next proposed solution to remove power consumption and unfair influence of high stakeholders. As De Angelis et al. explain in [25], PoA is a new branch developed from byzantine fault tolerance (BFT) consensus algorithms and it is adopted by two well-known Ethereum clients, Parity [26] and Geth [27]. The main difference of PoA compared to the popular algorithm of BFT family, practical byzantine fault tolerance (PBFT), is less message exchanging that leads to enhancement in performance. In fact, PoA initially was introduced to act as the consensus algorithm of private blockchain in Ethereum. Therefore, PoA appeared as two versions on Aura and Clique. Principally, PoA requires N trusted nodes called authorities. The authorities perform a consensus to handle the transactions. Also, there is a mining rotation schema, that leads to divide the responsibility of block generation among the authorities. PoA algorithms assume the majority of authorities are honest.

As De Angelis et al. elucidate, in Aura the authorities follow two queues, a queue for transactions and the other for blocks. When the leader is selected, the one should broadcast the pending transactions to the newly proposed block. Following that, the authorities send the received block to each other, if all the authorities receive the same block, the proposed block is granted by them. In case of not receiving the same block, the authorities commence voting to expel the leader.

As for Clique, it virtually follows the process of Aura while the most notable difference is, alongside the leader, some authorities are permitted to propose a new block as well. Similar to Aura, in case of dishonest activities of either the authorities or leader, they can be voted to be expelled. At each round, there are at most $N - (N/2 + 1)$ authorities allowed to propose where N is the number of authorities. Due to higher number of potential block proposers in Clique, the possibility of forks increases.

Furthermore, user participation is another problem required to be considered where reputation systems are deemed an intriguing choice to increase participation of users. Xie et al. in both [28] and [29], proposed a reputation mechanism to alleviate the interaction challenges of requesters and workers for crowdsourcing systems. As mentioned in the paper, three principal components of crowdsourcing systems are tasks, workers and requesters. The requesters try to find appropriate workers to assign them their tasks and benefit them with rewards after task completion. The main objective of this research work is to find the minimum rate of reward for a certain task. This rate might be considered too critical to be set since large amount attracts the workers while repels the requesters and vice versa happens in case of too low reward rate.

The outline of proposed idea is a requester posts a task with its corresponding reward and transaction fee. The information should be received by the crowdsourcing system administrator. The posted task is assumed to be received by n workers. When a task is solved by a worker, the user should share the solution with the system administrator. As all n solutions are submitted, the task is considered finished then the requester evaluates their methods and chooses the one with highest quality. After that the requester notifies the system administrator and the winner will be announced amongst the workers to be rewarded.

Alswailim et al. in [30] described a method applied in participatory sensing applications [31]. As a matter of fact, in participatory applications the users collaborate by sensing, collecting and sending the data to an application server. Due to openness of these applications, they may be fed with inaccurate data and consequently the application performance becomes downgraded. Alswailim et al. introduce a method, based on reputation systems, to evaluate the contributions of users. The solution requires the participants to collect and send their data using their sensor devices. Then the system weighs the received data packages and sends the most accurate ones to the application server.

As to details of aforementioned process, there are two major stages. In the first step, the users are divided into three groups, following that, the system assesses each group by calculating their corresponding values using the reputation of each user and the contributions of the group with highest value are dispatched to the application server. In the second phase, the participants score is updated based on the accuracy of their data where the group with highest values receive positive scores and rest of the groups are assigned a score reduction.

[32] is the research work conducted by He et al. that concentrates on ad-hoc networks. In fact, ad-hoc networks can be dynamically created by the mobile users in various locations. As to network restriction, ad-hoc networks users transmission range can be limited because of their power usage. Due to power limits, a selfish user may intend to circumvent the requirements and stop consuming resources to forward packets. To overcome similar hurdles, He et al. suggest using a reputation based solution. The authors explain, in the proposed method the neighbor nodes are required to share the reputation information of other nodes, in this way a node can construct a record of the reputation of its neighbor nodes. As to reputation rate, reputation of a node is calculated according to the packet forwarding ratio of the node. In this work, reputation propagation is perceived an efficient method to detect the selfish users and restrict the incidents occurred by them, and also the recognized selfish users encounter penalty by the network as well.

Despite all the indicated efforts, there are some identified concerns needed to answered. As [33] and [34] explain, the current reputation systems suffer from the possibility of being manipulated. As a matter of fact, selfish users can generate unnecessary transactions to show themselves collaborative and active in the network and as a result, gaining reputation. Secondly, susceptibility against sybil attack and lastly, vulnerability against whitewashing attack that can lead to data leakage, privacy violation and dysfunction of the network.

Chapter 3

Data Sharing System

In this chapter, our data sharing system is illustrated and the contributions of proposed idea are discussed.

3.1 System architecture

As to network design, the proposed data sharing framework inputs can either be contributed data from data owners or request to access data from data requesters. In case of data request to access data, the output is the requested data by considering security and privacy constraints.

As Figure 3.1 depicts, there exist two managers that are programmed to be responsible for evaluation of transaction integrity, and assessment of users behind the transaction. Plus, they manage data storage and encryption tasks. To come under close scrutiny, the transaction manager is required to receive data contributions and data requests and following that, it assesses the honesty of them. As to detailed explanation, the transaction manager is composed of several subsections namely, the pool of transactions, the vote handler, the authorities and the leader.

When a requester issues a data request, the transaction manager receives it. In the next stage, the transaction is sent to the pool of transactions which is a repository to store the transactions and their corresponding data sender (owner) and data receiver (requester). The pool regularly dispatches the transactions to the vote handler. As a matter of fact, the vote handler manages the process of transaction assessment wherein a transaction is distributed among the authorities then they send back

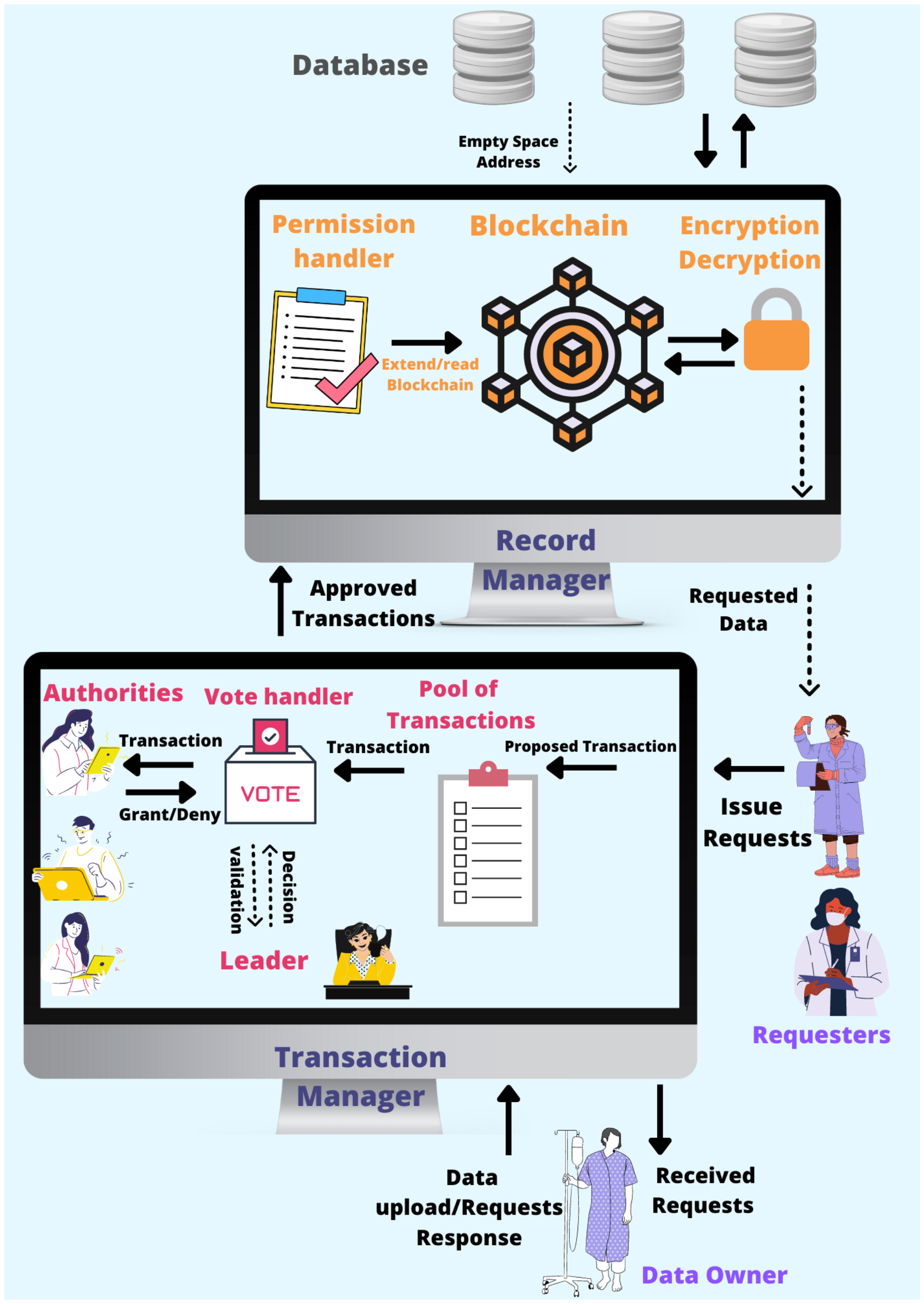


Figure 3.1: System Model

their vote to the vote handler and based on decision of majority, a transaction might be granted or annulled. It is noteworthy, the authorities are a group of selected users and their duty is to evaluate a transaction integrity. Besides, the leader steps in when the number of authorities is even and they equally vote for accepting and rejecting. In that case, the leader takes the transaction into the consideration and declares the vote. The granted transaction is sent to the record manager; however, the rejected one takes no further step. Lastly, in case of data contribution instead of data access request, the steps are virtually the same except for the transaction wherein the sender is proclaimed the data owner and the receiver is data sharing framework.

In terms of the record manager, as demonstrated in algorithm 1, the subsections comprise the permission handler, blockchain network and the encryption/decryption. The main responsibility of the record manager is to protect data access and also authorizing the users to safely store and retrieve the data utilizing techniques like data encryption. To investigate closely, the approved transaction is received by the permission handler that has the responsibility of authorizing the data requester to access the data. The aim is attained by updating the permission list of requester. As a matter of fact, each user has a permission list in her profile Pr_{user} that contains the permitted data ids to that users. As for data id, each data package has a data id in the database and it acts as a tool to identify the particular required data. The next phase is to send the data id from the permission handler to the blockchain network. The blockchain keeps data ids and their corresponding encrypted address of them in the database. The received data id, is inquired by the blockchain network and its corresponding address will be found. Due to encryption of the acquired address, it is decrypted by encryption/decryption subsection and finally based on the address, the requested data from database is retrieved to the requester.

Also, the procedure is approximately similar in case of data contribution in which the permission handler makes the contributed data accessible to the data owner and after that, the blockchain would be extended by storing new information regarding the data id and its encrypted address in the database.

Algorithm 1 Data sharing algorithm

```
    Initialization: Selection process of authorities and leader
    Input: $Pool$: pool of transactions; $Blockchain$: blockchain network; $action$ where

        $action \in \{DataRequest, DataContribution\}$

  if $action = DataRequest$ then
      $Req \leftarrow$ Requested data type
      while A data owner is not found do
          if A data owner is found then
              $id \leftarrow$ data id
              Add the transaction to $Pool$
          end if
      end while
  else if $action = DataContribution$ then
      $TempMemory \leftarrow$ the data from owner
      $id \leftarrow$ data id
      Add the transaction to $Pool$
  end if
  $Result \leftarrow$ Consensus algorithm
  if $Result = AcceptTransaction$ then
      if $action = DataContribution$ then
          Confer reputation to authorities and leader
          $C_o \leftarrow$ Estimate $C$ for the owner
          Confer $C_o$
          Add $id$ to permission list of the owner
          $Address \leftarrow$ Empty space address in database
          $EncAddress \leftarrow$ Encrypt $Address$
          $Blockchain \leftarrow id$, $EncAddress$
          $EncData \leftarrow$ Encrypt $TempMemory$
          Upload $EncData$ to database
      else
          Confer reputation to authorities and leader
          $C_o \leftarrow$ Estimate $C$ for the owner
          Confer $C_o$
          $C_r \leftarrow$ Estimate $C$ for the requester
          Confer $C_r$
          Add $id$ to permission list of the requester
          $Address \leftarrow$ Find address by $id$ in $Blockchain$
          $DecAddress \leftarrow$ Decrypt $Address$
          $Data \leftarrow$ Retrieve data by $DecAddress$
          $DecData \leftarrow$ Decrypt $Data$
          return $DecData$
      end if
  else
      Confer reputation to authorities and leader and delete the transaction from $Pool$
  end if
```

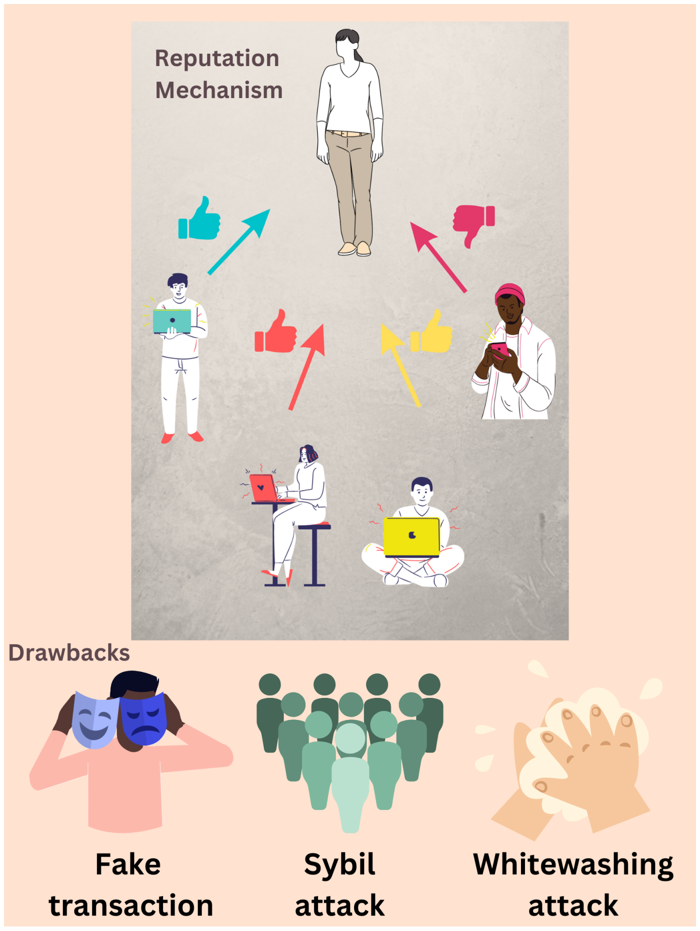


Figure 3.2: Reputation systems drawbacks

3.2 Incentive mechanism

Data sharing framework requires to attract users and increase participation. To encourage the users to engage, based on the employed network architecture, blockchain inspects different methods to value their participation [33]. As to solutions, the users can be rewarded by reputation and that is where reputation mechanisms emerge. As a matter of fact, the higher number of collaborations, results in the higher reputation for the user. In spite of advantages, as shown in figure 3.2, there are several unsolved issues in reputation mechanisms that repel the network designers to include reputation mechanism in their proposed ideas.

One of the most common methods to manipulate reputation mechanism is fake collaborations between a selfish data owner and a selfish data requester where they generate plenty of unnecessary transactions within a short time span. As a result, the framework recognizes them as active and collaborative users who will be rewarded consequently. Besides, in these sort of systems, the possibility of attacks like sybil attack and whitewashing attack is enhanced as well.

To prevent reputation system manipulations, we propose a new idea to replace the conventional one. In fact, we adopt conferred reputation given as:

$$C = \lceil R\frac{1}{n_c}\rceil, \tag{1}$$

where R is reputation rate and $\frac{1}{n_c}$ is protection factor (PF). In fact, n_c is the number of collaborations between a particular data owner and data requester within critical period P_c. To elucidate PF concept, firstly the network should be assigned with a rate of P_c. The optimal value can be obtained via investigation of data traffic of network and its value may vary based on different data sharing applications. As a matter of fact, IoT devices controlling a factory performance might produce transactions every millisecond; whereas, a small group of researchers might utilize a data sharing network to exchange data each using longer time steps.

According to equation 4, when a particular data owner and data requester produce a transaction within a short period of time (P_c), the conferred reputation equals R, since the number of transactions (n_c) is 1. While as the number of transactions commences to grow massively within the P_c, the framework plummets the conferred reputation due to detected unusual behavior.

3.3 Consensus algorithm

As mentioned in chapter 1, despite merits of blockchain network, for a long time it has been suffering from deficiencies imposed by its consensus algorithms. The earlier blockchain networks suggest PoW and requiring the participants to compete for solving a complex mathematical puzzle. Intolerable energy consumption of PoW led to establishment of the next generation of algorithms, that were stake-based ones where the users with higher stakes had more power to influence the decisions. Due to biased voting process and unfair network for users, different algorithms originated from PoA became popular. Nevertheless, there are two main disadvantages in exploiting PoA.

Firstly, the authorities generate high communication costs for the system. The issue is exacerbated in case of larger networks that are composed of high number of transactions and more number of authority members. Moreover, as De Angelis et al. describes in [25], the authorities only verify to receive the same proposed block from the leader or the authority who is allowed to propose new block. Therefore, more information like the reputation of data owner and data receiver of this transaction is not scrutinized. Hence, to solve aforementioned problems we propose a new PoA based algorithm, called smart PoA.

Smart PoA can mitigate communication costs issue and also reputation of users involved in the transaction impacts the judgment of the authorities. Respecting the authorities and the leader, they are chosen based on their reputation. In smart PoA, the leader is obliged to monitor the performance of authorities who are a group of validators to manage transactions. As the authorities receive the transaction to assess it, they can access records representing the reputation of data owner and data requester. These records are formulated as trust factors (TFs). As a matter of fact, one prominent superiority of smart PoA over conventional PoA is TFs. TFs demonstrate the level of security of a transaction. TFs are consisted of three factors namely, transaction trust factor (TTF), owner trust factor (OTF) and requester trust factor (RTF). The equations of TTF, OTF and RTF are given as:

$$OTF = \begin{cases} 1 & \text{if } N_{otrx} = 0 \\ \frac{Rep_{own}}{(RN_{otrx})} & \text{otherwise,} \end{cases} \tag{2}$$

$$RTF = \begin{cases} 1 & \text{if } N_{rtrx} = 0 \\ \frac{Rep_{req}}{(RN_{rtrx})} & \text{otherwise,} \end{cases} \tag{3}$$

$$TTF = OTF + RTF, \tag{4}$$

where Rep_{own} is the reputation of data owner, Rep_{req} is the reputation of data requester, N_{otrx} is the number of all transactions that data owner has been involved and also N_{rtrx} is the same for the data requester. By all transactions it refers to all participated approved/disapproved transactions plus all validated transactions either served as a leader or authority.

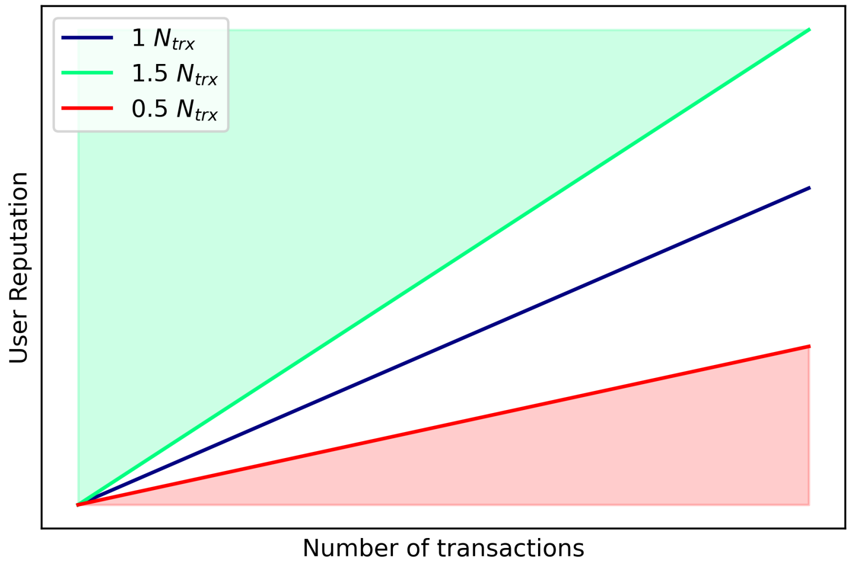


Figure 3.3: OTF and RTF rate

In our proposed scheme, as figure 3.3 shows, the common rate for OTF and RTF is estimated to be 0.5 to 1.5, below this range it warns authorities about low reputation and above 1.5 it shows high level of trustworthiness of a user. Also, TTF is calculated based on OTF and RTF. The safe rate of TTF is between 1 to 3. TTF renders the power to the leader to control the performance authorities. As shown in algorithm 2, the authorities judge a transaction based on TFs and in the case that TTF of a transaction is below 1 and it is accepted by a certain authority the transaction and vote of the authority is sent to the leader to evaluate the performance of authority. For instance, a negative vote to a transaction by TTF of around 1 is more acceptable to a leader compared to TTF of around 0.1. If the authority is deemed dishonest by the leader, the authority will be penalized by losing reputation. On the other hand, the authority and leader are rewarded by gaining reputation for their participation.

Algorithm 2 Smart PoA algorithm

```
  Initialization: Retrieve a transaction from pool of transactions
  Input: $N_{auth}$: the number of authorities
$RTF \leftarrow$ Estimate RTF
$OTF \leftarrow$ Estimate OTF
$TTF \leftarrow$ Estimate TTF
$TFs \leftarrow [RTF, OTF, TTF]$
while $VotesNumber/2 < N_{auth}$ do
    Vote collection
end while
if $ApprVotes = DisapprVotes$ then
    The leader decides
else if $TTF < 1$ then
    $List \leftarrow$ [Voters who approved]
    Send $List$ and $TFs$ to the leader
    if $AnyPenalty$ then
        Reduce reputation from $List$ members
    end if
else if $TTF > 3$ then
    $List \leftarrow$ [Voters who disapproved]
    Send $List$ and $TFs$ to the leader
    if $AnyPenalty$ then
        Reduce reputation from $List$ members
    end if
end if
return $Result$
```

3.4 Cloud-based encryption

In our proposed framework, we leverage an off-chain blockchain network due to communication and repository costs. As to transparency of blockchain, on-chain network may also leak some information while they could be protected in a secure off-chain network. In spite of higher security of off-chain, it requires precautionary security measures as well to protect the data stored in database. Moreover, as Gordon et al. indicate in [35], storing a considerable volume of data in an on-chain network is impossible in practice.

In spite of common public and private key encryption methods, our encryption solution is not on local side and user is not involved in encryption and decryption tasks, therefore, the whole process is straightforward, automated and on-cloud. In fact, to decrease the communication costs and complexity of the data sharing framework, a cloud-based encryption method outperforms. In our scheme, the key tools to allow us safely change a local-based method to the cloud-based one, are permission list and data id concepts that are replaced to cryptographic keys concept.

First acquired merit is about insertion of keys and validations of them where either done by data owner, requester or entity (e.g. hospital) is considered local-based computation, and usually local-based computation is less reliable compared to cloud-based one, due to weaker computational power and more latency. Furthermore, in terms of complexity, as we increase complexities like need to insert the public or private key or need to validate keys of data requester, less devices would be able to work in this sort of data sharing system. Due to the fact that they are usually only a simple electronic device (like a surveillance camera) and are not able to assess the keys or similar tasks. Equipped IoT devices with a technology to validate, send and receive of keys leads to increase in costs of production of IoT devices, thereby, it decreases the motivation to join the network. Plus, users of key based networks, are usually required to save different keys of their counterparts and involved entities (e.g. health care providers) and we believe this raises storage issues for small electronic devices in large networks with high number of users.

Chapter 4

Implementation

This chapter represents the implementation of our work. We implemented the proposed data sharing framework to demonstrate its feasibility and quality. As a matter of fact, we schemed the framework as a web application in which it needs to handle both back-end and front-end duties. Last but not least, the selected development codes are illustrated in appendix A, and also figure 4.1 summarizes the development tools involved in this implementation.

4.1 Back-end

Considering our development, the back-end is responsible for receiving the requests of users, e.g. GET and POST. Moreover, it has the duties in regards to receiving the data of data owners, protect the privacy of users by different techniques, like encryption of data, also back-end is required to manage the data requests and block dishonest users who aim to access data maliciously.

As to developing tools, the back-end of this web application is written all in Python programming language. Although, recently blockchain developers have shown their interests in programming languages like Solidity that are specifically built to serve blockchain applications, we preferred Python for several reasons. First and foremost, Python is a general purpose language and many applications, e.g. Artificial Intelligence (AI) and web ones, can be easily integrated in our blockchain-based data sharing code without leading to too complex environment. Nonetheless, for a Solidity based blockchain code, the program is required to be bound to the web or AI application written in

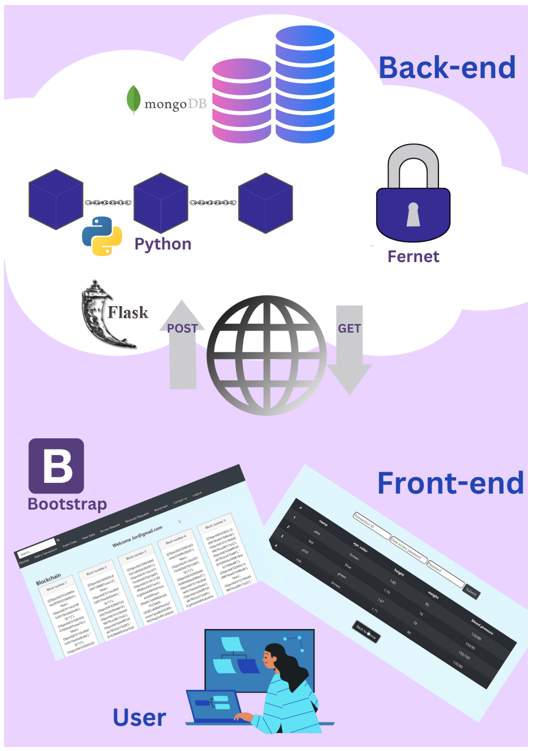


Figure 4.1: Implementation scheme

another language. As a result of this action, the appended code imposes latency since the connection of those codes yields communications cost. As another reason, even though Solidity provides some built-in libraries and modules to make the programming easier, it has many programming restrictions. Conversely, Python is a dynamic programming language and recent interests of programmers towards dynamic programming languages stems from the flexibility and the wide control that they provide over diverse aspects of the program [36].

The blockchain network is built by Python as part of back-end as well. For implementation we followed the fundamental concepts of blockchain in which a block contains hash, hash of previous block and data. Additionally, the web application of the data sharing framework is based on Flask that is a Python web framework. By virtue of Flask, we became able to receive GET requests of users and in response, we showed the corresponding web page. Also, after receiving POST requests, the data from user was received and dispatch to a proper address in the database.

Respecting encryption, we utilized Fernet that is a Python library to implement symmetric authenticated cryptography [37]. The reason to use symmetric authenticated cryptography concept is these sorts of cryptographic algorithms employ a unique key for encryption of plaintext and decryption of ciphertext. As for importance of the key, the encrypted data are impossible to be manipulated or accessed without having the key. Therefore, to avoid a successful attack by eavesdroppers, we minimized the number of times that the encryption key is called or transferred between files and functions. Also, all computational load of our developed solution is on cloud where this web application is running and the user does not involve in either encryption or decryption process. It should be noted, the equations shown in 1-4 are not implemented in this version of the framework.

In terms of the authentication of program, we exploited the concept of token. In fact, each user should enter the username and password, after that the program assigns the user with a token that automatically stays with the user as long as the one is logged in. Regarding the structure of the back-end of application, it is built on two Python files namely app.py and blockchain.py. The staple responsibilities are managed by app.py while the blockchain network is handled by blockchain.py file.

As to app.py, it is made up of several peripheral functions, routing functions and python decorators. For initialization of the program, we connected the application to our database and defined the used database collections. Based on the type of data, the database steers the information to the corresponding collection for instance, a transaction is dispatched to the pool collection where it receives the transaction id, transaction sender, transaction receiver and the data. One of the main challenges of development of this program was to make users able to be serviced simultaneously. At the beginning, this program was deployed as a desktop application and the initial architecture of back-end had restrictions consequently, the users could not utilize the system at the same time. The initial design was based on a global variable tasked to introduce the logged in user to the entire application. This solution is efficient and competent in case of a desktop application. To increase the accessibility of users to the data sharing framework we decided to change the program from a desktop application to a web application. Most of the differences made by this change were responded in a reasonable effort while simultaneous servicing faced a serious challenge due to application architecture. To explain the challenge in detail, when user A had logged to the web application, the global variable required the application to show the content relevant to the user A. At the same time, when user B logged in, the global variable asked the application to present the content related to user B. In this scenario a collusion used to happen in which when user A refreshed the pages, the content for user B is shown. In other words, the global variable lost user A and only holds the last person who has logged in. Hence, when a user requested the application to show a content, the program used to present the content related to the last person who is authenticated. To solve the problem we employed JSON web tokens (JWT) concept in which the server receives the user information and stores them along with a digital signature in the format of JSON. Verification of the signature is the tool of server to avoid any attack or impersonation of identities [38]. In our implementation, SHA256 algorithm is employed for signature generation. JWT is developed within token_required function. The function verifies the inserted username and password, and the authenticated user receives a token for accessibility to different part of program. A python decorator is placed before prominent functions that show a personal content to assure the identity of the person. The reason to use decorators is they can find the assigned token and verify it without needing the user to insert username and password multiple times. Besides, for security reasons, the lifetime of token is determined to 15 minutes and after that the user is logged out of the account.

There are many peripheral and short functions helping the processes like data management and authentication. To briefly indicate some of them, login_check function receives the username and password of the user and using the database assures the integrity of authorization request. The other important peripheral functions are accept_validate_trans and decline_validate_trans. The former one is called when a transaction is granted by an authority while the other one is responsible for rejected ones. The involved procedures of these functions include to firstly find the transaction in the database then ensure the authority has not already voted for this transaction. Receive of the authority's vote yields reward for the voter. Lastly, the functions verify whether more than half of the votes are on one side (yes or no) or not. If so, the voting is declared finished and the transaction is sent to the further steps or considered annulled.

Furthermore, add_to_blockchain function connects app.py to blockchain.py. In case of block generation, it receives the hash of previous block and also the data to be included in the block, then the new block becomes generated. It is worth mentioning, the information in the block is encrypted using encryption function. In fact, encryption function and decryption function are responsible for implementation of Fernet cryptography algorithm using the encryption key defined in the code initialization. Also, render_chain is another prominent function that shows the current blockchain network. The notable point about this function is it does not present similar information to all users. In fact, its input is the user identity, then it encrypts all sensitive information except the one that the user is involved in, in other words, the ones that the person is deemed authorized. As to extract_csv function, it receives the csv file then it detects the type inserted data and their corresponding information in each row. Following that, the data is stored in the database and the file no longer is kept. This method opposes the conventional solution in which the actual csv file is stored not the data in the file. We believe storing the actual file leads to higher storage requirement. Lastly, user2token function converts the token of authorized user to her username. In many cases that the program requires the username of the person, this function is called, e.g. issue of a transaction.

Followed by peripheral functions, routing functions are deployed. In fact, app.route() is our solution to to bind a URL to a function. At the ending part of each routing function, we aim to render a web page containing the data and calculation results of its above lines. For this purpose, we used Jinja template engine. The reason to use Jinja is it simply receives the targeted HTML page and the corresponding variables then renders the page. The most sophisticated function route is the /home URL and the reason is home page has a wide range of applications and connections to other pages. All the routing functions except sign up, login and logout URLs include token_required decorator and it is responsible to send back the token to the token_required function at the beginning of the code, for authorization purposes.

To shortly elucidate the duties of imperative routes, /insert_data is tasked to receive the posted data of users and collect the file information using extract_csv peripheral function. In our current implementation, we only receive csv file. The reason to put the restriction is we need to extract the information and we cannot apply a single particular extraction method for all formats of file. To avoid crashing while processing the data, we check the format of file beforehand. Only in case of csv file the route function goes further, otherwise it asks the user to provide a proper file format. Moreover, /add_transactions is the place where a user can generate a transaction. The user who is the transaction creator fills out the forms then posts the data, following that, this route function updates the database correspondingly. The other crucial route is /validation wherein it renders the list of transactions in the pool and authorities can vote to either refusal or acceptance. After each vote, the route updates pool immediately. For /blk, it receives the user identity and based on that shows the specifically decrypted blockchain network to the user.

To make data requests, /access_request route requires the requester to declare a particular transaction then the request is stored in the Access_Request_Pool that is the relevant database collection to keep the requests. On the other hand, /received_request route is responsible to show the data owners the received requests. Also, it collects the their responses to data requests as well and finally updates Access_Request_Pool in the database. To access the data, the approved requester can use /view_data route. In fact, the requester enters the data address and other corresponding information to view the data. It should be noted, a malicious user who obtains the data address and other required information might use route function and access the data. To avoid that attack scenario, we implemented permission list for the users.

As to blockchain.py, it controls the blockchain network via the class Blockchain. The class is consisted of several functions. To concisely illuminate the process, when a new block is intended to be generated in app.py, it calls get_previous_block of Blockchain class (in blockchain.py). This function simply returns the last element of the list of blocks. After receiving the last block, the hash of last block is extracted. This process is not complex since the blocks are defined as a dictionary where hash value can be easily found by its key.

As the next step, add_transactions of the class is activated where it receives parameters of the proposed transaction comprising sender, receiver and data. When the transaction is transferred from app.py to blockchain.py, create_block function of the class generates a new block based on the transaction, hash of previous block and hash of the generated block.

4.2 Front-end

Regarding the front-end role, it plays an intermediate role between the user and the back-end. In fact, it provides a human readable atmosphere for the user to cover all the complexities of back-end that are machine readable. With that aim, we used Bootstrap 4.1.3 version that is a popular front-end framework. We worked on this web framework and applied it on our initial web pages that were designed by HTML and CSS.

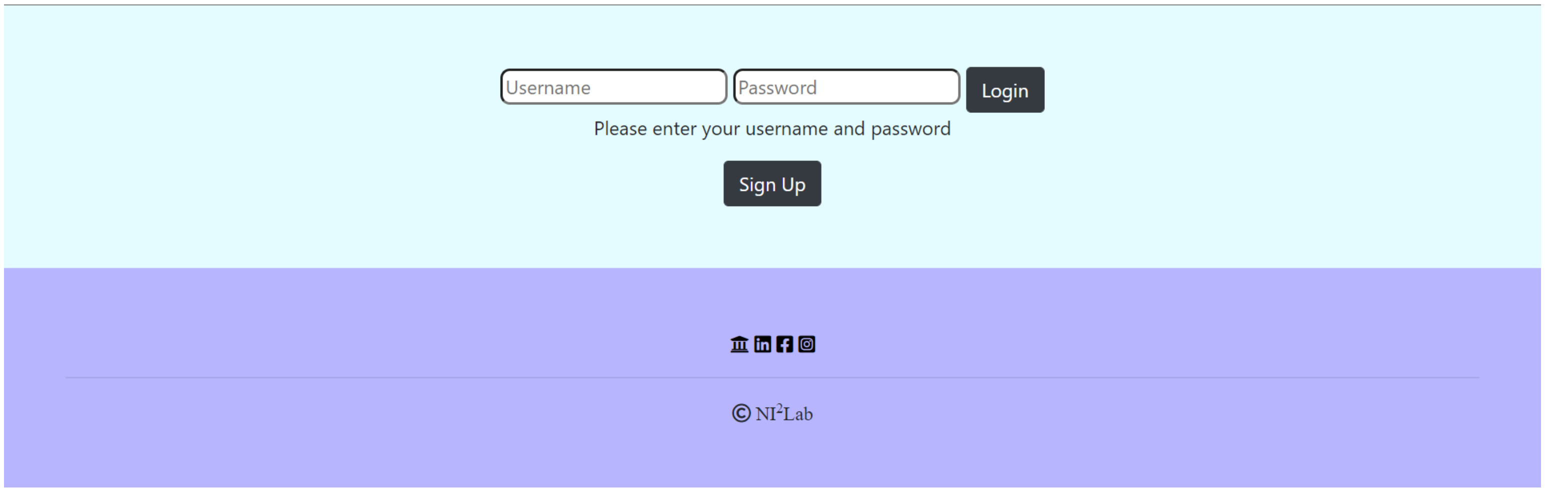
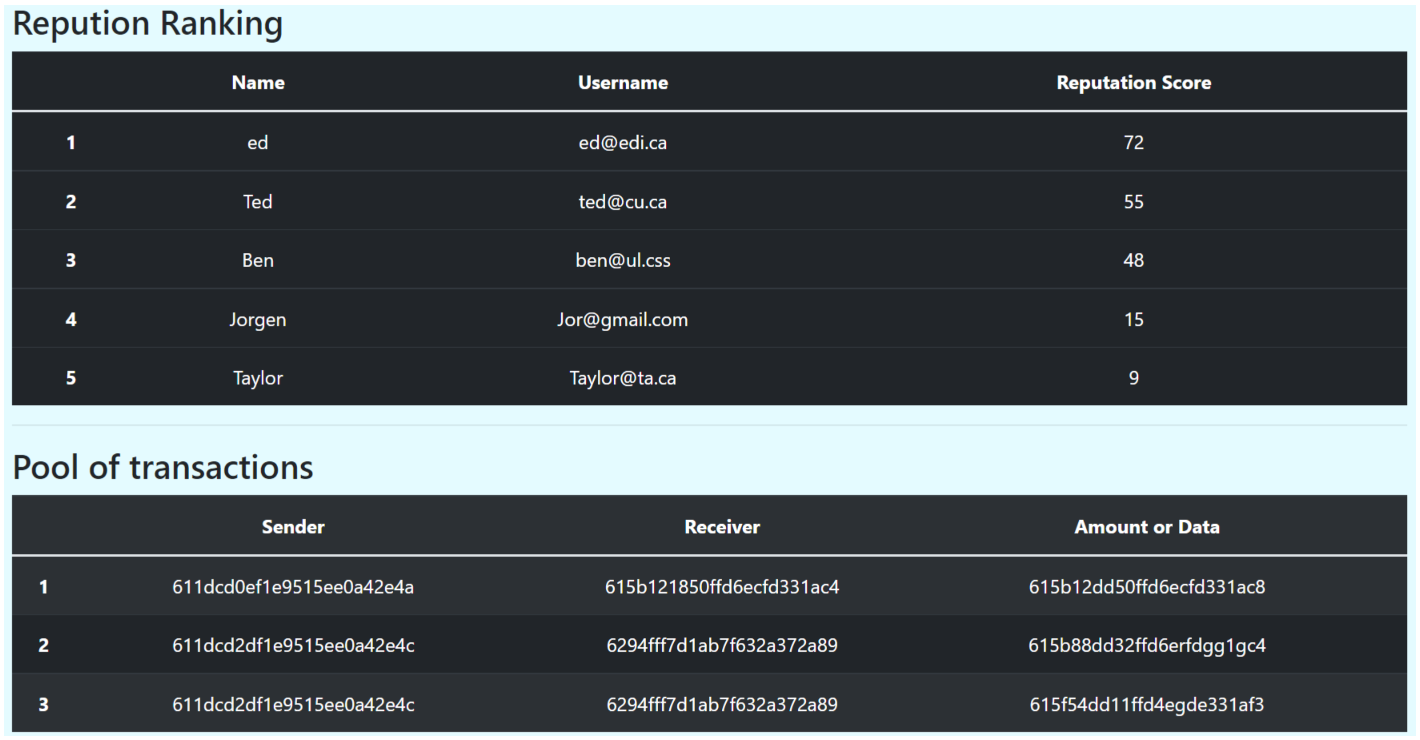


Figure 4.2: User authentication page of the framework

The most generic and multifarious HTML page of the framework is home.html in which, at early lines, there exists the navigation bar code that uses Bootstrap. Following that, we included python code in the HTML file since we wanted to define conditions and more complex loops. The examples of the applications of python code is it checks whether the person is an authority or not. If not an authority, it only presents the current blockchain network while if the user is an authority it shows the pool of transactions and reputation of users as well as the blockchain network.

As another point about our HTML pages, we move through the pages using url_for() method which is a crucial attribute of Flask. For instance, when an authority pushes the validation button for the transactions, placed on the home page, by url_for('validation',token=token), it takes the user to the validation.html page to conduct the validation. The other parameter of the method, token, is the token of user that should be transferred as the user goes to a new page. Also, as the user confronts validation.html, the program assesses the integrity of token to show the new page to the user. It is noteworthy, all the mentioned token authentication process is behind the scene in the back-end, and the user feels no inconvenience. To demonstrate the front-end code results, the screenshots of the framework from user experience perspective have been shown in figure 4.2-4.7.


Reption Ranking

	Name	Username	Reputation Score
1	ed	ed@edi.ca	72
2	Ted	ted@cu.ca	55
3	Ben	ben@ul.css	48
4	Jorgen	Jor@gmail.com	15
5	Taylor	Taylor@ta.ca	9

Pool of transactions

	Sender	Receiver	Amount or Data
1	611dcd0ef1e9515ee0a42e4a	615b121850ffd6ecfd331ac4	615b12dd50ffd6ecfd331ac8
2	611dcd2df1e9515ee0a42e4c	6294fff7d1ab7f632a372a89	615b88dd32ffd6erfdgg1gc4
3	611dcd2df1e9515ee0a42e4c	6294fff7d1ab7f632a372a89	615f54dd11ffd4egde331af3

Figure 4.3: Reputation of the framework users

4.3 Database

As for the database, we confronted a dilemma in which we could either utilize SQL or NoSQL sort of database. Due to the fact that NoSQL type outperforms relational databases (SQL), in terms of quicker data response [39], we selected NoSQL databases. In addition, among diverse NoSQL databases, we chose MongoDB since it manages the data more efficiently compared to the counterparts. Especially as the data size increases, its outperformance will be more comprehensible [40]. Also, the other feature that gave MongoDB the upper hand is the compatibility with Python language. Lastly, we exploited MongoDB database by the tier that allows to store data up to 512MB size.

Respecting the database architecture, it is consisted of multiple collections. Each collection is similar to a branch protecting particular data types. As shown in 4.8, collection data holds the actual data of data owners. First parameter is id of this document, secondly we have the identity of the data owner. Following parameter is password that is a password assigned by the data owner on this data package. Subsequently, there exists the data that are extracted using extract_csv function.

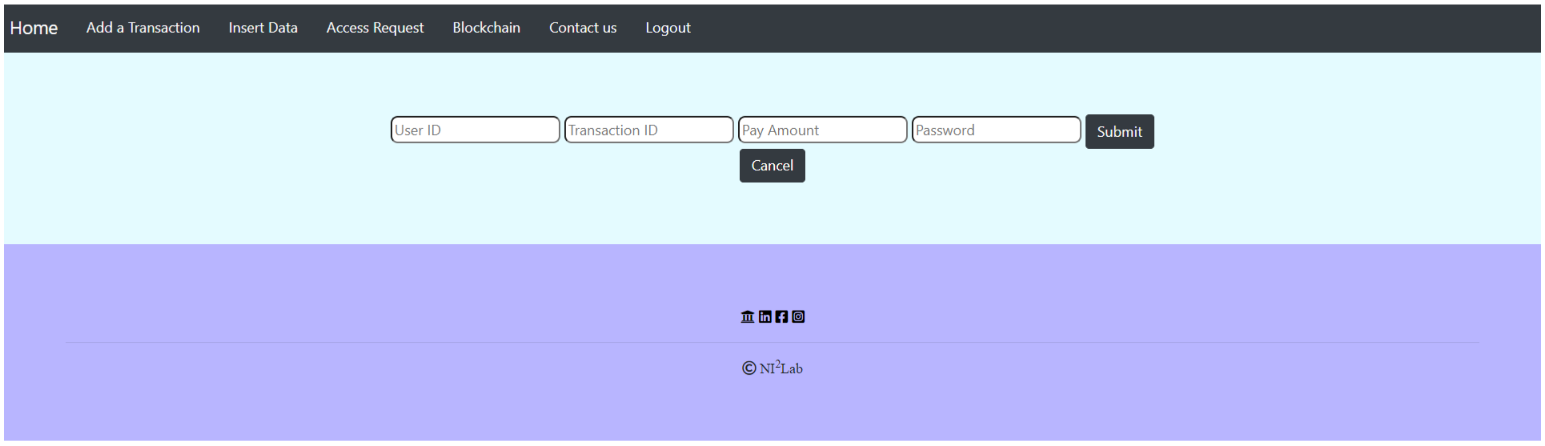
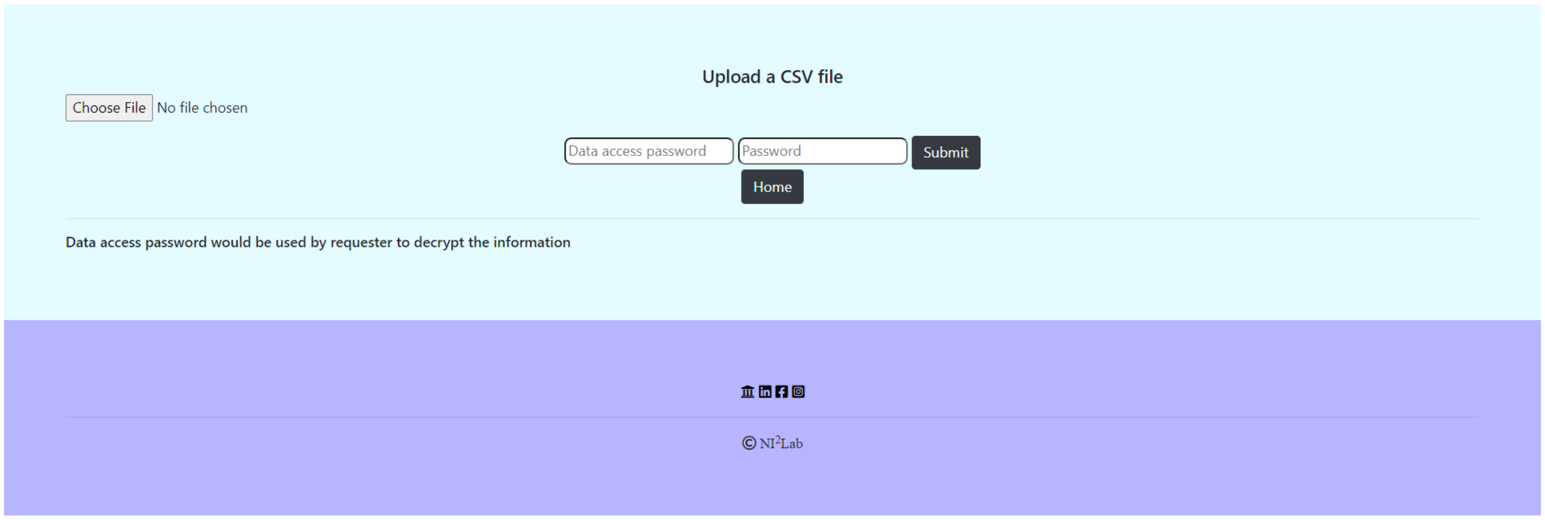


Figure 4.4: Request generation page of the framework

The collection blockchain, as shown in figure 4.9, contains records each one showing a sender, receiver and data of a transaction held by a certain block. Figure 4.10 depicts the trusted collection wherein the list of authorities is stored. As a matter of fact, its records track the document id plus information regarding the identity of the authority. As an authority is expelled or added to the list, this collection becomes updated via app.py. Pool of transactions is included in pool collection and, according to figure 4.11, each transaction owns an id, a sender, receiver and data. Finally, figure 4.12 demonstrates the pool of access requests where it contains information regarding the data owner, data requester identity as well as the targeted transaction.

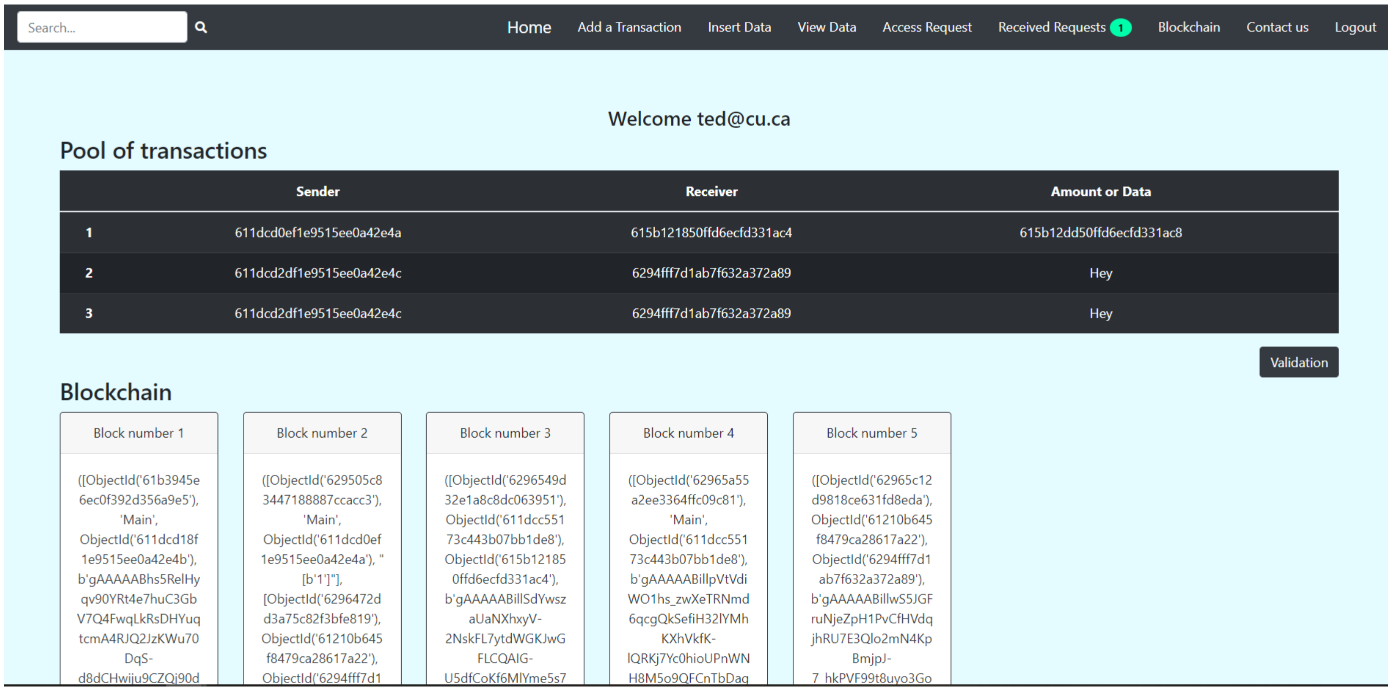

Upload a CSV file

Choose File No file chosen

Data access password would be used by requester to decrypt the information

© NI²Lab

Figure 4.5: Data contribution page of the framework



Welcome ted@cu.ca

Pool of transactions

	Sender	Receiver	Amount or Data
1	611dcd0ef1e9515ee0a42e4a	615b121850ffd6ecfd331ac4	615b12dd50ffd6ecfd331ac8
2	611dcd2df1e9515ee0a42e4c	6294fff7d1ab7f632a372a89	Hey
3	611dcd2df1e9515ee0a42e4c	6294fff7d1ab7f632a372a89	Hey

Blockchain

Block number 1
```
([ObjectId('61b3945e
6ec0f392d356a9e5'),
'Main',
ObjectId('611dcd18f
1e9515ee0a42e4b'),
b'gAAAAABhs5RelHy
qv90YRt4e7huC3Gb
V7Q4FwqLkRsDHYuq
tcmA4RJQ2JzKWu70
DqS-
d8dCHwiju9CZQj90d
```

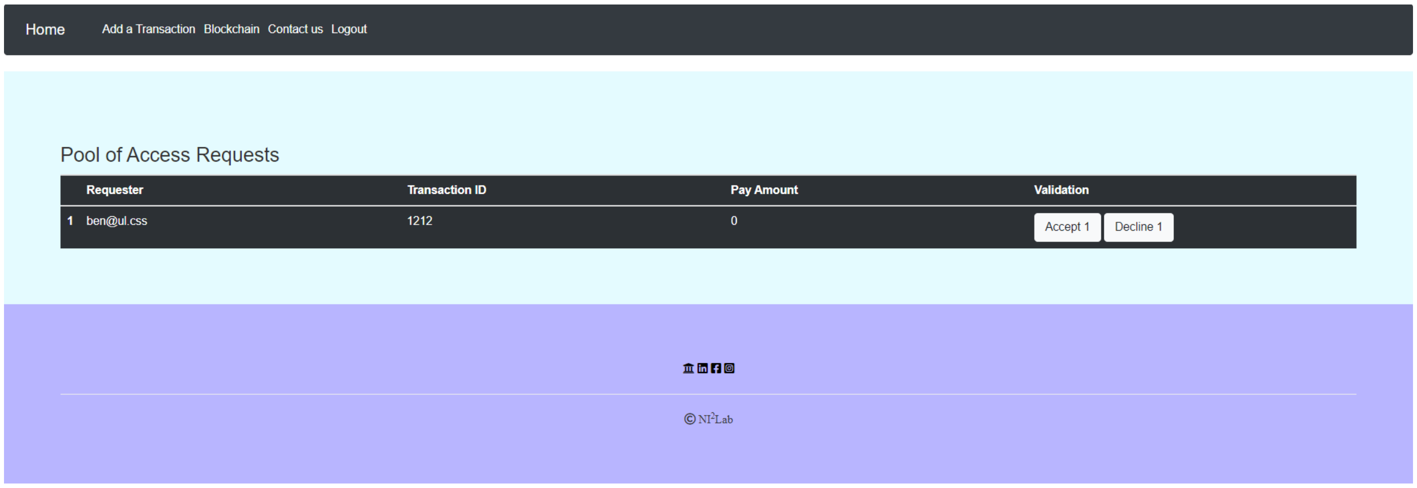
Block number 2
```
([ObjectId('629505c8
3447188887ccacc3'),
'Main',
ObjectId('611dcd0ef
1e9515ee0a42e4a'), "
[b'1']"],
[ObjectId('6296472d
d3a75c82f3bfe819'),
ObjectId('61210b645
f8479ca28617a22'),
ObjectId('6294fff7d1
```

Block number 3
```
([ObjectId('6296549d
32e1a8c8dc063951'),
ObjectId('611dcc551
73c443b07bb1de8'),
ObjectId('615b12185
0ffd6ecfd331ac4'),
b'gAAAAABillSdYwsz
aUaNXhxyV-
2NskFL7ytdWGKJwG
FLCQAIG-
U5dfCoKf6MlYme5s7
```

Block number 4
```
([ObjectId('62965a55
a2ee3364ffc09c81'),
'Main',
ObjectId('611dcc551
73c443b07bb1de8'),
b'gAAAAABillpVtVdi
WO1hs_zwXeTRNmd
6qcgQkSefiH32lYMh
KXhVkfK-
lQRKj7Yc0hioUPnWN
H8M5o9QFCnTbDag
```

Block number 5
```
([ObjectId('62965c12
d9818ce631fd8eda'),
ObjectId('61210b645
f8479ca28617a22'),
ObjectId('6294fff7d1
ab7f632a372a89'),
b'gAAAAABillwS5JGF
ruNjeZpH1PvCfHVdq
jhRU7E3Qlo2mN4Kp
BmjpJ-
7_hkPVF99t8uyo3Go
```

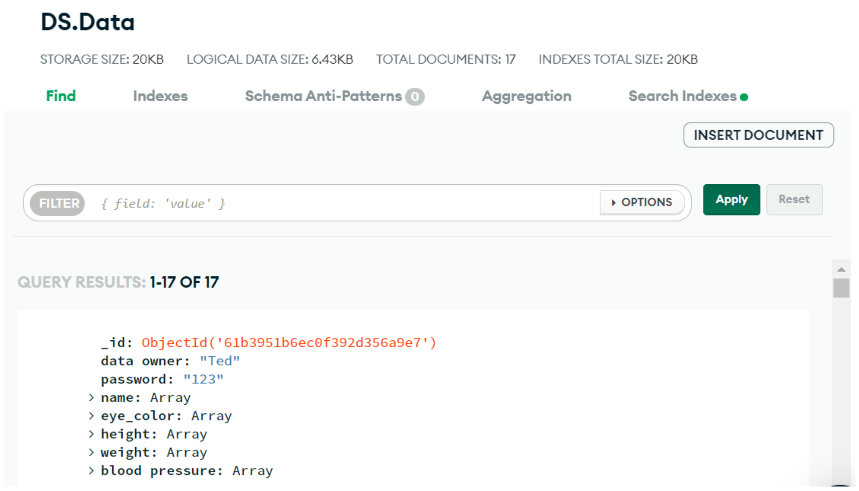
Figure 4.6: Received requests notification for a data owner



Pool of Access Requests

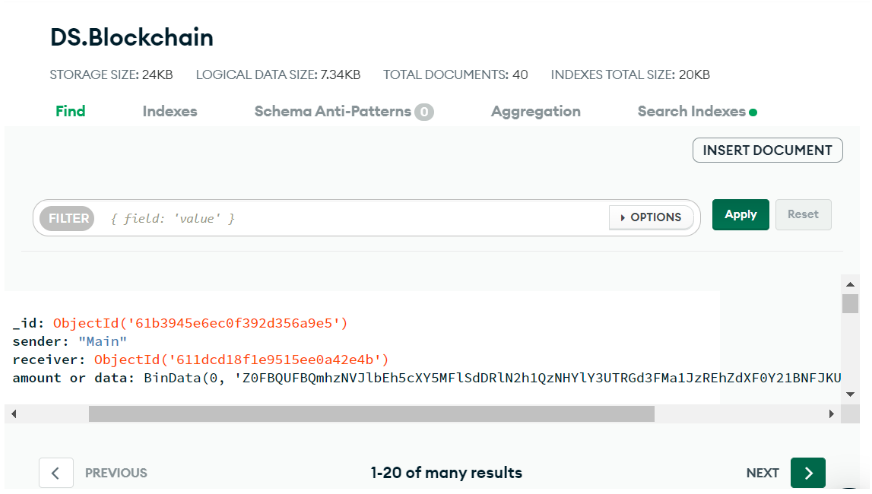
	Requester	Transaction ID	Pay Amount	Validation
1	ben@ul.css	1212	0	Accept 1 Decline 1

Figure 4.7: Received requests for a data owner



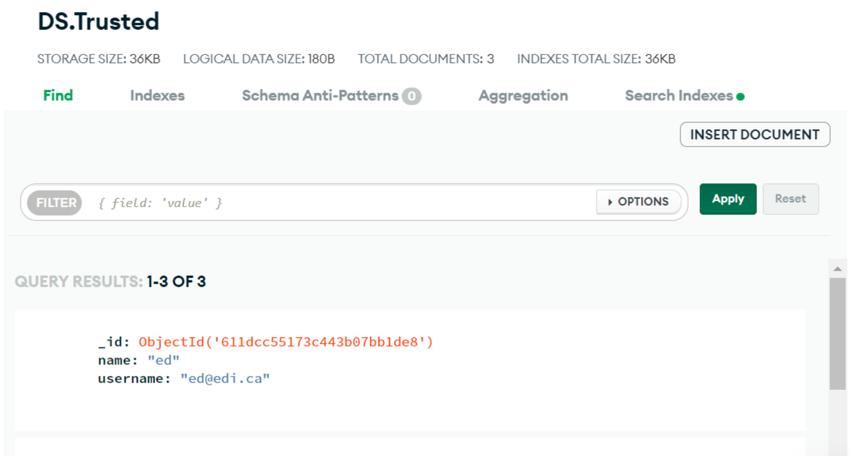
```
  _id: ObjectId('61b3951b6ec0f392d356a9e7')
  data owner: "Ted"
  password: "123"
> name: Array
> eye_color: Array
> height: Array
> weight: Array
> blood pressure: Array
```

Figure 4.8: Database (MongoDB) architecture, data collection



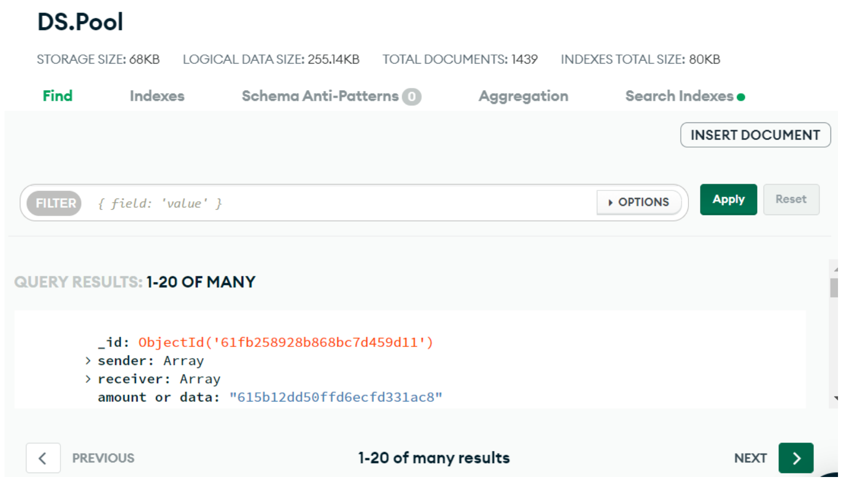
```
_id: ObjectId('61b3945e6ec0f392d356a9e5')
sender: "Main"
receiver: ObjectId('611dcd18f1e9515ee0a42e4b')
amount or data: BinData(0, 'Z0FBQUFBQmhzNVJlbEh5cXY5MFlSdDRlN2h1QzNHYlY3UTRGd3FMa1JzREhZdXF0Y21BNFJKU
```

Figure 4.9: Database (MongoDB) architecture, blockchain collection



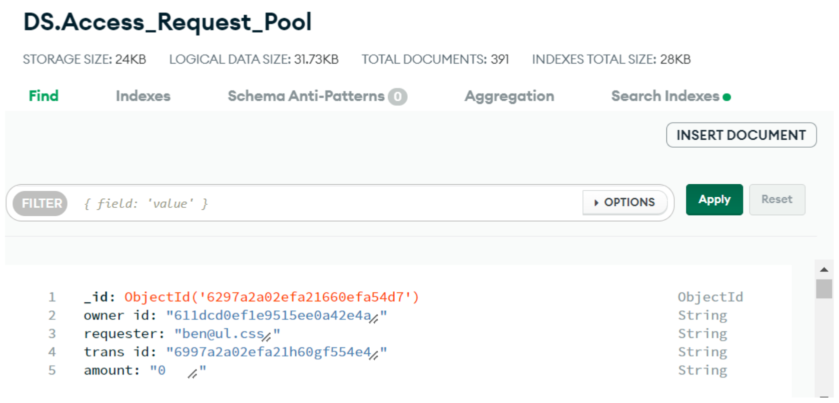
```
_id: ObjectId('611dcc55173c443b07bb1de8')
name: "ed"
username: "ed@edi.ca"
```

Figure 4.10: Database (MongoDB) architecture, trusted collection



```
  _id: ObjectId('61fb258928b868bc7d459d11')
> sender: Array
> receiver: Array
  amount or data: "615b12dd50ffd6ecfd331ac8"
```

Figure 4.11: Database (MongoDB) architecture, pool collection



```
_id: ObjectId('6297a2a02efa21660efa54d7')          ObjectId
owner id: "611dcd0ef1e9515ee0a42e4a"              String
requester: "ben@ul.css"                           String
trans id: "6997a2a02efa21h60gf554e4"              String
amount: "0   "                                    String
```

Figure 4.12: Database (MongoDB) architecture, access requested pool collection

Chapter 5

Performance Assessment

This chapter shows the performance and quality of the proposed data sharing framework and also measures diverse metrics to analyze the functionality.

5.1 Scalability

To measure and compare the scalability of our proposed idea, we analyzed two popular works of our field namely, Medblock [10] and Medshare[11]. To be consistent with [10] and [11], we evaluated real-case scenarios in which the latency is assumed as the time span between producing a package of data to delivering that. It includes all the stages required to process a request by entire entities, to have a fair comparison with our two counterparts. The results are measured within five experiments and the presented results are the average rates. We assessed the latency for different number of users and this is for two reasons. As the first one, to grasp the trend of changes of latency by increasing the users, and for the later reason, to avoid any accidental results that might be gained by a certain number of users. As figure 5.1 demonstrates, our proposed scheme generates fairly lower latency while dealing with data delivery. We believe the reason of our outperformance traces back to the differences in structure of Medblock and Medshare compared to ours. In terms of Medblock, as Fan el al. explain, the security of this work is highly dependent on cryptographic keys. For instance, as the first step, Certificate Authority (CA), an entity of Medblock, generates public and private keys of a patient, the generated private key of patient might be utilized by CA to

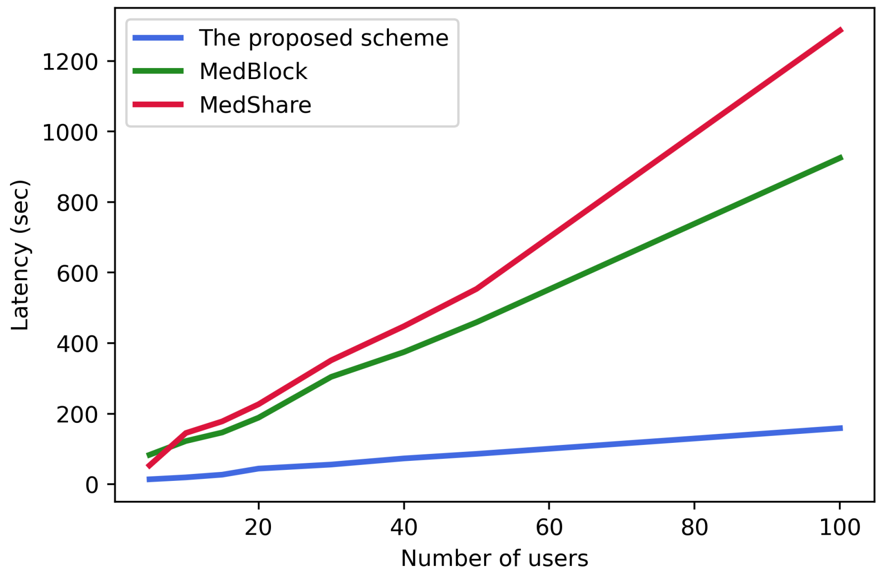


Figure 5.1: Latency of service provider requests

decrypt data in several cases as well. Also, when a patient intends to encrypt the data, the one has to use her public key then sign the data by her private key [10].

Furthermore, Xia et al. indicate that Medshare virtually follows the same path. As a matter of fact, the requester in Medshare is responsible for generating her public key and private key. Also, requester has to send her public key to the data owner authenticator with purpose of verification of the requester identity. Besides, the authenticator utilizes the requester public key to encrypt the response package to the requester. Considering authenticator contract keys in Medshare, they ought to be generated by authenticator the corresponding actions of data exchange [11]. All the aforementioned steps of generation, transfer and validation of public keys, private keys and signature imposes latency to the systems. In addition, due to local based nature of this sort of encryption, the performance of user's device including its computational power and network connectivity may massively influence the latency of framework.

The second factor that impacts the latency of the data sharing frameworks is use of external tools. As an illustration, Medblock exploits bread crumbs to improve information retrieval process [10]. This idea is in opposition to our unified Python code. In fact, involving several tools in the framework foists communication cost and latency on the system.

Besides, we evaluated our proposed consensus algorithm to assure the performance. As a matter of fact, the consensus latency is the parameter to be assessed and as shown in figure 5.2, it is calculated in case of 1, 10, 100 and 1000 transactions. We demonstrate the results by two diagrams. Table 5.1-5.5 show the validation time of each authority where we assigned five authorities to our framework. In our validation we assumed each authority accepts the transaction, thereby, the latency to reach consensus is the time to receive three out of five votes that considered as the majority. According to the figure 5.2, as expected the consensus latency vastly increases by growing the number of users. The notable point is, by having 1000 transactions, the consensus time does not surpass one minute. It is deemed remarkable since the mentioned latency is quite lower in comparison with two popular clients of Ethereum blockchain, Parity and Geth, that can process 1000 transactions within 1.74 and 3.27 minutes, respectively [41].

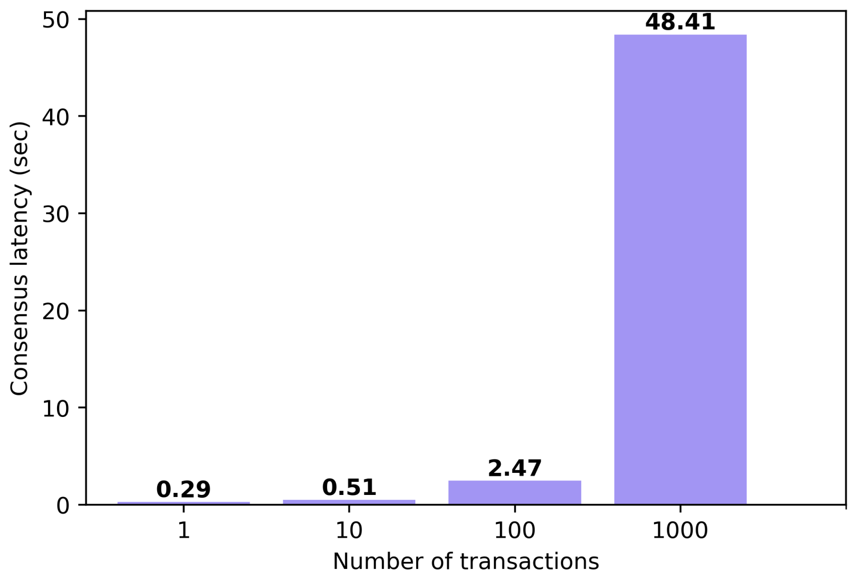


Figure 5.2: Latency of consensus algorithm

5.2 Security

Due to importance of concept of security in data sharing framework, both transaction manager and record manager protect the security and privacy of users. In transaction manager, the users require high reputation to be in the list of the potential authorities and leader. Also, the selection process of the authority members and leader is random. This idea ceases any deterministic plan for attack by malicious users. Besides, even if a malicious user successfully penetrates through the group of authorities, still the votes of majority of validators outweigh. Moreover, due to presence of TFs, any unusual decision of the attacker is reported to the leader and the leader may issue penalty. These measurements result in decreasing the feasibility of sybil attack wherein a malicious user or group of users try to influence the validation of transactions.

Additionally, record manager has permission handler subsection by which it can control the access of users to the data packages. In fact, permission handler appends a particular data id to the permission list of a valid user. This is an efficient solution to protect data in cases like attack to the blockchain and access to address of data in database. In this attack scenario the system annuls retrieval of data to attacker's profile.

Reputation systems, in spite of their merits, bring undeniable security concerns as well, as explained in [33] and [34]. Nonetheless, we believe our proposed data sharing framework is able to manage these concerns. The first problem is the selfish users who try to manipulate the system to gain reputation. To prevent these users who try to produce many unnecessary transactions within a short period of time, PF is featured in our proposed method. According to our scheme, PF reduces the conferred reputation to a certain data owner and data requester who has generated unusual data traffic.

As to second concern, sybil attack substantially threatens the security of reputation systems. In our proposed data sharing framework, the attackers have to obtain the control of transactions validation and also take the leader seat to nullify the effect of TFs. In our framework there are two major obstacles against taking the majority of authorities by malicious users. Firstly, to be considered for an authority, the user has to gain high reputation while by PF dishonest behavior will not result in high reputation. As the second hurdle, we proposed a random selection to avoid any

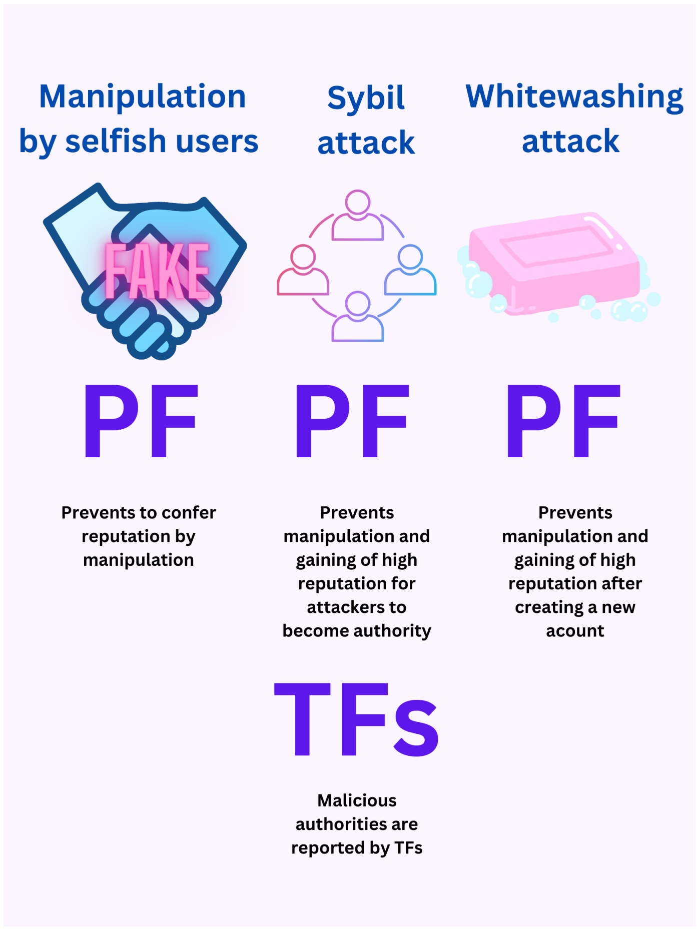

Manipulation by selfish users
Sybil attack
Whitewashing attack
FAKE
PF
PF
PF
Prevents to confer reputation by manipulation
Prevents manipulation and gaining of high reputation for attackers to become authority
Prevents manipulation and gaining of high reputation after creating a new acount
TFs
Malicious authorities are reported by TFs

Figure 5.3: Summary of improvements by proposed methods

deterministic attack scheme by malicious users. As to susceptibility against whitewashing attacks, in this type of attack, the user with low reputation, deletes her account and creates a new account by which she acquires the chance to become an influential user of the system, e.g. become an authority. In our proposed scheme, this action only leads to removal of the user account while the user still requires to gain reputation in an honest way, to be a potential candidate of authority. Finally, figure 5.3 has summarized the incentive mechanism enhancement by our proposed solutions.

Authority 1 latency to vote (sec)			
Transaction 1	Transaction 10	Transaction 100	Transaction 1000
0.283	0.496	2.471	48.231

Table 5.1: Latency of authority 1

Authority 2 latency to vote (sec)			
Transaction 1	Transaction 10	Transaction 100	Transaction 1000
0.412	0.506	2.768	48.973

Table 5.2: Latency of authority 2

Authority 3 latency to vote (sec)			
Transaction 1	Transaction 10	Transaction 100	Transaction 1000
0.282	0.481	2.909	46.662

Table 5.3: Latency of authority 3

Authority 4 latency to vote (sec)			
Transaction 1	Transaction 10	Transaction 100	Transaction 1000
0.306	0.536	2.416	52.224

Table 5.4: Latency of authority 4

Authority 5 latency to vote (sec)			
Transaction 1	Transaction 10	Transaction 100	Transaction 1000
0.286	0.517	2.191	48.410

Table 5.5: Latency of authority 5

Chapter 6

Conclusion and Future Work

This chapter concludes the contributions of this work and finally, it presents future works to further develop the proposed data sharing framework.

6.1 Conclusion

In conclusion, data sharing technology has received plenty of benefits by adoption of blockchain. A considerable number of security concerns are tackled due to immutability, decentralization and no need to have a third party; however, all that glitters is not gold. Data sharing has encountered several challenges brought by blockchain as well like substantial power consumption, biased voting and high communication costs. We proposed a new system model leveraging a new consensus algorithm in which the aforementioned problems are solved. In other words, the proposed method is based on a group of authorities and a leader who validates the transactions.

In addition, the reputation systems are popular sort of incentive mechanism by which the networks can reward the honest and collaborative users. Although, reputation systems sound promising, they add many loopholes to the networks. Regarding the drawbacks, they lead to susceptibility of data sharing framework against manipulation by selfish users and different attack scenarios. In our proposed idea, we introduced PF and TFs, these two new concepts can defend the reputation systems against mentioned issues. In terms of encryption, we proposed a substitute for local based encryption methods in which cryptographic keys play a crucial role for validation, encryption and decryption tasks. In fact, we adopted a cloud-centric encryption solution by which the latency is improved.

We implemented our proposed scheme using a different methodology compared to conventional implementation of data sharing frameworks in similar research works. As a matter of fact, we developed a web application in which most of the functionalities are programmed from scratch to avoid unpleasant side effects of built-in and external tools like communication costs. Finally, as shown in the results, our proposed data sharing framework not only mitigated the security concerns but also it has reduced the latency and communication costs to a considerable extent.

6.2 Future Work

We have divided the future works into two subsections that are represented in 6.2.1 and 6.2.2.

6.2.1 Data Quality Management

When a researcher utilizes a certain type of data, the one has to assure that the quality of data is satisfying, since inadequate data quality leads a researcher to an incorrect conclusion or train her model falsely. Moreover, by evaluation of the quality of data contributed by a data owner, the owner with higher quality data can be incentivized more by our reputation system. Therefore, a requester can trust the data owner who has high reputation.

The data quality management problem originates from the fact that data sharing framework is composed of users each contributing a large volume of data and it is perceived a big data platform wherein the control of entered data is complicated. Regarding the existing solutions, one popular and naïve idea is to validate the quality of data by group of validators. However, we have to seek an automated solution to replace manual assessment of the data quality by validators, and there are different reasons to urge us finding this solution. First and foremost, an automated solution is done by a computer program and there is no involved third party, therefore, user privacy is not considered violated. As the second reason, for large data volumes, manual validation yields substantial rate of latency.

To solve the problems, the methods are based on statistical and machine learning algorithms.

For example, one of the potential solutions is to use k-means clustering algorithm in which each piece of data is grouped into diverse categories based on its features and correlation compared to its counterparts. For instance, in a medical application for data sharing framework, when height data of people is listed and it varies between approximately 1.5 m to 2 m, insertion of 4 m as the height is deemed outlier and low quality data. These sorts of systems usually are used in fraud detection and financial applications, therefore, use of them in data sharing applications is a novel research field.

There are several challenges and unsolved problems to train an AI model as described above. Firstly, a data sharing framework should become specified for a particular application before training the model. The reason is the framework has to be prepared and trained for the received type of data, e.g. only trained for medical data, and training a general purpose model for all types of data seems infeasible.

As the second challenge, training the models by data of users is not a reliable solution. In fact, the users might contribute low quality data. The system is not able to distinguish low quality data before being trained by high quality one. In other words, regarding the aforementioned height example, a user may contribute height data that is ranged between 5m to 6m. Therefore, the model is trained falsely and expects this range from further data owners. As a result, we have to provide high quality data to train the model in the first place. Undeniably, supplying high quality data is not always easy especially in industrial IoT applications and health care where the data might be confidential.

6.2.2 Recommender System

To clarify the idea, the recommender system of this framework is responsible to receive the requested data type by the data requester then recommends the relevant data owners. As an illustration, if a data requester asks for ”fat” data, the recommender system expects to find data owners' data type like ”fat”, ”Cholesterol”, ”HDL”, ”LDL”, etc.

Our solution for this problem is based on deep learning. As a matter of fact, Natural Language Processing (NLP) is a branch of deep learning to process speech and text information. We believe NLP alongside an unsupervised learning algorithm of machine learning can manage the task. NLP is a requisite to organize and vectorize the text and make it ready to be processed. Also, the unsupervised learning should receive a vector for each word. The vectors that are close can be placed within one cluster. There are plenty of methods to calculate the closeness of vectors, a popular example is Cosine similarity method.

Bibliography

[1] Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: *Decentralized Business Review* (2008), p. 21260.

[2] Wood. “Ethereum: A secure decentralised generalised transaction ledger”. In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[3] Nguyen et al. “Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities”. In: *IEEE Access* 7 (2019), pp. 85727–85745.

[4] Yang et al. “Delegated proof of stake with downgrade: A secure and efficient blockchain consensus algorithm with downgrade mechanism”. In: *IEEE Access* 7 (2019), pp. 118541–118555.

[5] Cao et al. “When Internet of Things meets blockchain: Challenges in distributed consensus”. In: *IEEE Network* 33 (2019), pp. 133–139.

[6] Han et al. “How can incentive mechanisms and blockchain benefit with each other? a survey”. In: *ACM Computing Surveys (CSUR)* (2022).

[7] Zhang et al. “Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain”. In: *Journal of medical systems* 42 (2018), pp. 1–8.

[8] Liang et al. “Integrating blockchain for data sharing and collaboration in mobile healthcare applications”. In: *2017 IEEE 28th annual international symposium on personal, indoor, and mobile radio communications (PIMRC)* (2017), pp. 1–5.

[9] Cachin. “Architecture of the hyperledger blockchain fabric”. In: *Workshop on distributed cryptocurrencies and consensus ledgers* 310 (2016), pp. 1–4.

[10] Fan et al. “Medblock: Efficient and secure medical data sharing via blockchain”. In: *Journal of medical systems* 42 (2018), pp. 1–11.

[11] Xia et al. “MeDShare: Trust-less medical data sharing among cloud service providers via blockchain”. In: *IEEE access* (2017), pp. 14757–14767.

[12] Shen et al. “MedChain: Efficient healthcare data sharing via blockchain”. In: *Applied sciences* (2019), p. 1207.

[13] Xia et al. “BBDS: Blockchain-based data sharing for electronic medical records in cloud environments”. In: *Information* 8 (2017), p. 44.

[14] Yu et al. “Blockchain-enhanced data sharing with traceable and direct revocation in IIoT”. In: *IEEE transactions on industrial informatics* 17 (2021), pp. 7669–7678.

[15] Li et al. “Blockchain-based privacy-preserving and rewarding private data sharing for IoT”. In: *IEEE Internet of Things Journal* 9 (2022), pp. 15138–15149.

[16] Kumar et al. “Blockchain-federated-learning and deep learning models for covid-19 detection using ct imaging”. In: *IEEE Sensors Journal* 21 (2021), pp. 16301–16314.

[17] Shayan et al. “Biscotti: A blockchain system for private and secure federated learning”. In: *IEEE Transactions on Parallel and Distributed Systems* 32 (2020), pp. 1513–1525.

[18] Lu et al. “Blockchain and federated learning for privacy-preserved data sharing in industrial IoT”. In: *IEEE Transactions on Industrial Informatics* 16 (2019), pp. 4177–4186.

[19] Nguyen et al. “A survey about consensus algorithms used in blockchain”. In: *Journal of Information processing systems* 14 (2018), pp. 101–128.

[20] Azaria et al. “Medrec: Using blockchain for medical data access and permission management”. In: *2016 2nd international conference on open and big data (OBD)* (2016), pp. 25–30.

[21] Tosh et al. “CloudPoS: A proof-of-stake consensus design for blockchain integrated cloud”. In: *2018 IEEE 11th international conference on cloud computing (CLOUD)* (2018), pp. 302–309.

[22] Liu et al. “BPDS: A blockchain based privacy-preserving data sharing for electronic medical records”. In: *2018 IEEE Global Communications Conference (GLOBECOM)* (2018), pp. 1–6.

[23] Liu et al. “A blockchain-based medical data sharing and protection scheme”. In: *IEEE Access* 7 (2019), pp. 118943–118953.

[24] Bentov et al. “Proof of activity: Extending bitcoin’s proof of work via proof of stake [extended abstract] y”. In: *ACM SIGMETRICS Performance Evaluation Review* 42 (2014), pp. 34–37.

[25] De Angelis et al. “PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain”. In: *ITASEC* (2018), pp. 1–11.

[26] https://www.parity.io.

[27] https://geth.ethereum.org.

[28] Xie et al. “Design and analysis of incentive and reputation mechanisms for online crowdsourcing systems”. In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 1 (2016), pp. 1–27.

[29] Xie et al. “Incentive and Reputation Mechanisms for Online Crowdsourcing Systems”. In: *IEEE/ACM International Symposium on Quality and Service* (2015), pp. 207–212.

[30] Alswailim et al. “A reputation system to evaluate participants for participatory sensing”. In: *2016 IEEE Global Communications Conference (GLOBECOM)* (2016), pp. 1–6.

[31] Burke et al. “Participatory sensing”. In: *World Sensor Web Workshop ACM Sensys* (2006), pp. 117–134.

[32] He et al. “SORI: A secure and objective reputation-based incentive scheme for ad-hoc networks”. In: *2004 IEEE Wireless Communications and Networking Conference (IEEE Cat. No. 04TH8733)* (2004), pp. 825–830.

[33] He et al. “A blockchain based truthful incentive mechanism for distributed P2P applications”. In: *IEEE Access* 6 (2018), pp. 27324–27335.

[34] Wang et al. “A blockchain based privacy-preserving incentive mechanism in crowdsensing applications”. In: *IEEE Access* 6 (2018), pp. 17545–17556.

[35] Gordon et al. “Blockchain technology for healthcare: facilitating the transition to patient-driven interoperability”. In: *Computational and structural biotechnology journal* 16 (2018), pp. 224–230.

[36] Pang et al. “What programming languages do developers use? a theory of static vs dynamic language choice”. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2018), pp. 239–247.

[37] https://cryptography.io/en/latest/fernet/.

[38] Alkhulaifi et al. “Exploring lattice-based post-quantum signature for JWT authentication: review and case study”. In: *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)* (2020), pp. 1–5.

[39] Sharma et al. “SQL and NoSQL Databases”. In: *International Journal of Advanced Research in Computer Science and Software Engineering* 2 (2012), pp. 20–27.

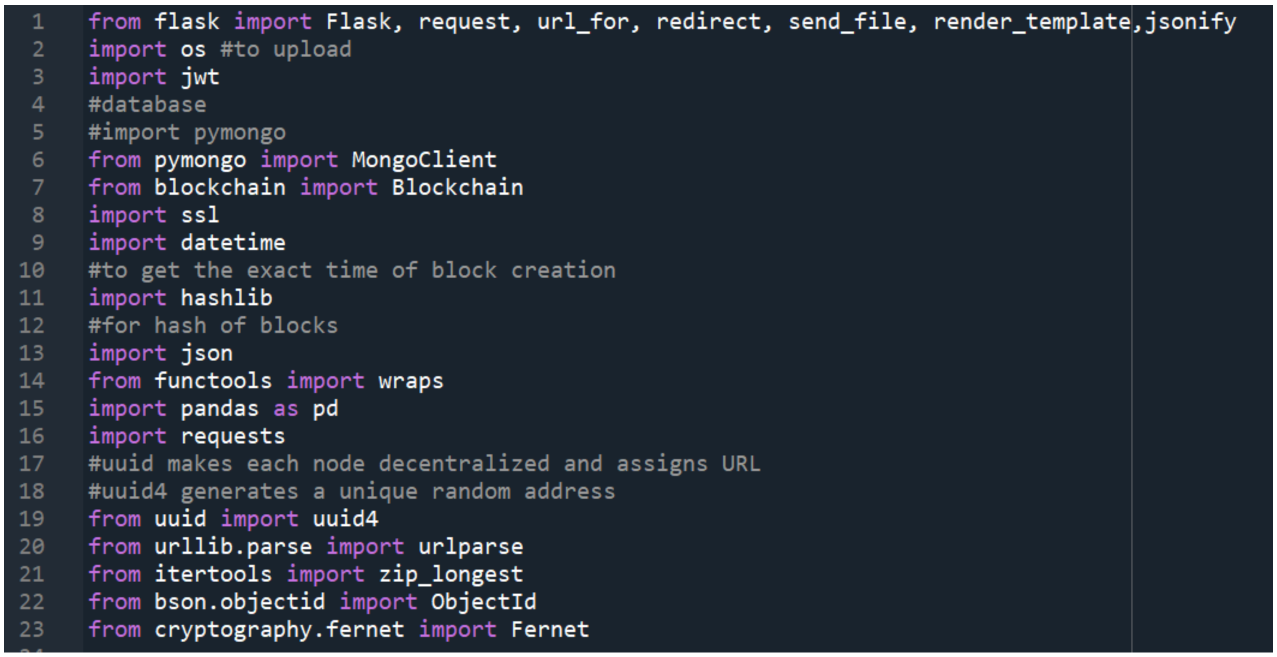
[40] Khan et al. “SQL and NoSQL Databases Software architectures performance analysis and assessments–A Systematic Literature review”. In: *arXiv preprint arXiv:2209.06977* (2022).

[41] Rouhani et al. “Performance analysis of Ethereum transactions in private blockchain”. In: *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)* (2017), pp. 70–74.

Appendix A

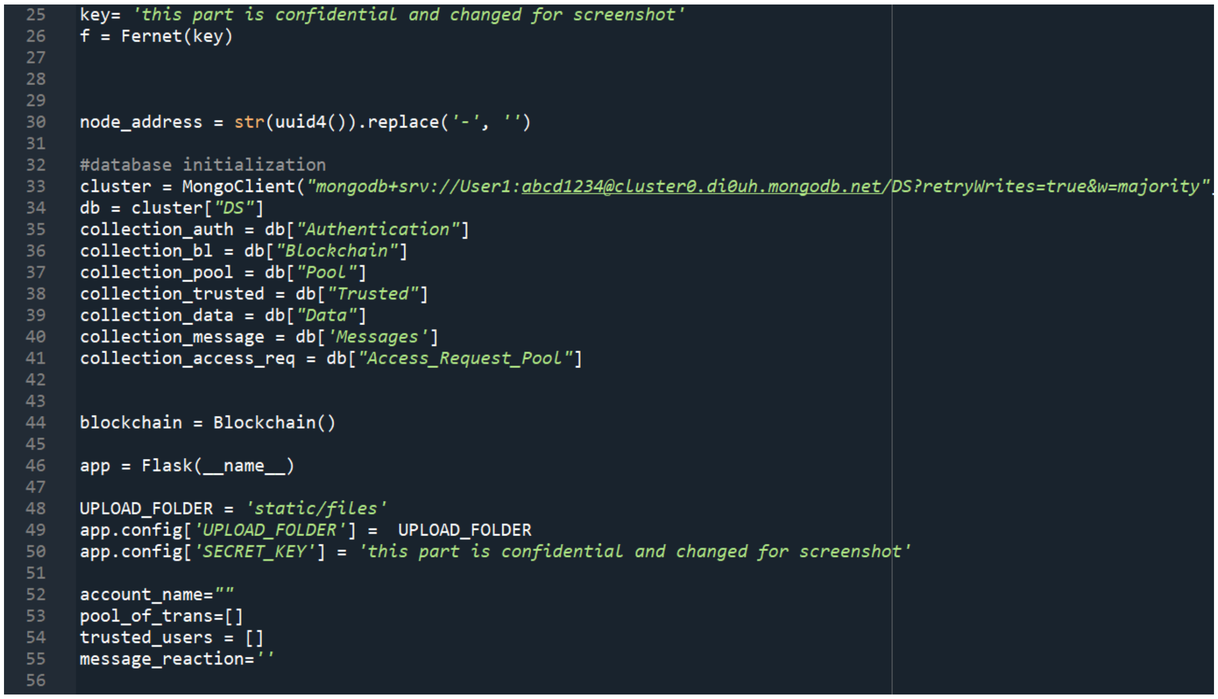
Implementation Materials (back-end)

The appendix contains back-end development codes of the data sharing framework.



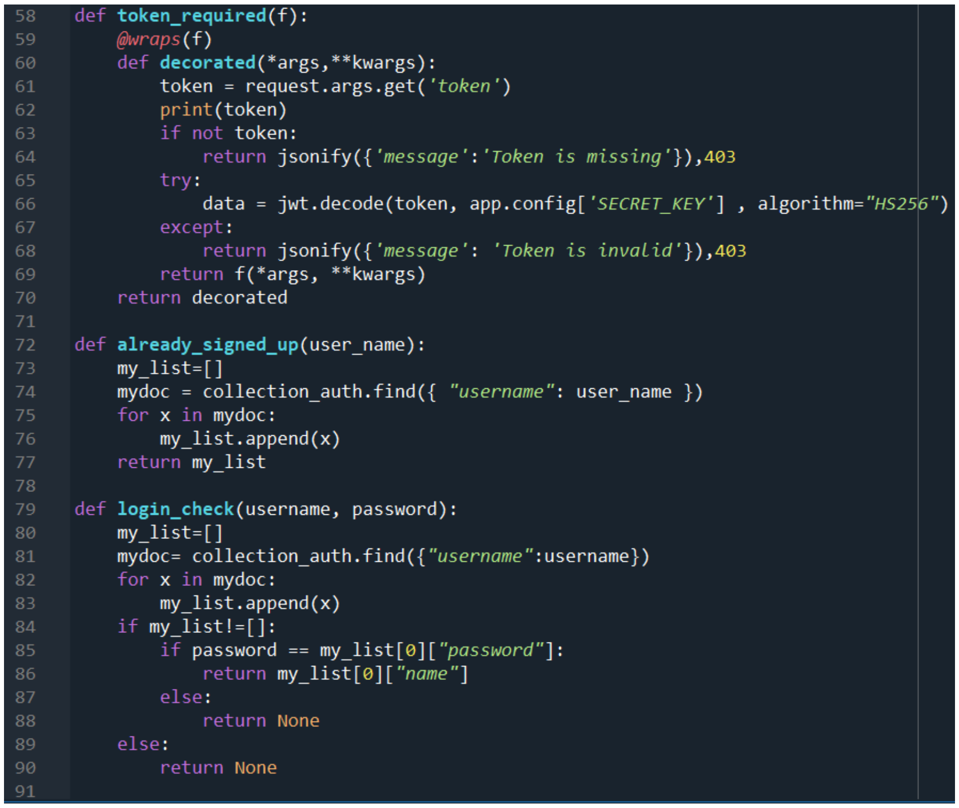
```python
from flask import Flask, request, url_for, redirect, send_file, render_template,jsonify
import os #to upload
import jwt
#database
#import pymongo
from pymongo import MongoClient
from blockchain import Blockchain
import ssl
import datetime
#to get the exact time of block creation
import hashlib
#for hash of blocks
import json
from functools import wraps
import pandas as pd
import requests
#uuid makes each node decentralized and assigns URL
#uuid4 generates a unique random address
from uuid import uuid4
from urllib.parse import urlparse
from itertools import zip_longest
from bson.objectid import ObjectId
from cryptography.fernet import Fernet
```

Figure A.1: The code in app.py file



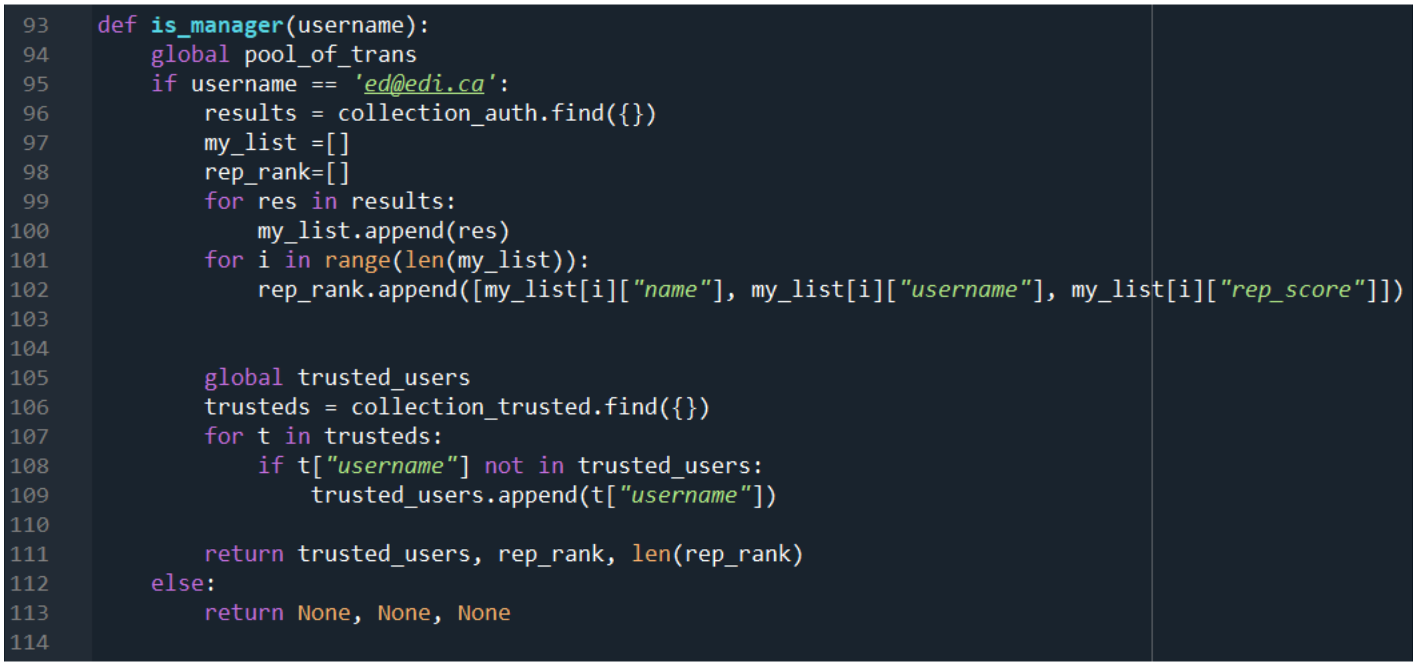
```python
key= 'this part is confidential and changed for screenshot'
f = Fernet(key)



node_address = str(uuid4()).replace('-', '')

#database initialization
cluster = MongoClient("mongodb+srv://User1:abcd1234@cluster0.di0uh.mongodb.net/DS?retryWrites=true&w=majority"
db = cluster["DS"]
collection_auth = db["Authentication"]
collection_bl = db["Blockchain"]
collection_pool = db["Pool"]
collection_trusted = db["Trusted"]
collection_data = db["Data"]
collection_message = db['Messages']
collection_access_req = db["Access_Request_Pool"]


blockchain = Blockchain()

app = Flask(__name__)

UPLOAD_FOLDER = 'static/files'
app.config['UPLOAD_FOLDER'] =  UPLOAD_FOLDER
app.config['SECRET_KEY'] = 'this part is confidential and changed for screenshot'

account_name=""
pool_of_trans=[]
trusted_users = []
message_reaction=''

```

Figure A.2: The code in app.py file



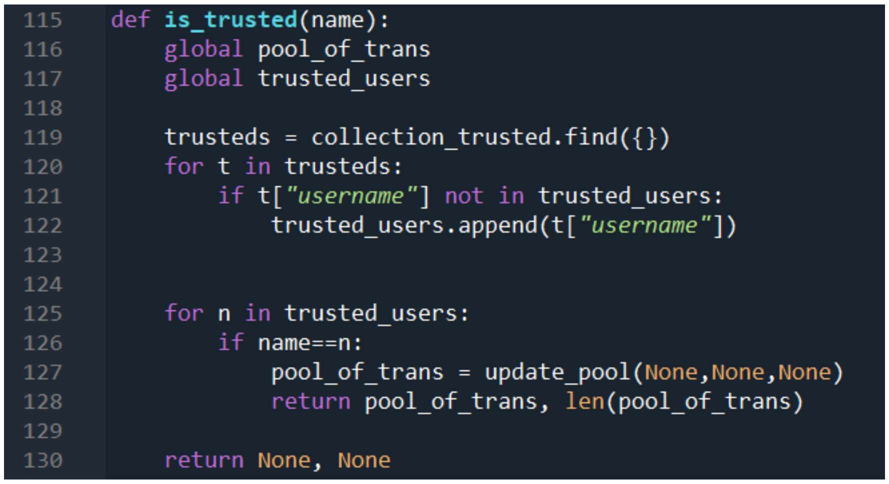
```python
def token_required(f):
    @wraps(f)
    def decorated(*args,**kwargs):
        token = request.args.get('token')
        print(token)
        if not token:
            return jsonify({'message':'Token is missing'}),403
        try:
            data = jwt.decode(token, app.config['SECRET_KEY'] , algorithm="HS256")
        except:
            return jsonify({'message': 'Token is invalid'}),403
        return f(*args, **kwargs)
    return decorated

def already_signed_up(user_name):
    my_list=[]
    mydoc = collection_auth.find({ "username": user_name })
    for x in mydoc:
        my_list.append(x)
    return my_list

def login_check(username, password):
    my_list=[]
    mydoc= collection_auth.find({"username":username})
    for x in mydoc:
        my_list.append(x)
    if my_list!=[]:
        if password == my_list[0]["password"]:
            return my_list[0]["name"]
        else:
            return None
    else:
        return None

```

Figure A.3: The code in app.py file



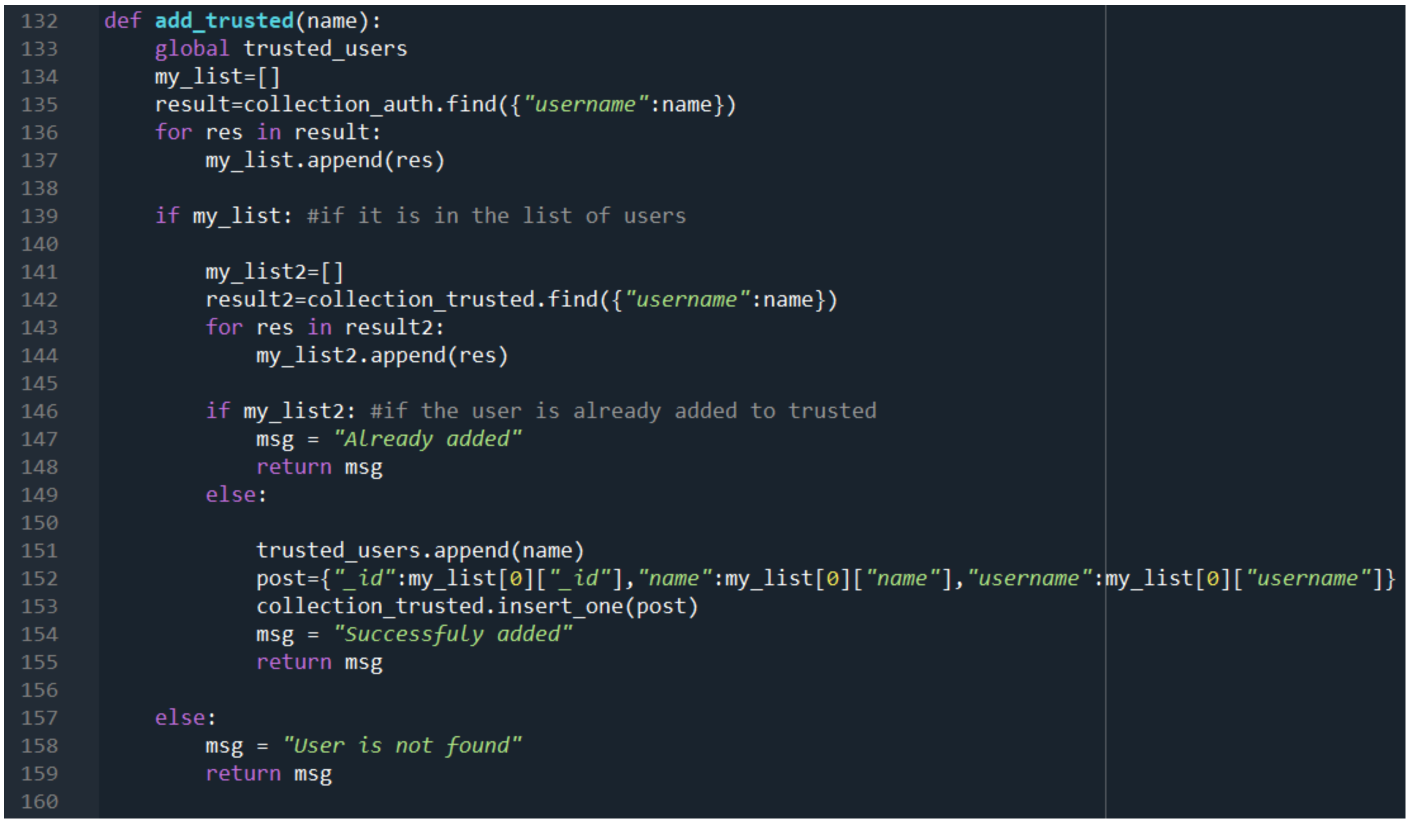
```python
def is_manager(username):
    global pool_of_trans
    if username == 'ed@edi.ca':
        results = collection_auth.find({})
        my_list =[]
        rep_rank=[]
        for res in results:
            my_list.append(res)
        for i in range(len(my_list)):
            rep_rank.append([my_list[i]["name"], my_list[i]["username"], my_list[i]["rep_score"]])


        global trusted_users
        trusteds = collection_trusted.find({})
        for t in trusteds:
            if t["username"] not in trusted_users:
                trusted_users.append(t["username"])

        return trusted_users, rep_rank, len(rep_rank)
    else:
        return None, None, None

```

Figure A.4: The code in app.py file



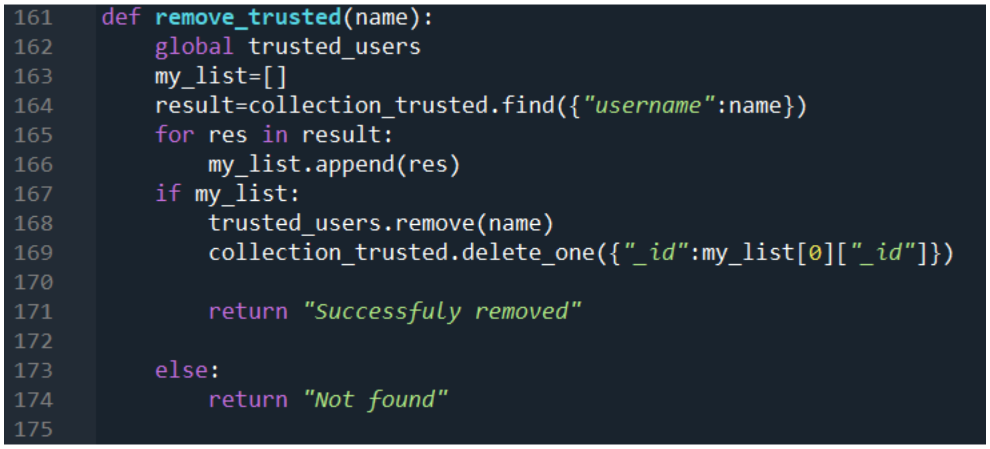
```python
def is_trusted(name):
    global pool_of_trans
    global trusted_users

    trusteds = collection_trusted.find({})
    for t in trusteds:
        if t["username"] not in trusted_users:
            trusted_users.append(t["username"])


    for n in trusted_users:
        if name==n:
            pool_of_trans = update_pool(None,None,None)
            return pool_of_trans, len(pool_of_trans)

    return None, None
```

Figure A.5: The code in app.py file



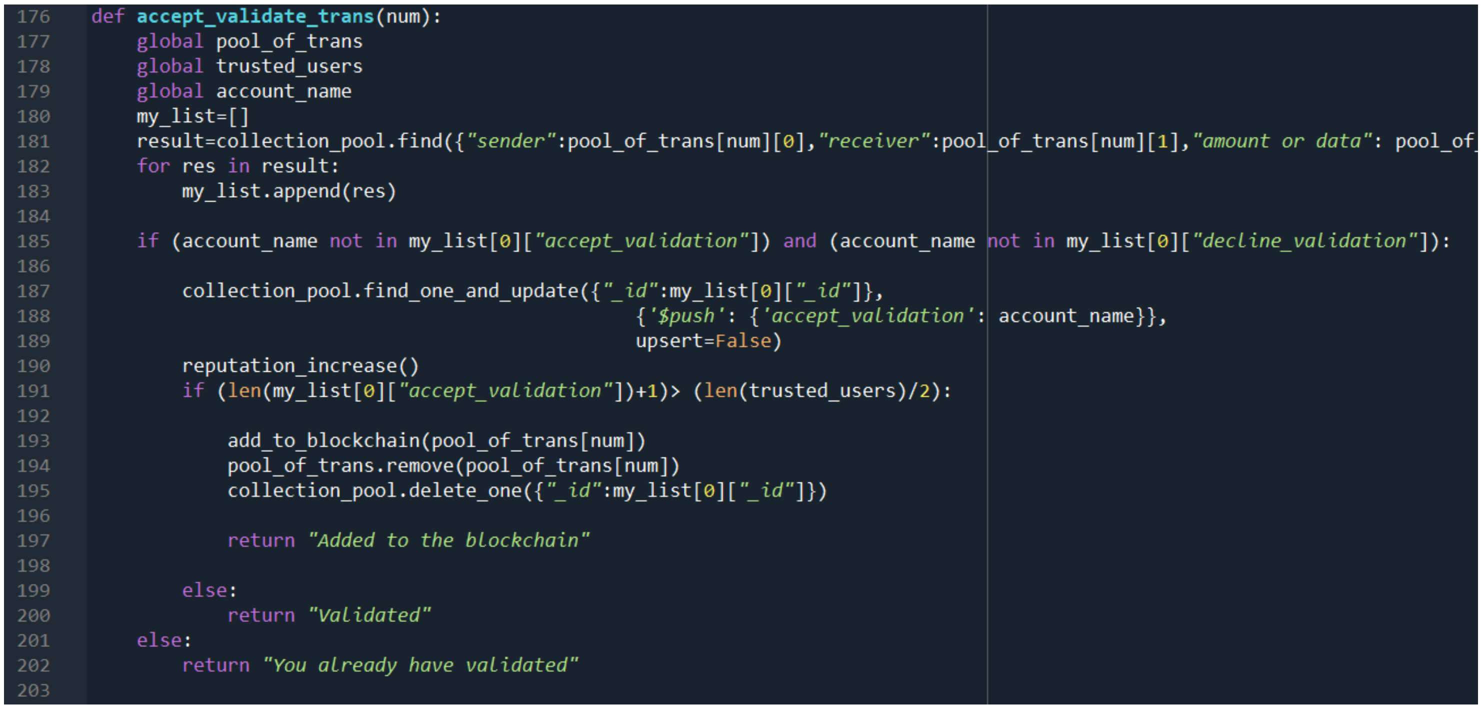
```python
def add_trusted(name):
    global trusted_users
    my_list=[]
    result=collection_auth.find({"username":name})
    for res in result:
        my_list.append(res)

    if my_list: #if it is in the list of users

        my_list2=[]
        result2=collection_trusted.find({"username":name})
        for res in result2:
            my_list2.append(res)

        if my_list2: #if the user is already added to trusted
            msg = "Already added"
            return msg
        else:

            trusted_users.append(name)
            post={"_id":my_list[0]["_id"],"name":my_list[0]["name"],"username":my_list[0]["username"]}
            collection_trusted.insert_one(post)
            msg = "Successfuly added"
            return msg

    else:
        msg = "User is not found"
        return msg

```

Figure A.6: The code in app.py file



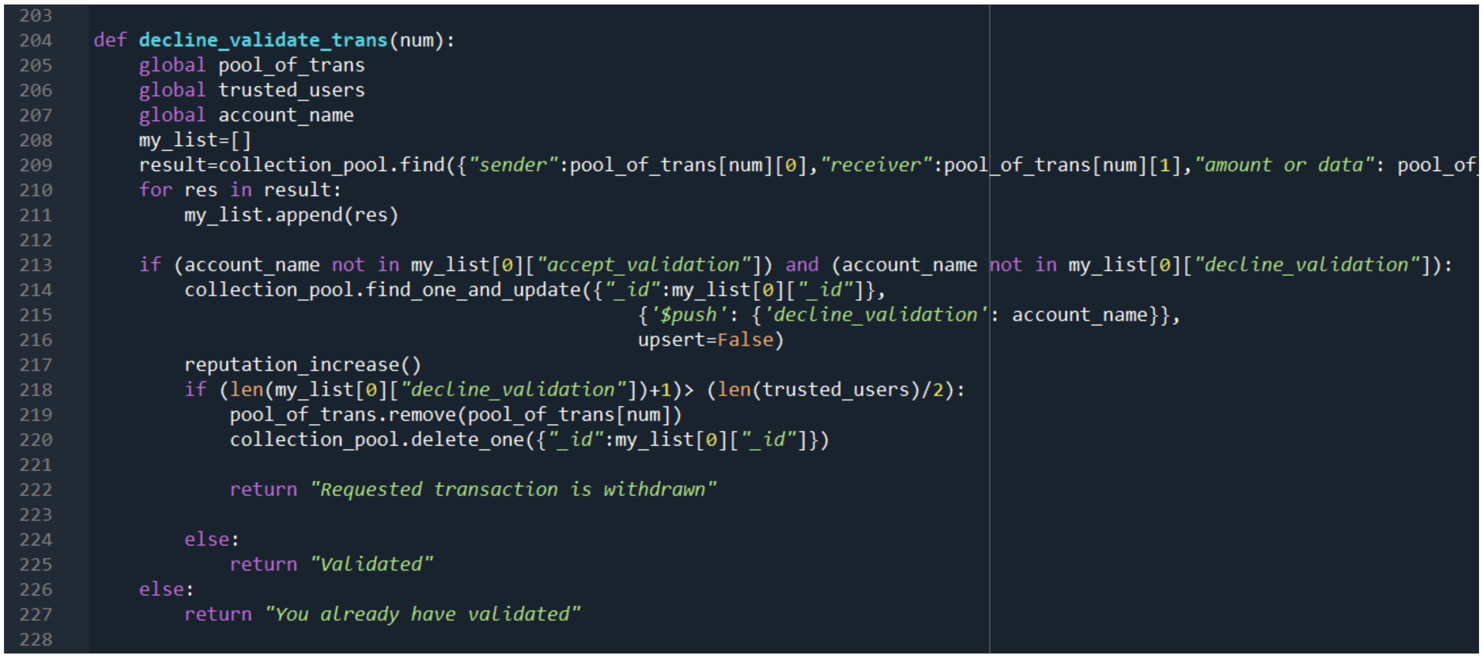
```python
def remove_trusted(name):
    global trusted_users
    my_list=[]
    result=collection_trusted.find({"username":name})
    for res in result:
        my_list.append(res)
    if my_list:
        trusted_users.remove(name)
        collection_trusted.delete_one({"_id":my_list[0]["_id"]})

        return "Successfuly removed"

    else:
        return "Not found"

```

Figure A.7: The code in app.py file



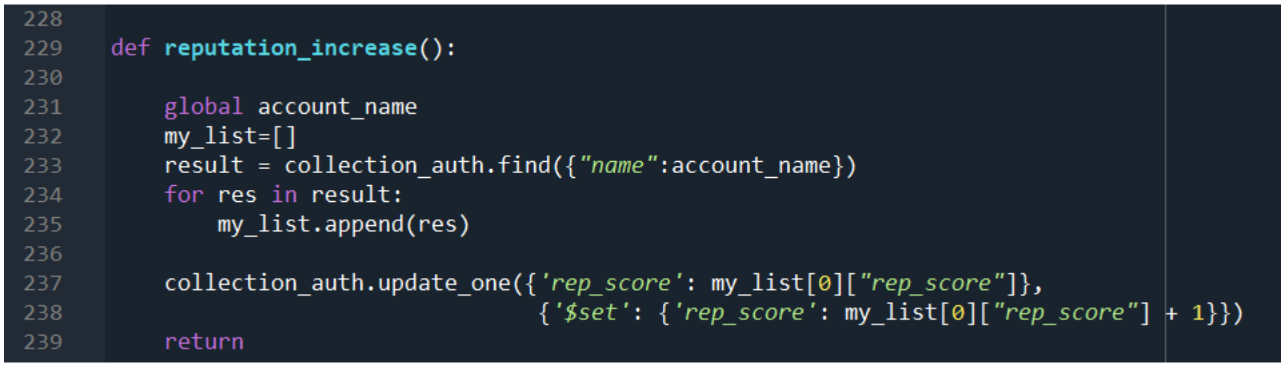
```python
def accept_validate_trans(num):
    global pool_of_trans
    global trusted_users
    global account_name
    my_list=[]
    result=collection_pool.find({"sender":pool_of_trans[num][0],"receiver":pool_of_trans[num][1],"amount or data": pool_of
    for res in result:
        my_list.append(res)

    if (account_name not in my_list[0]["accept_validation"]) and (account_name not in my_list[0]["decline_validation"]):

        collection_pool.find_one_and_update({"_id":my_list[0]["_id"]},
                                            {'$push': {'accept_validation': account_name}},
                                            upsert=False)
        reputation_increase()
        if (len(my_list[0]["accept_validation"])+1)> (len(trusted_users)/2):

            add_to_blockchain(pool_of_trans[num])
            pool_of_trans.remove(pool_of_trans[num])
            collection_pool.delete_one({"_id":my_list[0]["_id"]})

            return "Added to the blockchain"

        else:
            return "Validated"
    else:
        return "You already have validated"

```

Figure A.8: The code in app.py file



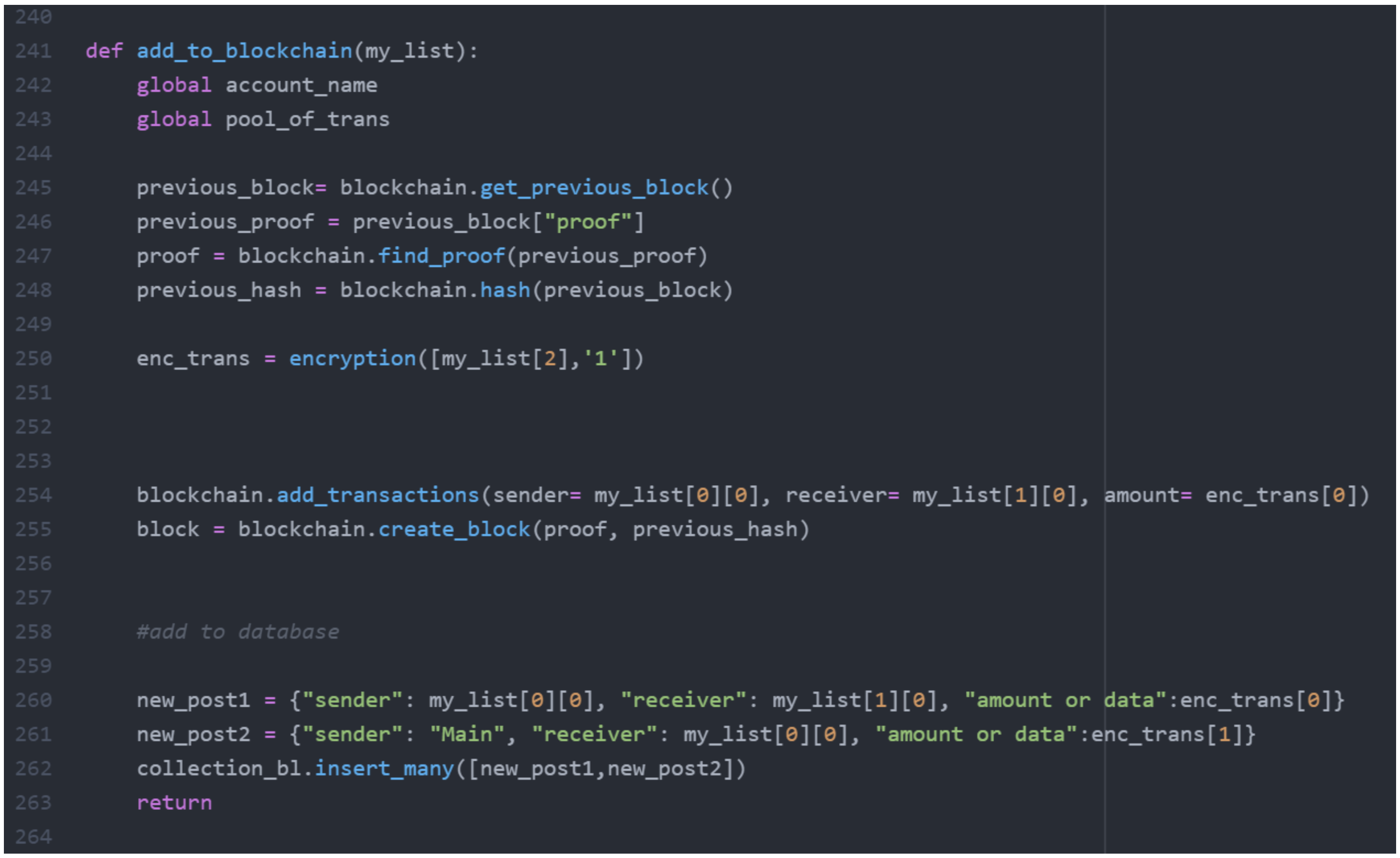
```python

def decline_validate_trans(num):
    global pool_of_trans
    global trusted_users
    global account_name
    my_list=[]
    result=collection_pool.find({"sender":pool_of_trans[num][0],"receiver":pool_of_trans[num][1],"amount or data": pool_of
    for res in result:
        my_list.append(res)

    if (account_name not in my_list[0]["accept_validation"]) and (account_name not in my_list[0]["decline_validation"]):
        collection_pool.find_one_and_update({"_id":my_list[0]["_id"]},
                                            {'$push': {'decline_validation': account_name}},
                                            upsert=False)
        reputation_increase()
        if (len(my_list[0]["decline_validation"])+1)> (len(trusted_users)/2):
            pool_of_trans.remove(pool_of_trans[num])
            collection_pool.delete_one({"_id":my_list[0]["_id"]})

            return "Requested transaction is withdrawn"

        else:
            return "Validated"
    else:
        return "You already have validated"

```

Figure A.9: The code in app.py file



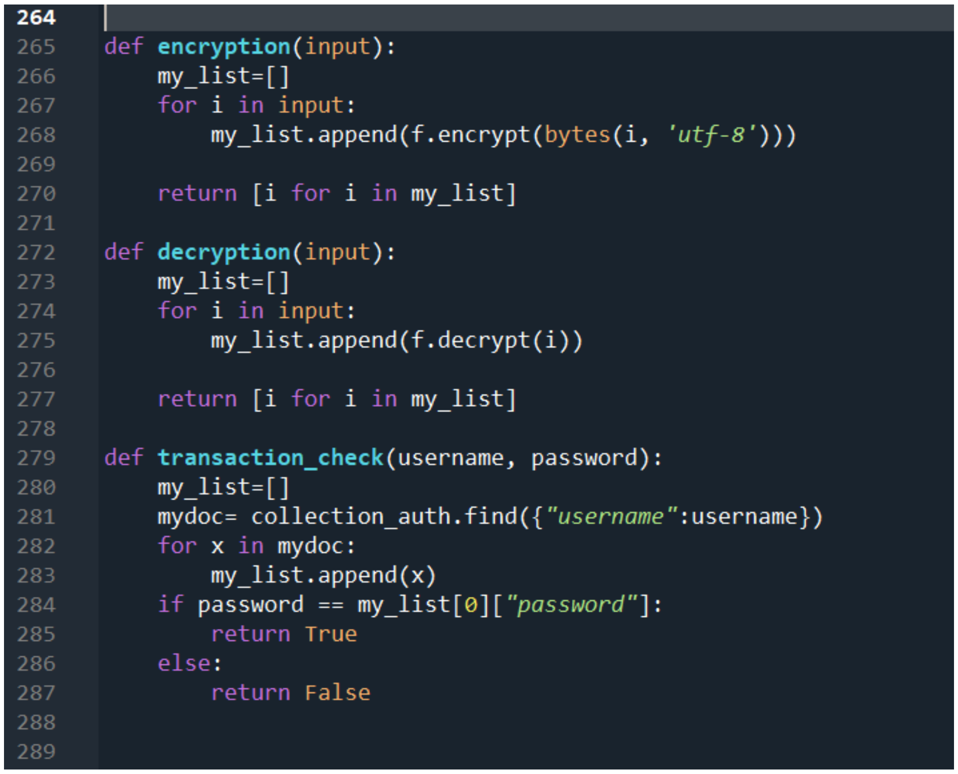
```python

def reputation_increase():

    global account_name
    my_list=[]
    result = collection_auth.find({"name":account_name})
    for res in result:
        my_list.append(res)

    collection_auth.update_one({'rep_score': my_list[0]["rep_score"]},
                               {'$set': {'rep_score': my_list[0]["rep_score"] + 1}})
    return
```

Figure A.10: The code in app.py file



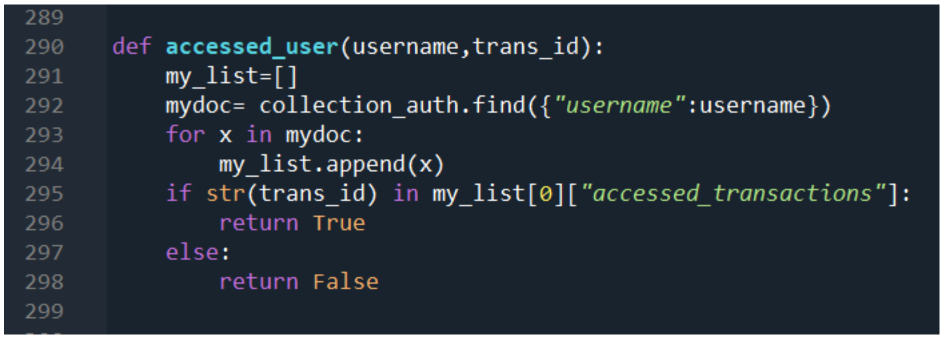
```python

def add_to_blockchain(my_list):
    global account_name
    global pool_of_trans

    previous_block= blockchain.get_previous_block()
    previous_proof = previous_block["proof"]
    proof = blockchain.find_proof(previous_proof)
    previous_hash = blockchain.hash(previous_block)

    enc_trans = encryption([my_list[2],'1'])



    blockchain.add_transactions(sender= my_list[0][0], receiver= my_list[1][0], amount= enc_trans[0])
    block = blockchain.create_block(proof, previous_hash)


    #add to database

    new_post1 = {"sender": my_list[0][0], "receiver": my_list[1][0], "amount or data":enc_trans[0]}
    new_post2 = {"sender": "Main", "receiver": my_list[0][0], "amount or data":enc_trans[1]}
    collection_bl.insert_many([new_post1,new_post2])
    return

```

Figure A.11: The code in app.py file



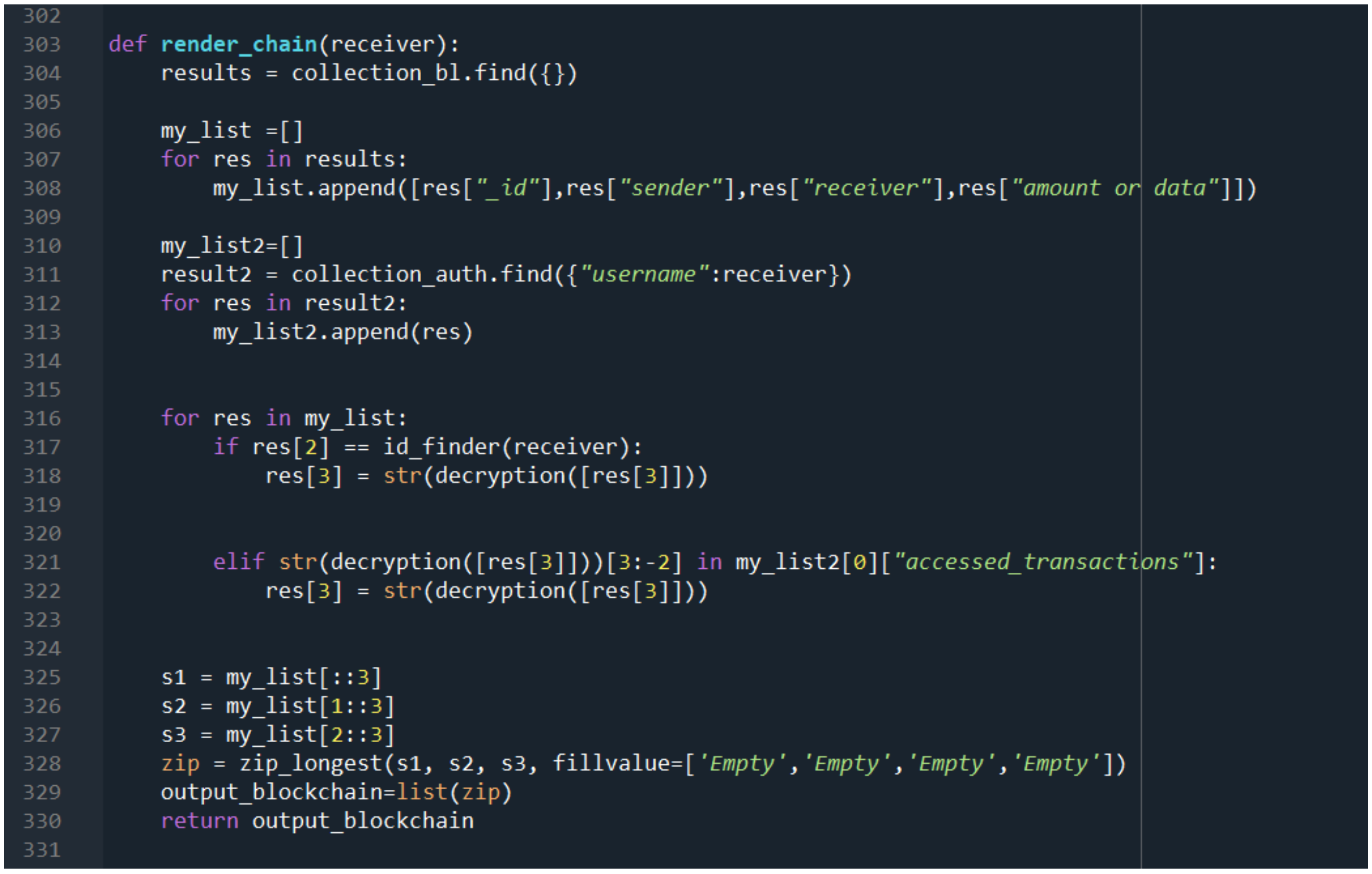
```python

def encryption(input):
    my_list=[]
    for i in input:
        my_list.append(f.encrypt(bytes(i, 'utf-8')))

    return [i for i in my_list]

def decryption(input):
    my_list=[]
    for i in input:
        my_list.append(f.decrypt(i))

    return [i for i in my_list]

def transaction_check(username, password):
    my_list=[]
    mydoc= collection_auth.find({"username":username})
    for x in mydoc:
        my_list.append(x)
    if password == my_list[0]["password"]:
        return True
    else:
        return False


```

Figure A.12: The code in app.py file



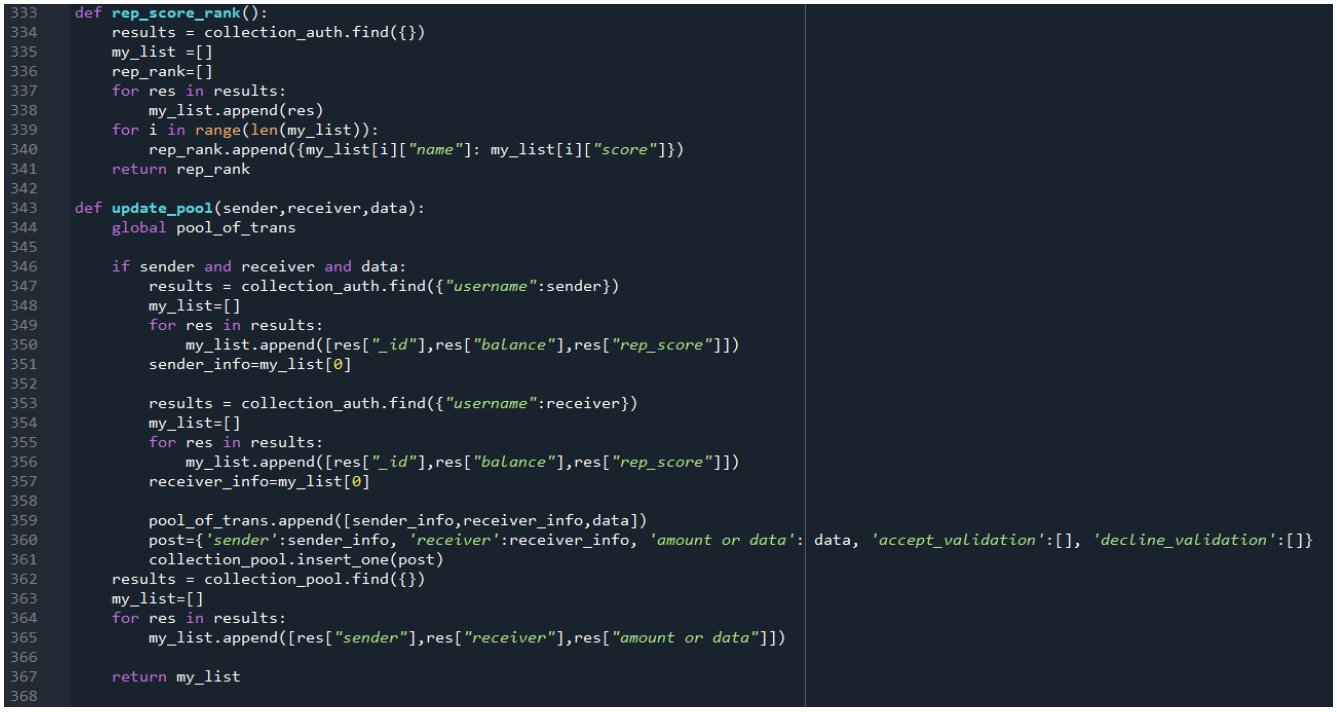
```python

def accessed_user(username,trans_id):
    my_list=[]
    mydoc= collection_auth.find({"username":username})
    for x in mydoc:
        my_list.append(x)
    if str(trans_id) in my_list[0]["accessed_transactions"]:
        return True
    else:
        return False

```

Figure A.13: The code in app.py file



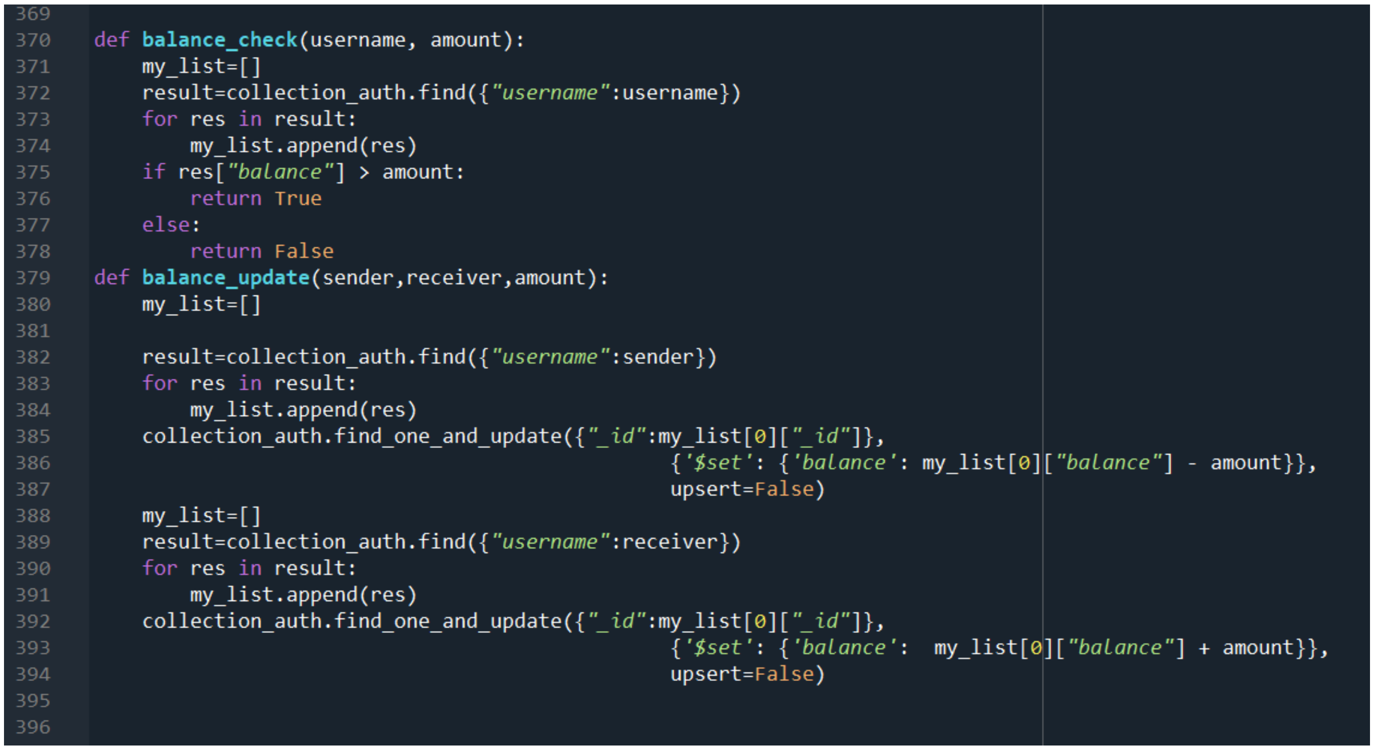
```python

def render_chain(receiver):
    results = collection_bl.find({})

    my_list =[]
    for res in results:
        my_list.append([res["_id"],res["sender"],res["receiver"],res["amount or data"]])

    my_list2=[]
    result2 = collection_auth.find({"username":receiver})
    for res in result2:
        my_list2.append(res)


    for res in my_list:
        if res[2] == id_finder(receiver):
            res[3] = str(decryption([res[3]]))


        elif str(decryption([res[3]]))[3:-2] in my_list2[0]["accessed_transactions"]:
            res[3] = str(decryption([res[3]]))


    s1 = my_list[::3]
    s2 = my_list[1::3]
    s3 = my_list[2::3]
    zip = zip_longest(s1, s2, s3, fillvalue=['Empty','Empty','Empty','Empty'])
    output_blockchain=list(zip)
    return output_blockchain

```

Figure A.14: The code in app.py file



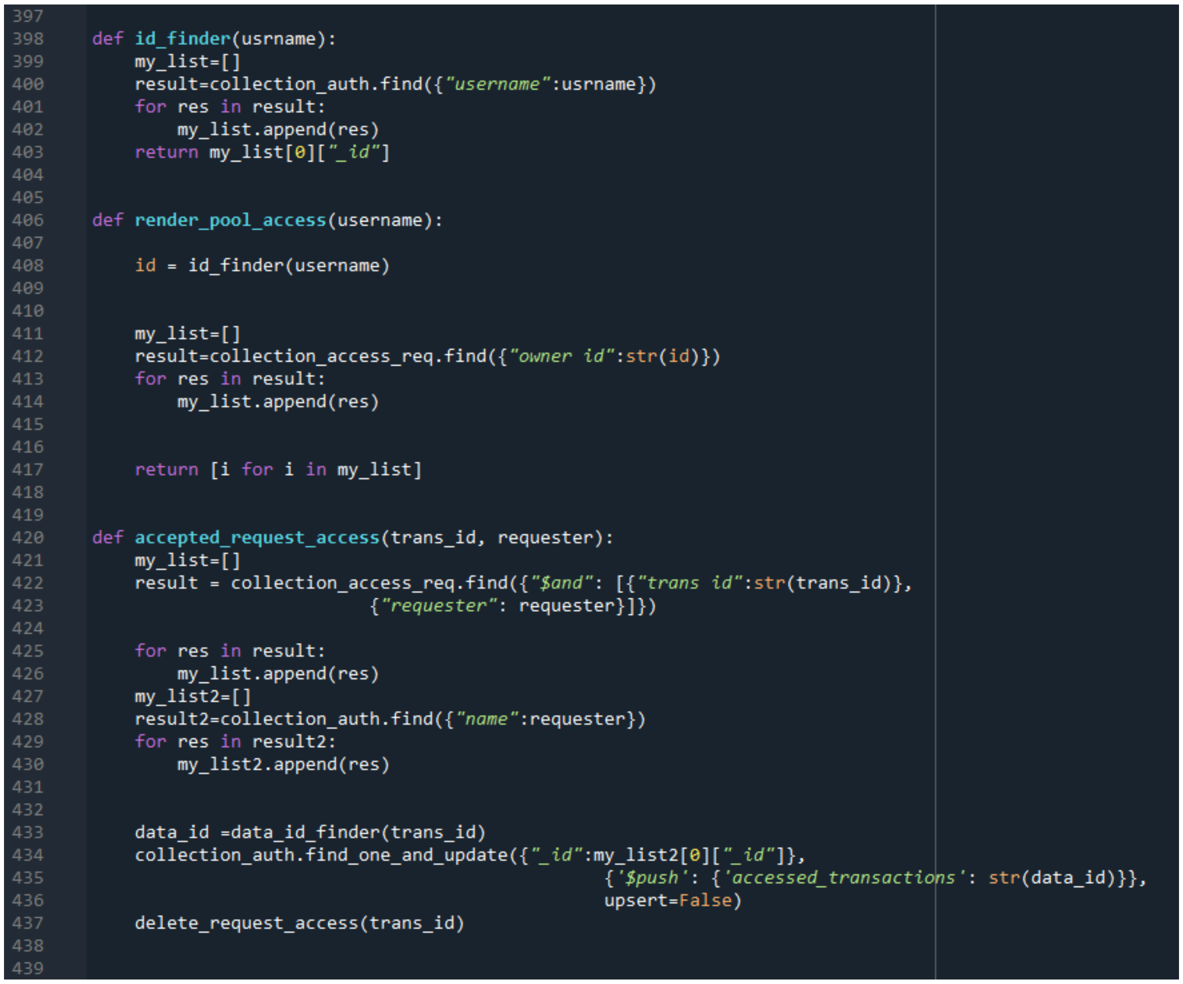
```python
def rep_score_rank():
    results = collection_auth.find({})
    my_list =[]
    rep_rank=[]
    for res in results:
        my_list.append(res)
    for i in range(len(my_list)):
        rep_rank.append({my_list[i]["name"]: my_list[i]["score"]})
    return rep_rank

def update_pool(sender,receiver,data):
    global pool_of_trans

    if sender and receiver and data:
        results = collection_auth.find({"username":sender})
        my_list=[]
        for res in results:
            my_list.append([res["_id"],res["balance"],res["rep_score"]])
        sender_info=my_list[0]

        results = collection_auth.find({"username":receiver})
        my_list=[]
        for res in results:
            my_list.append([res["_id"],res["balance"],res["rep_score"]])
        receiver_info=my_list[0]

        pool_of_trans.append([sender_info,receiver_info,data])
        post={'sender':sender_info, 'receiver':receiver_info, 'amount or data': data, 'accept_validation':[], 'decline_validation':[]}
        collection_pool.insert_one(post)
    results = collection_pool.find({})
    my_list=[]
    for res in results:
        my_list.append([res["sender"],res["receiver"],res["amount or data"]])

    return my_list

```

Figure A.15: The code in app.py file



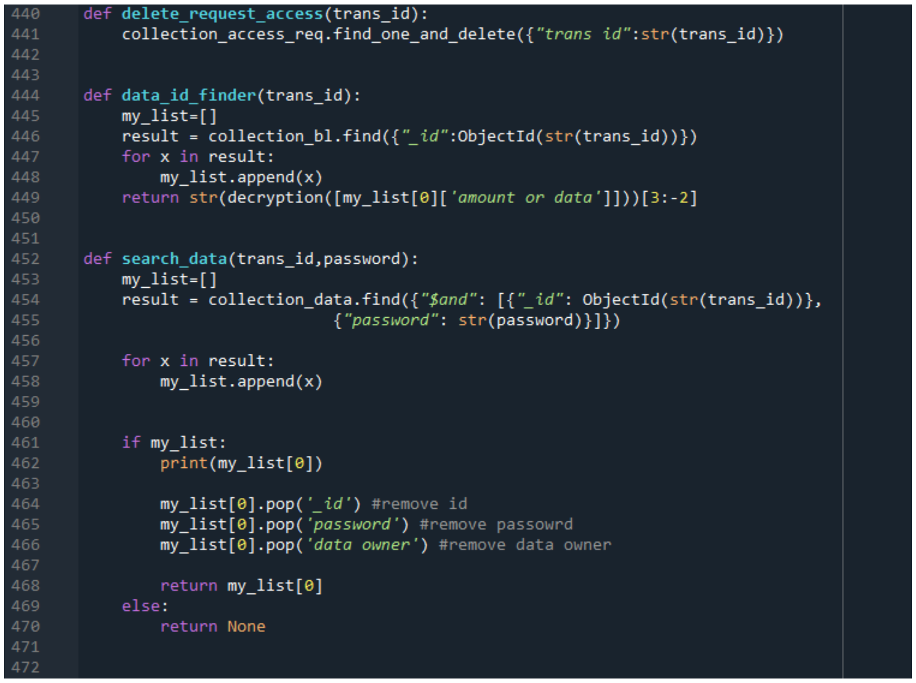
```python

def balance_check(username, amount):
    my_list=[]
    result=collection_auth.find({"username":username})
    for res in result:
        my_list.append(res)
    if res["balance"] > amount:
        return True
    else:
        return False
def balance_update(sender,receiver,amount):
    my_list=[]

    result=collection_auth.find({"username":sender})
    for res in result:
        my_list.append(res)
    collection_auth.find_one_and_update({"_id":my_list[0]["_id"]},
                                        {'$set': {'balance': my_list[0]["balance"] - amount}},
                                        upsert=False)
    my_list=[]
    result=collection_auth.find({"username":receiver})
    for res in result:
        my_list.append(res)
    collection_auth.find_one_and_update({"_id":my_list[0]["_id"]},
                                        {'$set': {'balance':  my_list[0]["balance"] + amount}},
                                        upsert=False)


```

Figure A.16: The code in app.py file



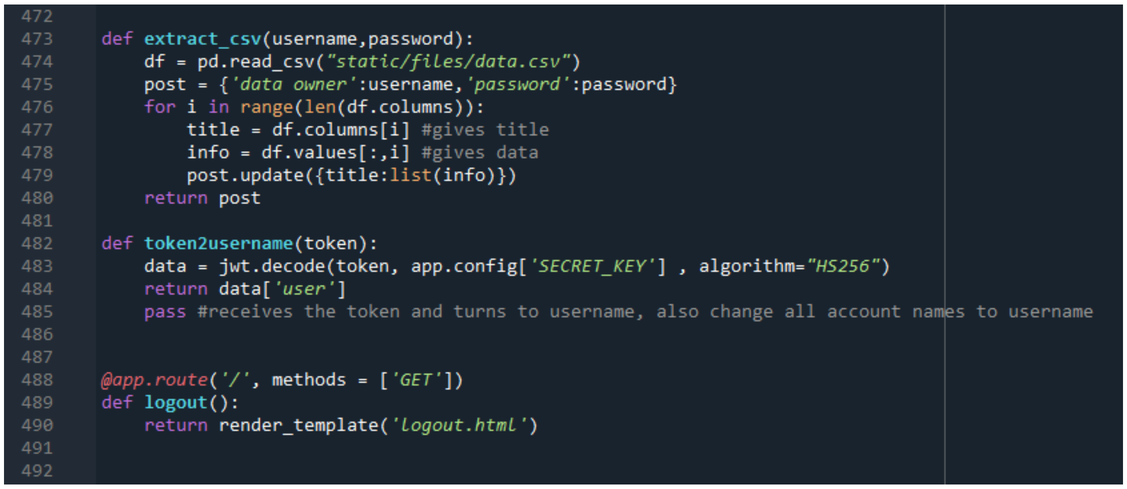
```python

def id_finder(usrname):
    my_list=[]
    result=collection_auth.find({"username":usrname})
    for res in result:
        my_list.append(res)
    return my_list[0]["_id"]


def render_pool_access(username):

    id = id_finder(username)


    my_list=[]
    result=collection_access_req.find({"owner id":str(id)})
    for res in result:
        my_list.append(res)


    return [i for i in my_list]


def accepted_request_access(trans_id, requester):
    my_list=[]
    result = collection_access_req.find({"$and": [{"trans id":str(trans_id)},
                      {"requester": requester}]})

    for res in result:
        my_list.append(res)
    my_list2=[]
    result2=collection_auth.find({"name":requester})
    for res in result2:
        my_list2.append(res)


    data_id =data_id_finder(trans_id)
    collection_auth.find_one_and_update({"_id":my_list2[0]["_id"]},
                                            {'$push': {'accessed_transactions': str(data_id)}},
                                            upsert=False)
    delete_request_access(trans_id)

```

Figure A.17: The code in app.py file



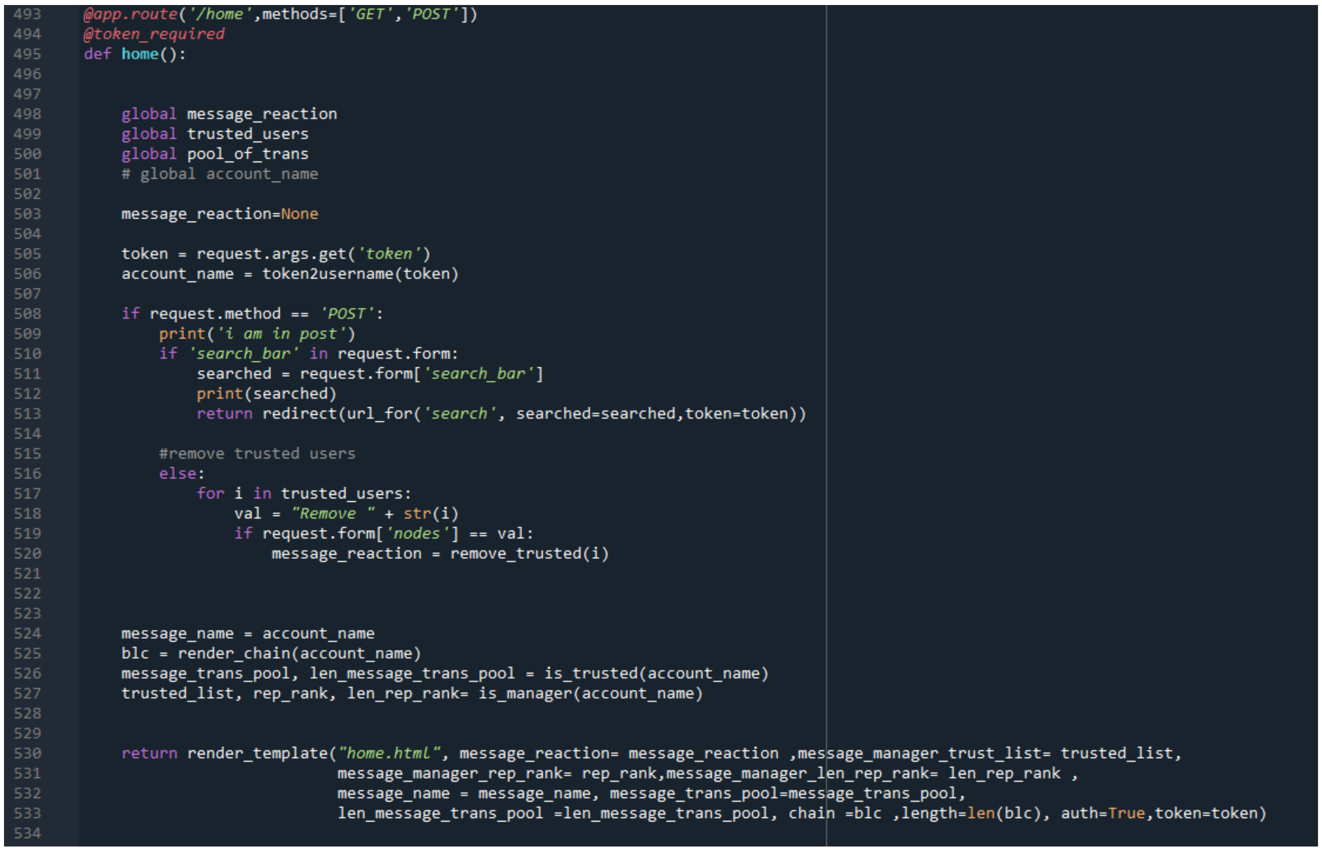
```
def delete_request_access(trans_id):
    collection_access_req.find_one_and_delete({"trans id":str(trans_id)})


def data_id_finder(trans_id):
    my_list=[]
    result = collection_bl.find({"_id":ObjectId(str(trans_id))})
    for x in result:
        my_list.append(x)
    return str(decryption([my_list[0]['amount or data']]))[3:-2]


def search_data(trans_id,password):
    my_list=[]
    result = collection_data.find({"$and": [{"_id": ObjectId(str(trans_id))},
                           {"password": str(password)}]})

    for x in result:
        my_list.append(x)


    if my_list:
        print(my_list[0])

        my_list[0].pop('_id') #remove id
        my_list[0].pop('password') #remove passowrd
        my_list[0].pop('data owner') #remove data owner

        return my_list[0]
    else:
        return None

```

Figure A.18: The code in app.py file



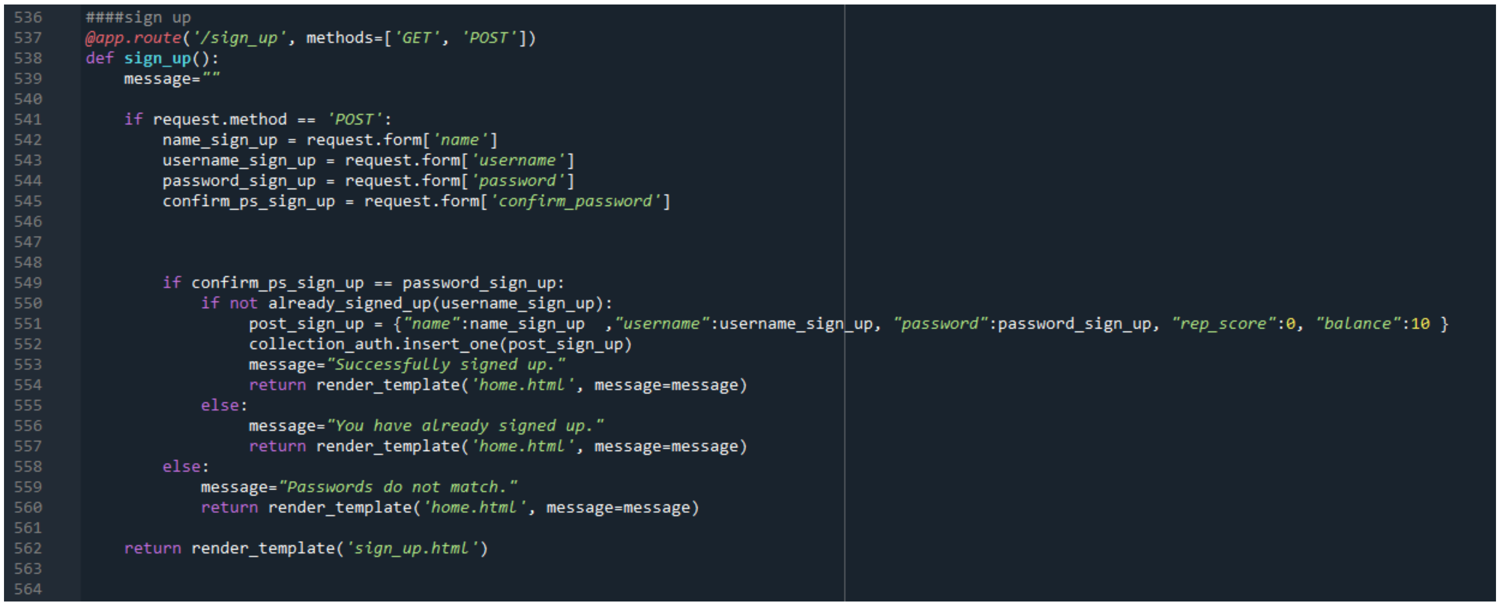
```

def extract_csv(username,password):
    df = pd.read_csv("static/files/data.csv")
    post = {'data owner':username,'password':password}
    for i in range(len(df.columns)):
        title = df.columns[i] #gives title
        info = df.values[:,i] #gives data
        post.update({title:list(info)})
    return post

def token2username(token):
    data = jwt.decode(token, app.config['SECRET_KEY'] , algorithm="HS256")
    return data['user']
    pass #receives the token and turns to username, also change all account names to username


@app.route('/', methods = ['GET'])
def logout():
    return render_template('logout.html')

```

Figure A.19: The code in app.py file



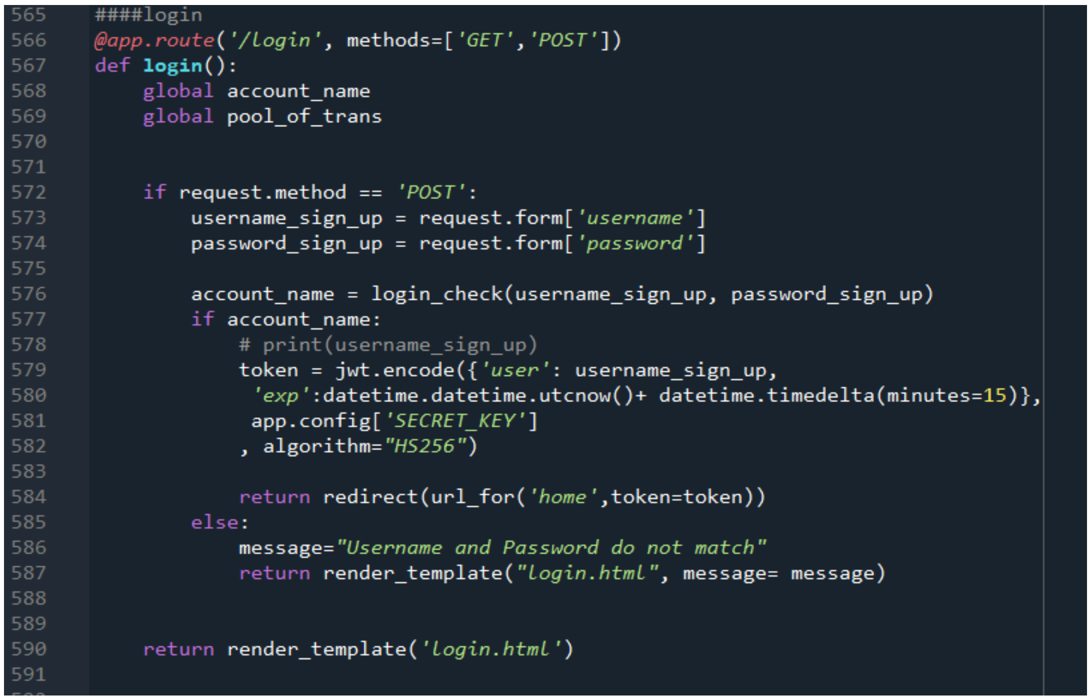
```python
@app.route('/home',methods=['GET','POST'])
@token_required
def home():


    global message_reaction
    global trusted_users
    global pool_of_trans
    # global account_name

    message_reaction=None

    token = request.args.get('token')
    account_name = token2username(token)

    if request.method == 'POST':
        print('i am in post')
        if 'search_bar' in request.form:
            searched = request.form['search_bar']
            print(searched)
            return redirect(url_for('search', searched=searched,token=token))

        #remove trusted users
        else:
            for i in trusted_users:
                val = "Remove " + str(i)
                if request.form['nodes'] == val:
                    message_reaction = remove_trusted(i)



    message_name = account_name
    blc = render_chain(account_name)
    message_trans_pool, len_message_trans_pool = is_trusted(account_name)
    trusted_list, rep_rank, len_rep_rank= is_manager(account_name)


    return render_template("home.html", message_reaction= message_reaction ,message_manager_trust_list= trusted_list,
                           message_manager_rep_rank= rep_rank,message_manager_len_rep_rank= len_rep_rank ,
                           message_name = message_name, message_trans_pool=message_trans_pool,
                           len_message_trans_pool =len_message_trans_pool, chain =blc ,length=len(blc), auth=True,token=token)

```

Figure A.20: The code in app.py file



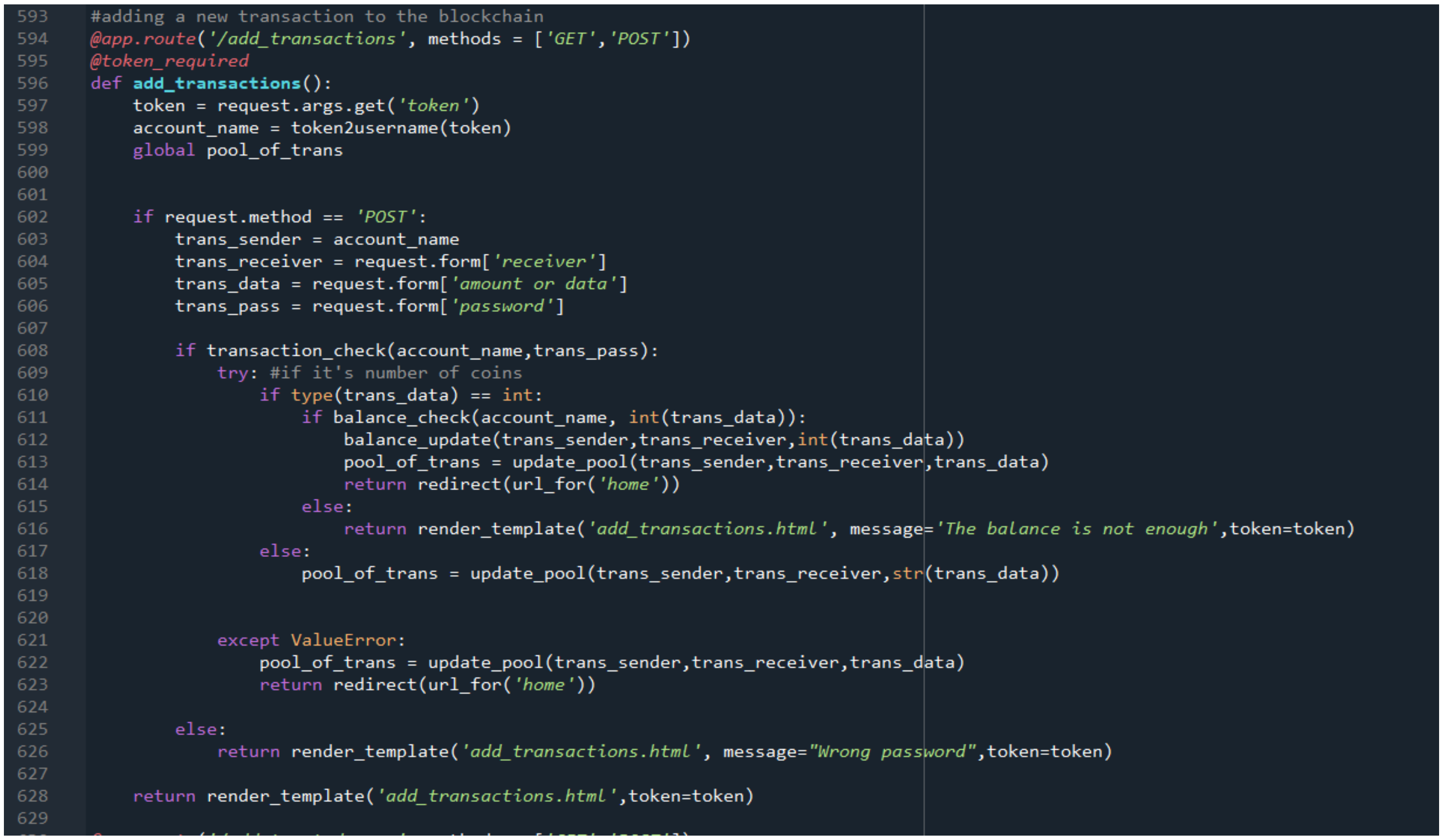
```python
####sign up
@app.route('/sign_up', methods=['GET', 'POST'])
def sign_up():
    message=""

    if request.method == 'POST':
        name_sign_up = request.form['name']
        username_sign_up = request.form['username']
        password_sign_up = request.form['password']
        confirm_ps_sign_up = request.form['confirm_password']



        if confirm_ps_sign_up == password_sign_up:
            if not already_signed_up(username_sign_up):
                post_sign_up = {"name":name_sign_up  ,"username":username_sign_up, "password":password_sign_up, "rep_score":0, "balance":10 }
                collection_auth.insert_one(post_sign_up)
                message="Successfully signed up."
                return render_template('home.html', message=message)
            else:
                message="You have already signed up."
                return render_template('home.html', message=message)
        else:
            message="Passwords do not match."
            return render_template('home.html', message=message)

    return render_template('sign_up.html')

```

Figure A.21: The code in app.py file



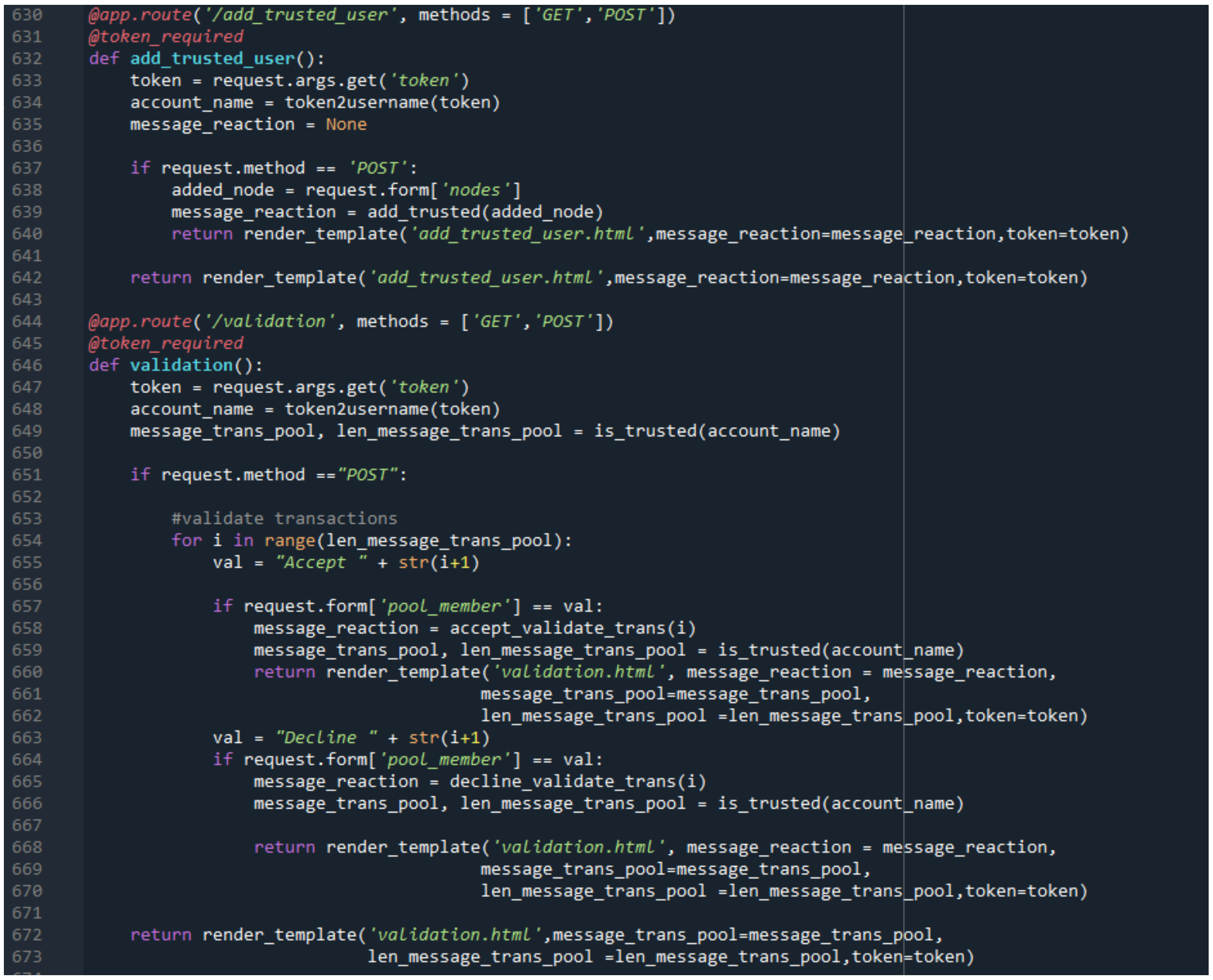
```
####login
@app.route('/login', methods=['GET','POST'])
def login():
    global account_name
    global pool_of_trans


    if request.method == 'POST':
        username_sign_up = request.form['username']
        password_sign_up = request.form['password']

        account_name = login_check(username_sign_up, password_sign_up)
        if account_name:
            # print(username_sign_up)
            token = jwt.encode({'user': username_sign_up,
             'exp':datetime.datetime.utcnow()+ datetime.timedelta(minutes=15)},
             app.config['SECRET_KEY']
            , algorithm="HS256")

            return redirect(url_for('home',token=token))
        else:
            message="Username and Password do not match"
            return render_template("login.html", message= message)


    return render_template('login.html')
```

Figure A.22: The code in app.py file



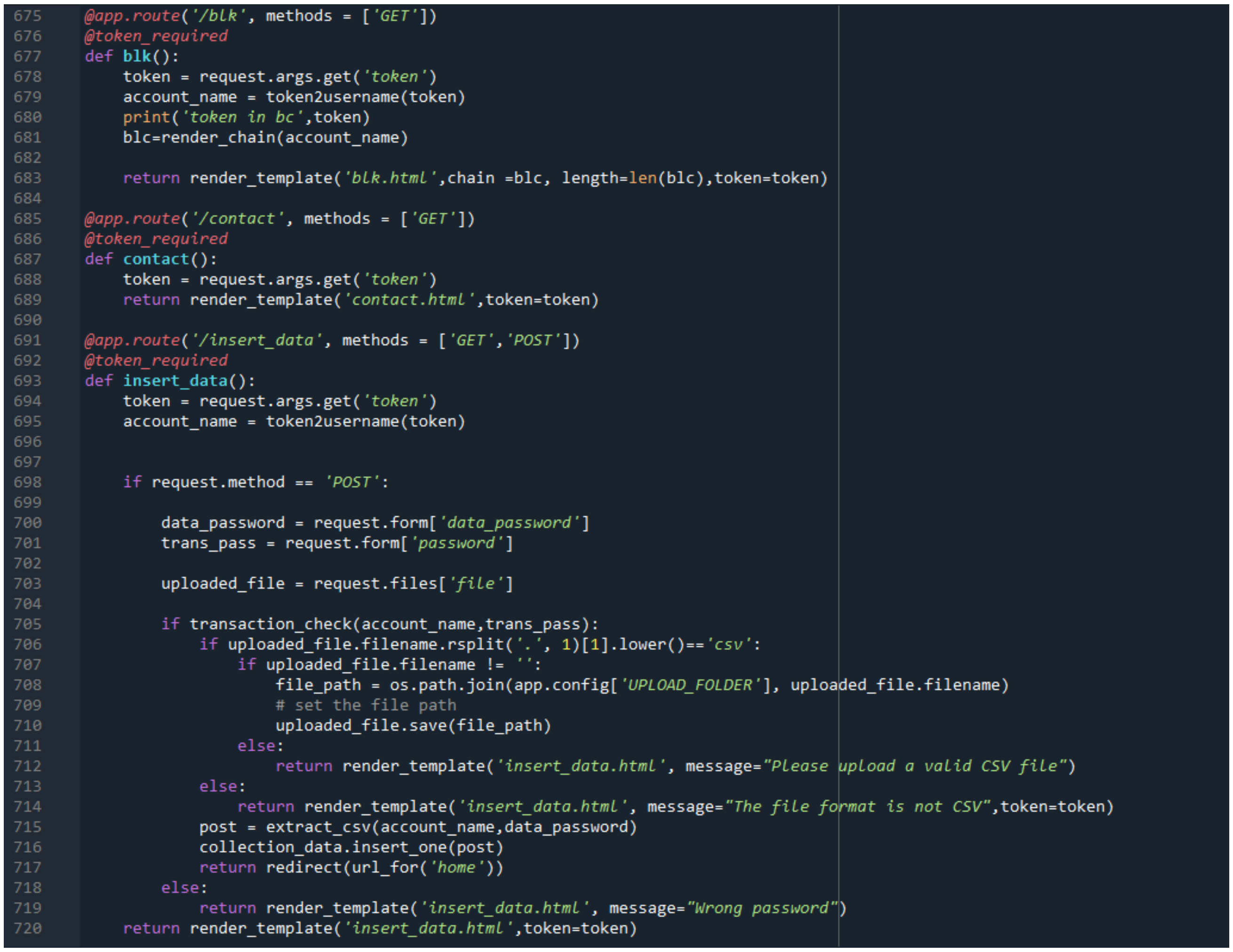
```python
#adding a new transaction to the blockchain
@app.route('/add_transactions', methods = ['GET','POST'])
@token_required
def add_transactions():
    token = request.args.get('token')
    account_name = token2username(token)
    global pool_of_trans


    if request.method == 'POST':
        trans_sender = account_name
        trans_receiver = request.form['receiver']
        trans_data = request.form['amount or data']
        trans_pass = request.form['password']

        if transaction_check(account_name,trans_pass):
            try: #if it's number of coins
                if type(trans_data) == int:
                    if balance_check(account_name, int(trans_data)):
                        balance_update(trans_sender,trans_receiver,int(trans_data))
                        pool_of_trans = update_pool(trans_sender,trans_receiver,trans_data)
                        return redirect(url_for('home'))
                    else:
                        return render_template('add_transactions.html', message='The balance is not enough',token=token)
                else:
                    pool_of_trans = update_pool(trans_sender,trans_receiver,str(trans_data))


            except ValueError:
                pool_of_trans = update_pool(trans_sender,trans_receiver,trans_data)
                return redirect(url_for('home'))

        else:
            return render_template('add_transactions.html', message="Wrong password",token=token)

    return render_template('add_transactions.html',token=token)
```

Figure A.23: The code in app.py file



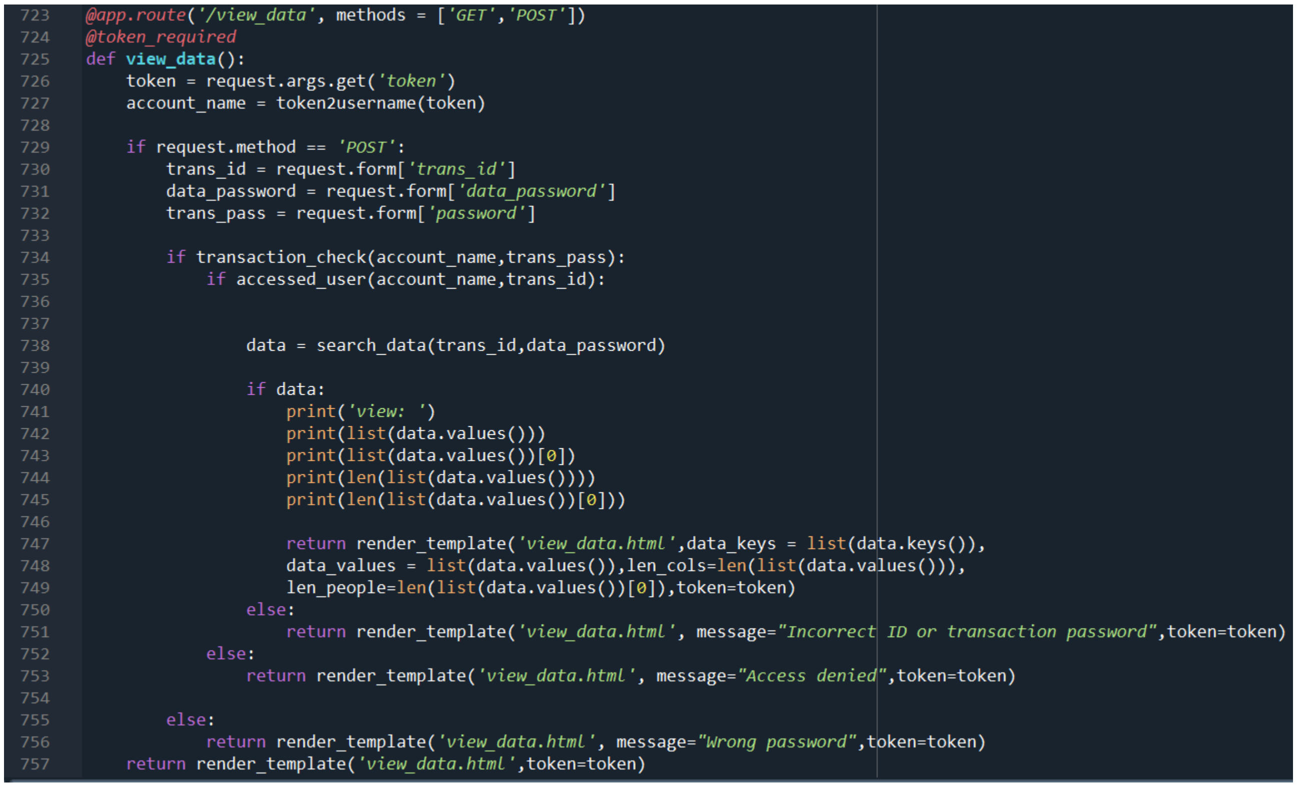
```python
@app.route('/add_trusted_user', methods = ['GET','POST'])
@token_required
def add_trusted_user():
    token = request.args.get('token')
    account_name = token2username(token)
    message_reaction = None

    if request.method == 'POST':
        added_node = request.form['nodes']
        message_reaction = add_trusted(added_node)
        return render_template('add_trusted_user.html',message_reaction=message_reaction,token=token)

    return render_template('add_trusted_user.html',message_reaction=message_reaction,token=token)

@app.route('/validation', methods = ['GET','POST'])
@token_required
def validation():
    token = request.args.get('token')
    account_name = token2username(token)
    message_trans_pool, len_message_trans_pool = is_trusted(account_name)

    if request.method =="POST":

        #validate transactions
        for i in range(len_message_trans_pool):
            val = "Accept " + str(i+1)

            if request.form['pool_member'] == val:
                message_reaction = accept_validate_trans(i)
                message_trans_pool, len_message_trans_pool = is_trusted(account_name)
                return render_template('validation.html', message_reaction = message_reaction,
                                      message_trans_pool=message_trans_pool,
                                      len_message_trans_pool =len_message_trans_pool,token=token)
            val = "Decline " + str(i+1)
            if request.form['pool_member'] == val:
                message_reaction = decline_validate_trans(i)
                message_trans_pool, len_message_trans_pool = is_trusted(account_name)

                return render_template('validation.html', message_reaction = message_reaction,
                                      message_trans_pool=message_trans_pool,
                                      len_message_trans_pool =len_message_trans_pool,token=token)

    return render_template('validation.html',message_trans_pool=message_trans_pool,
                           len_message_trans_pool =len_message_trans_pool,token=token)
```

Figure A.24: The code in app.py file



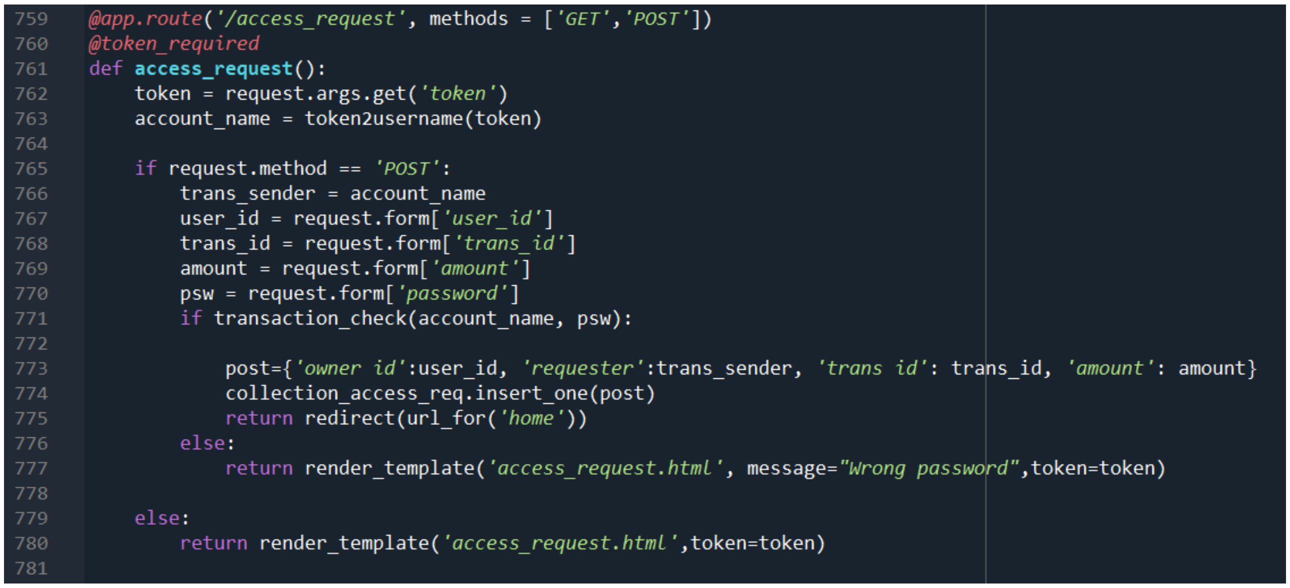
```python
@app.route('/blk', methods = ['GET'])
@token_required
def blk():
    token = request.args.get('token')
    account_name = token2username(token)
    print('token in bc',token)
    blc=render_chain(account_name)

    return render_template('blk.html',chain =blc, length=len(blc),token=token)

@app.route('/contact', methods = ['GET'])
@token_required
def contact():
    token = request.args.get('token')
    return render_template('contact.html',token=token)

@app.route('/insert_data', methods = ['GET','POST'])
@token_required
def insert_data():
    token = request.args.get('token')
    account_name = token2username(token)


    if request.method == 'POST':

        data_password = request.form['data_password']
        trans_pass = request.form['password']

        uploaded_file = request.files['file']

        if transaction_check(account_name,trans_pass):
            if uploaded_file.filename.rsplit('.', 1)[1].lower()=='csv':
                if uploaded_file.filename != '':
                    file_path = os.path.join(app.config['UPLOAD_FOLDER'], uploaded_file.filename)
                    # set the file path
                    uploaded_file.save(file_path)
                else:
                    return render_template('insert_data.html', message="Please upload a valid CSV file")
            else:
                return render_template('insert_data.html', message="The file format is not CSV",token=token)
            post = extract_csv(account_name,data_password)
            collection_data.insert_one(post)
            return redirect(url_for('home'))
        else:
            return render_template('insert_data.html', message="Wrong password")
    return render_template('insert_data.html',token=token)
```

Figure A.25: The code in app.py file



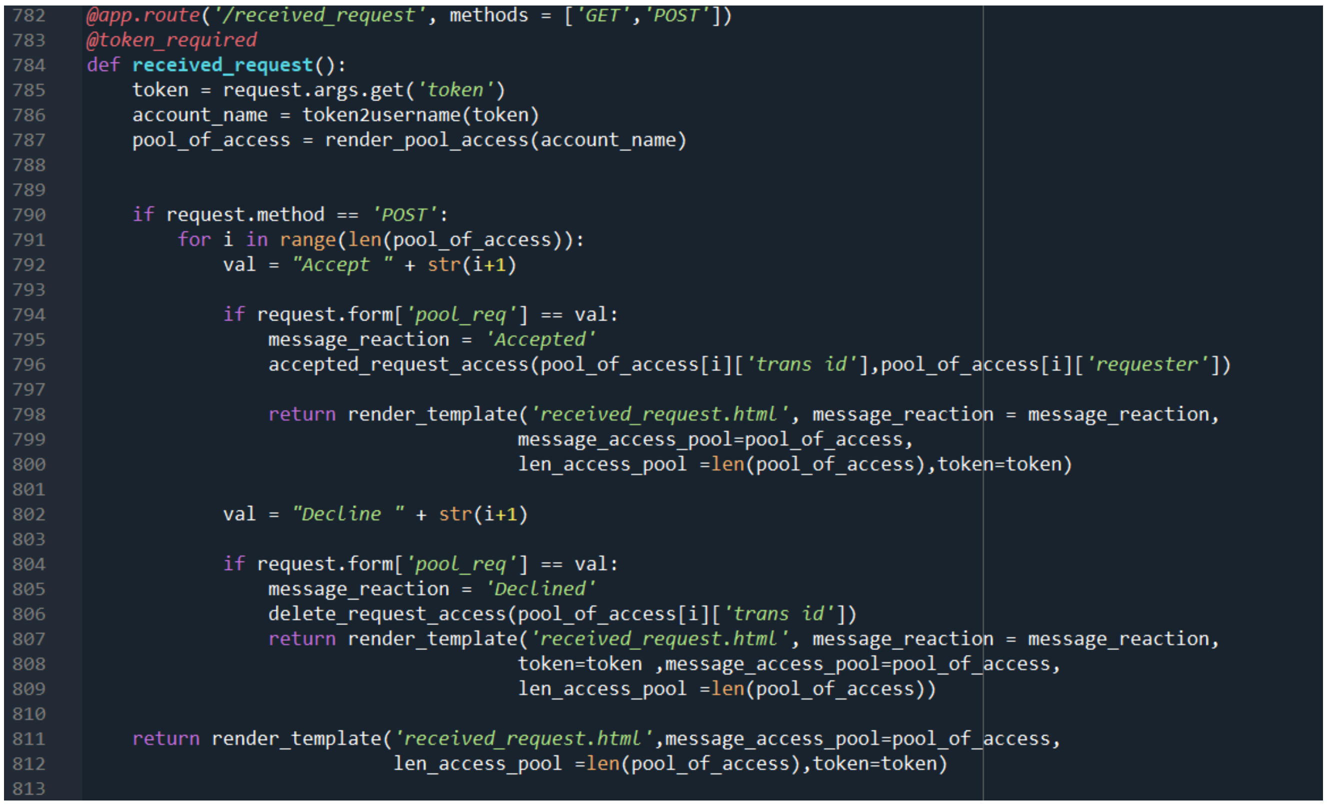
```python
@app.route('/view_data', methods = ['GET','POST'])
@token_required
def view_data():
    token = request.args.get('token')
    account_name = token2username(token)

    if request.method == 'POST':
        trans_id = request.form['trans_id']
        data_password = request.form['data_password']
        trans_pass = request.form['password']

        if transaction_check(account_name,trans_pass):
            if accessed_user(account_name,trans_id):


                data = search_data(trans_id,data_password)

                if data:
                    print('view: ')
                    print(list(data.values()))
                    print(list(data.values())[0])
                    print(len(list(data.values())))
                    print(len(list(data.values())[0]))

                    return render_template('view_data.html',data_keys = list(data.keys()),
                    data_values = list(data.values()),len_cols=len(list(data.values())),
                    len_people=len(list(data.values())[0]),token=token)
                else:
                    return render_template('view_data.html', message="Incorrect ID or transaction password",token=token)
            else:
                return render_template('view_data.html', message="Access denied",token=token)

        else:
            return render_template('view_data.html', message="Wrong password",token=token)
    return render_template('view_data.html',token=token)
```

Figure A.26: The code in app.py file



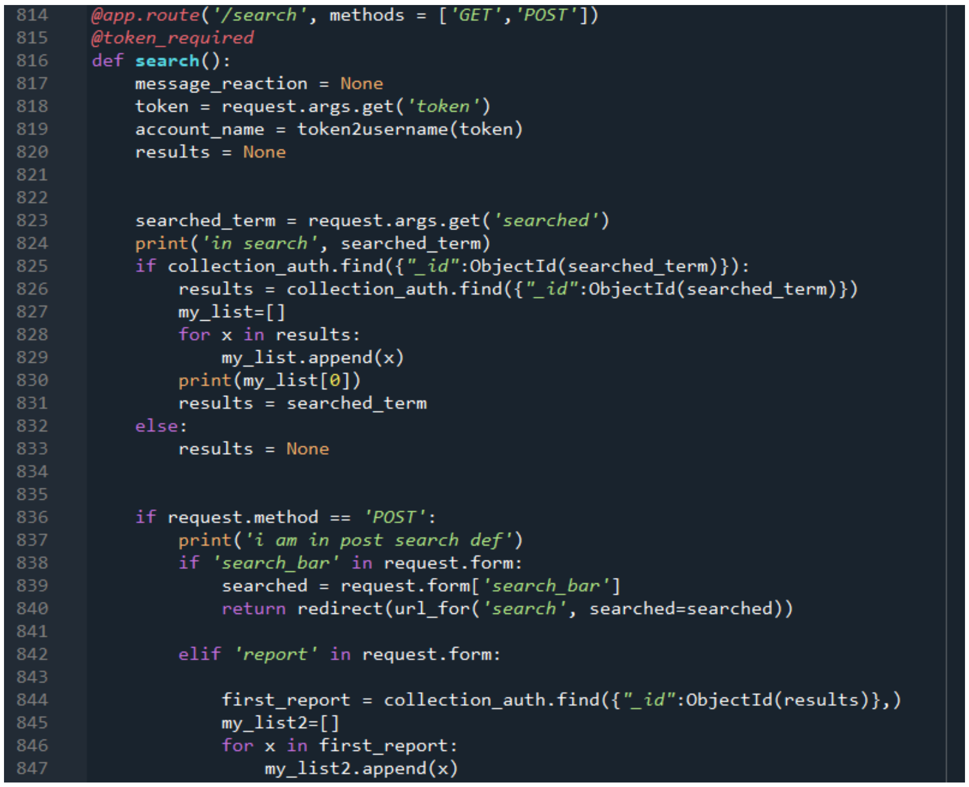
```python
@app.route('/access_request', methods = ['GET','POST'])
@token_required
def access_request():
    token = request.args.get('token')
    account_name = token2username(token)

    if request.method == 'POST':
        trans_sender = account_name
        user_id = request.form['user_id']
        trans_id = request.form['trans_id']
        amount = request.form['amount']
        psw = request.form['password']
        if transaction_check(account_name, psw):

            post={'owner id':user_id, 'requester':trans_sender, 'trans id': trans_id, 'amount': amount}
            collection_access_req.insert_one(post)
            return redirect(url_for('home'))
        else:
            return render_template('access_request.html', message="Wrong password",token=token)

    else:
        return render_template('access_request.html',token=token)

```

Figure A.27: The code in app.py file



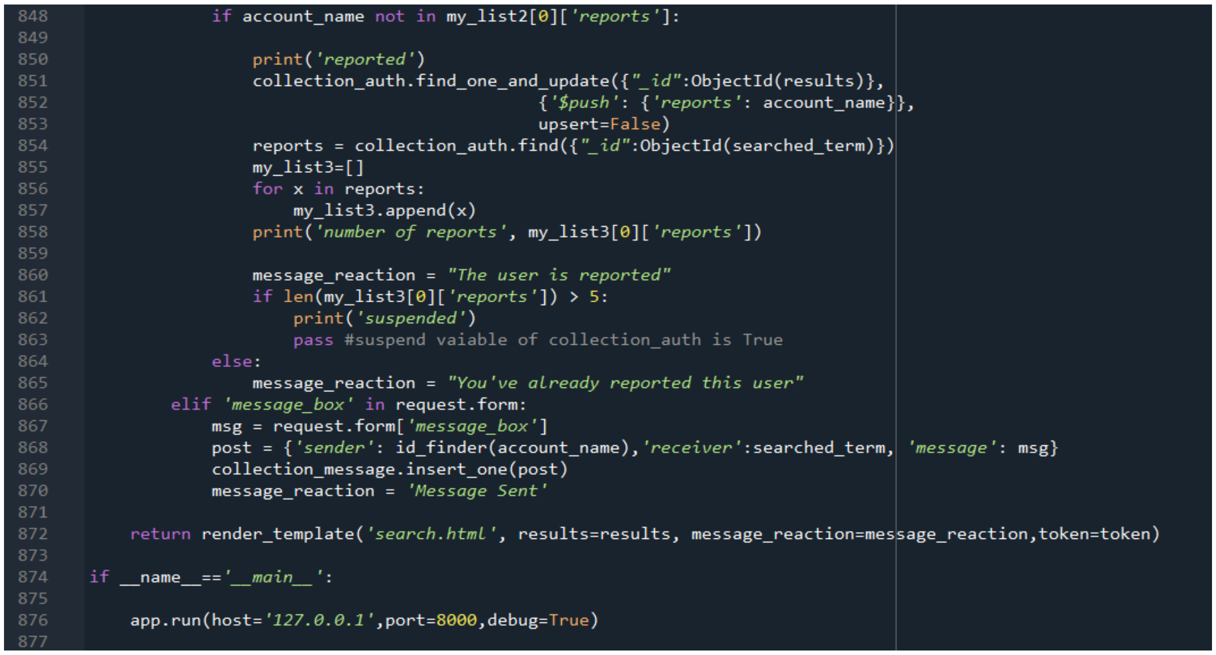
```python
@app.route('/received_request', methods = ['GET','POST'])
@token_required
def received_request():
    token = request.args.get('token')
    account_name = token2username(token)
    pool_of_access = render_pool_access(account_name)


    if request.method == 'POST':
        for i in range(len(pool_of_access)):
            val = "Accept " + str(i+1)

            if request.form['pool_req'] == val:
                message_reaction = 'Accepted'
                accepted_request_access(pool_of_access[i]['trans id'],pool_of_access[i]['requester'])

                return render_template('received_request.html', message_reaction = message_reaction,
                                       message_access_pool=pool_of_access,
                                       len_access_pool =len(pool_of_access),token=token)

            val = "Decline " + str(i+1)

            if request.form['pool_req'] == val:
                message_reaction = 'Declined'
                delete_request_access(pool_of_access[i]['trans id'])
                return render_template('received_request.html', message_reaction = message_reaction,
                                       token=token ,message_access_pool=pool_of_access,
                                       len_access_pool =len(pool_of_access))

    return render_template('received_request.html',message_access_pool=pool_of_access,
                           len_access_pool =len(pool_of_access),token=token)
```

Figure A.28: The code in app.py file



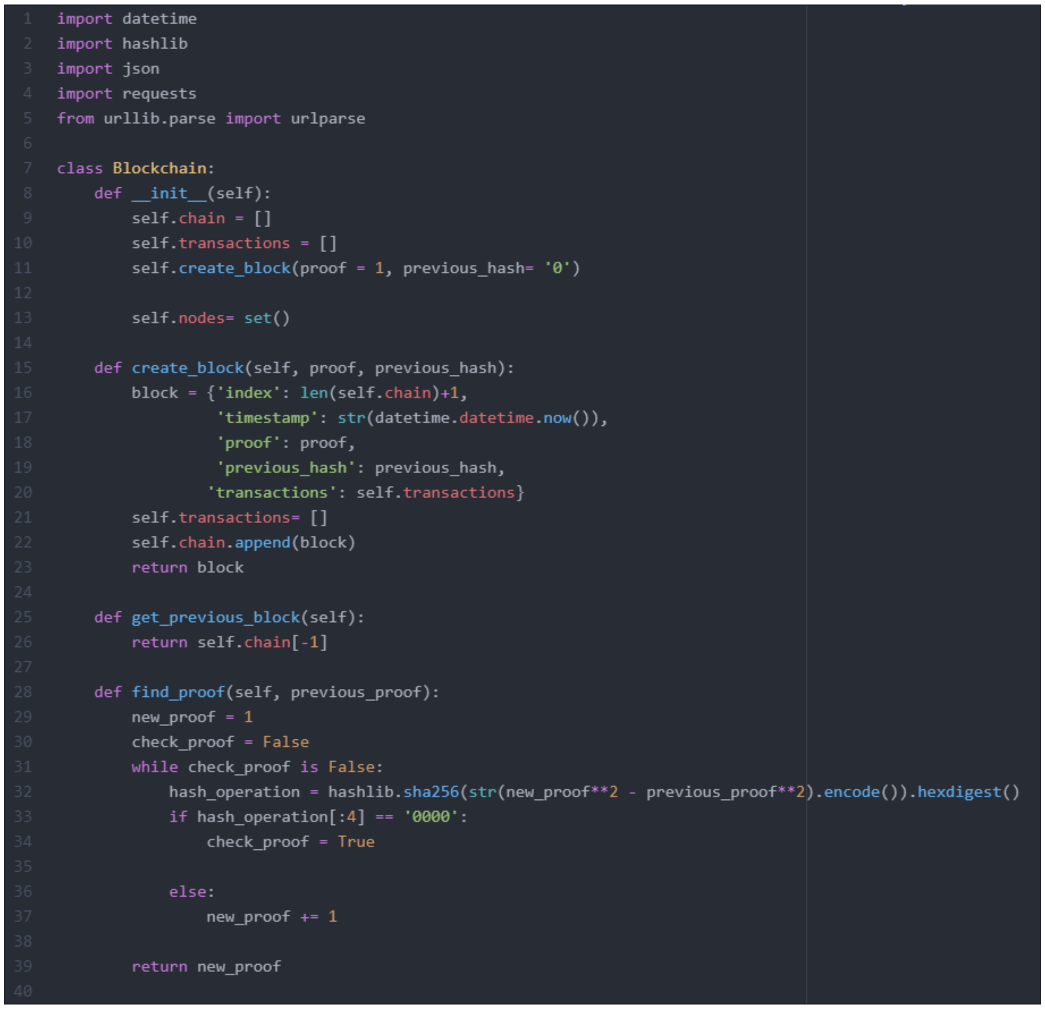
```python
@app.route('/search', methods = ['GET','POST'])
@token_required
def search():
    message_reaction = None
    token = request.args.get('token')
    account_name = token2username(token)
    results = None


    searched_term = request.args.get('searched')
    print('in search', searched_term)
    if collection_auth.find({"_id":ObjectId(searched_term)}):
        results = collection_auth.find({"_id":ObjectId(searched_term)})
        my_list=[]
        for x in results:
            my_list.append(x)
        print(my_list[0])
        results = searched_term
    else:
        results = None


    if request.method == 'POST':
        print('i am in post search def')
        if 'search_bar' in request.form:
            searched = request.form['search_bar']
            return redirect(url_for('search', searched=searched))

        elif 'report' in request.form:

            first_report = collection_auth.find({"_id":ObjectId(results)},)
            my_list2=[]
            for x in first_report:
                my_list2.append(x)
```

Figure A.29: The code in app.py file



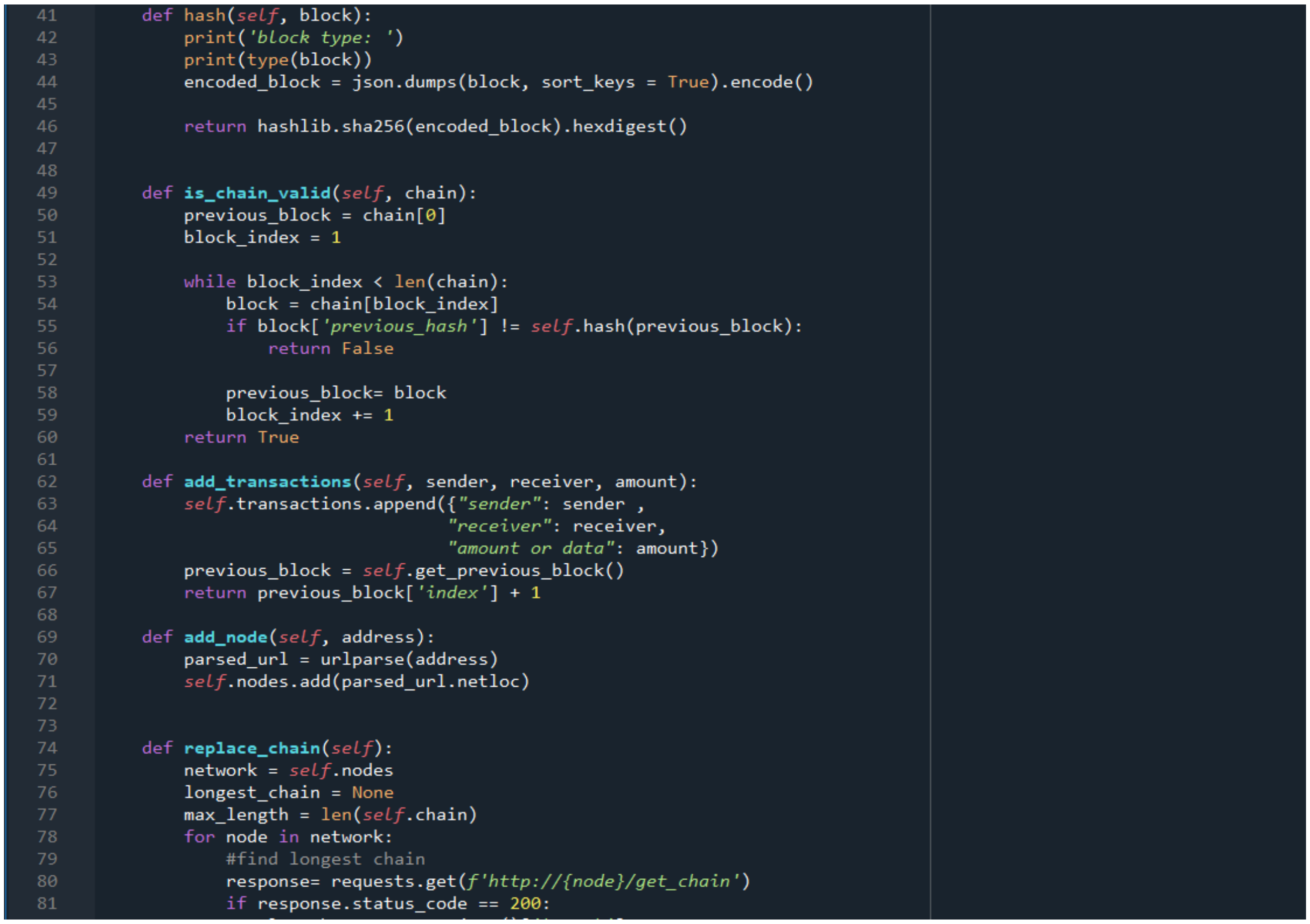
```python
            if account_name not in my_list2[0]['reports']:

                print('reported')
                collection_auth.find_one_and_update({"_id":ObjectId(results)},
                                            {'$push': {'reports': account_name}},
                                            upsert=False)
                reports = collection_auth.find({"_id":ObjectId(searched_term)})
                my_list3=[]
                for x in reports:
                    my_list3.append(x)
                print('number of reports', my_list3[0]['reports'])

                message_reaction = "The user is reported"
                if len(my_list3[0]['reports']) > 5:
                    print('suspended')
                    pass #suspend vaiable of collection_auth is True
            else:
                message_reaction = "You've already reported this user"
        elif 'message_box' in request.form:
            msg = request.form['message_box']
            post = {'sender': id_finder(account_name),'receiver':searched_term, 'message': msg}
            collection_message.insert_one(post)
            message_reaction = 'Message Sent'

    return render_template('search.html', results=results, message_reaction=message_reaction,token=token)

if __name__=='__main__':

    app.run(host='127.0.0.1',port=8000,debug=True)
```

Figure A.30: The code in app.py file



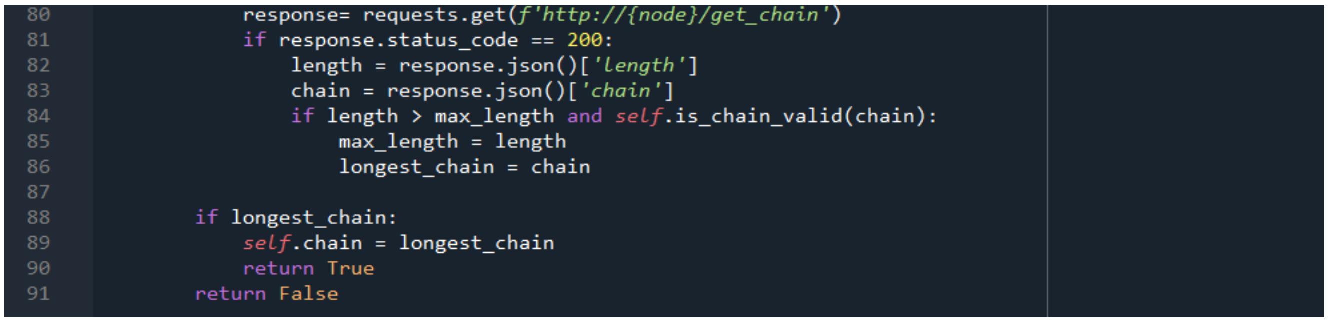
```
import datetime
import hashlib
import json
import requests
from urllib.parse import urlparse

class Blockchain:
    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(proof = 1, previous_hash= '0')

        self.nodes= set()

    def create_block(self, proof, previous_hash):
        block = {'index': len(self.chain)+1,
                 'timestamp': str(datetime.datetime.now()),
                 'proof': proof,
                 'previous_hash': previous_hash,
                'transactions': self.transactions}
        self.transactions= []
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def find_proof(self, previous_proof):
        new_proof = 1
        check_proof = False
        while check_proof is False:
            hash_operation = hashlib.sha256(str(new_proof**2 - previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] == '0000':
                check_proof = True

            else:
                new_proof += 1

        return new_proof

```

Figure A.31: The code in blockchain.py file



```python
    def hash(self, block):
        print('block type: ')
        print(type(block))
        encoded_block = json.dumps(block, sort_keys = True).encode()

        return hashlib.sha256(encoded_block).hexdigest()


    def is_chain_valid(self, chain):
        previous_block = chain[0]
        block_index = 1

        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False

            previous_block= block
            block_index += 1
        return True

    def add_transactions(self, sender, receiver, amount):
        self.transactions.append({"sender": sender ,
                                  "receiver": receiver,
                                  "amount or data": amount})
        previous_block = self.get_previous_block()
        return previous_block['index'] + 1

    def add_node(self, address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)


    def replace_chain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)
        for node in network:
            #find longest chain
            response= requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
```

Figure A.32: The code in blockchain.py file



```python
            response= requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']
                if length > max_length and self.is_chain_valid(chain):
                    max_length = length
                    longest_chain = chain

        if longest_chain:
            self.chain = longest_chain
            return True
        return False
```

Figure A.33: The code in blockchain.py file