

UNIVERSITY OF THE PHILIPPINES MANILA
COLLEGE OF ARTS AND SCIENCES
DEPARTMENT OF PHYSICAL SCIENCES AND MATHEMATICS

PHARMACOGENOMIC DATA STORAGE AND SHARING
USING PRIVATE BLOCKCHAIN

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Gabriel Austin Untalan

June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled “Pharmacogenomic Data Storage and Sharing Using Private Blockchain” prepared and submitted by Gabriel Austin Untalan in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Marbert John C. Marasigan, M.Sc. (*cand.*)
Adviser

EXAMINERS:

	Approved	Disapproved
1. Avegail D. Carpio, M.Sc.	_____	_____
2. Richard Bryann L. Chua, Ph.D (<i>cand.</i>)	_____	_____
3. Perlita E. Gasmien, M.Sc. (<i>cand.</i>)	_____	_____
4. Ma. Sheila A. Magboo, Ph.D. (<i>cand.</i>)	_____	_____
5. Vincent Peter C. Magboo, M.D., M.Sc.	_____	_____
6. Geoffrey A. Solano, Ph.D.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

_____	_____
Vio Jianu C. Mojica, M.Sc.	Marie Josephine M. De Luna, Ph.D.
Unit Head	Chair
Mathematical and Computing Sciences Unit	Department of Physical Sciences
Department of Physical Sciences	and Mathematics
and Mathematics	

Maria Constanca O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

Pharmacogenomics is an emerging field that combines pharmacology and genomics to study how an individual's genetic makeup influences their response to drugs. This personalized approach to medicine holds great potential for improving drug efficacy and reducing adverse effects. However, the effective implementation of pharmacogenomics in clinical practice is hindered by challenges in securely storing and sharing large volumes of sensitive genomic data. With the help of MultiChain, a private blockchain, a data storage and sharing system was developed to address these issues. MultiChain acts as the off-chain and on-chain storage for the system. It allows organizations to join the network and request data while preserving patient privacy.

Keywords: Private Blockchain, MultiChain, Pharmacogenomics, Data Sharing

Contents

Acceptance Sheet	i
Abstract	ii
List of Figures	v
List of Tables	vi
I. Introduction	1
A. Background of the Study	1
B. Statement of the Problem	2
C. Objectives of the Study	2
D. Significance of the Project	3
E. Scope and Limitations	4
F. Assumptions	4
II. Review of Related Literature	5
A. Pharmacogenomics	5
A..1 Challenges in Pharmacogenomics	5
B. Blockchain	6
B..1 Data Repository, Data Sharing, and Access Control Using Blockchain	6
B..2 Application of Blockchain on Pharmacogenomic Data	10
C. Synthesis	11
III. Theoretical Framework	13
A. Pharmacogenomics	13
A..1 Genetic Polymorphism	13
B. Blockchain	13
B..1 Workflow	14
B..2 Architecture	14

B..3	Blocks	15
B..4	Blockchain Categorization	16
C.	MultiChain	16
C..1	Smart Filters	17
C..2	Stream	17
IV.	Design and Implementation	18
A.	Use-case Diagram	18
B.	Data Storage	19
C.	Access Control	20
D.	System Architecture	21
E.	Technical Architecture	21
V.	Results	22
VI.	Discussions	31
VII.	Conclusions	34
VIII.	Recommendations	35
	References	36
IX.	Bibliography	36
X.	Appendix	40
A.	Source Code	40
A..1	Django Files	40
A..2	Templates	47
XI.	Acknowledgment	54

List of Figures

1	Use-Case Diagram	18
2	Login Page	22
3	Register Page	23
4	Organization not granted	23
5	Check Join Requests	24
6	Upload Data Page	24
7	Example CSV File	25
8	Request Page	25
9	View Data (Patient)	26
10	Check Request Page	26
11	Patient Consent	27
12	View Data (Data Requester)	27
13	Manage Access Page	28
14	Auditor's Transaction Page	28
15	Patient's Transaction Page	29
16	Organization and Data Requester's Transaction Page	29
17	Transactions in multichain-cli	30
18	Decoded Hex Data	30

List of Tables

1	Data structure of the access control stream	20
---	---	----

I. Introduction

A. Background of the Study

Pharmacogenomics (PGx) is the use of genetic information to improve the therapeutic outcomes of pharmacotherapy. Patients' responses to treatments are influenced by both genetic and non-genetic factors in many ways. Up to 95 percent of the variations in response to treatment can be attributed to genetic factors. The significance of these variations, however, may be greatly influenced by additional variables, including cultural, behavioral, and environmental ones. The purpose of PGx tests is to identify patients who respond or do not respond to treatment, interact with other drugs, experience side effects, and may need their dosage adjusted [1]. It has evolved into a significant aspect of drug development from the early stages of target discovery through the final stages of clinical development [2].

However, using pharmacogenomic data presents a number of challenges, particularly in terms of preserving patient privacy. One of the main challenges is the inclusion of genetic information in medical records, which raises concerns about the security and confidentiality of the data. Unauthorized access and misuse of this personal data pose risks of being shared to the public. Data-sharing regulations are also a barrier for pharmacogenomic studies [3]. There are policies such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA) that require access rights to data owners which complicates genetic data collection and sharing for study.

Blockchain is a decentralized and distributed digital ledger that records transactions on multiple computers, providing a secure and transparent way to record and verify transactions without the need for a central authority. Blockchain technology uses cryptography to ensure the authenticity and integrity of the transaction data [4]. One of the main uses of blockchain is in the financial industry, where it can be used to create and track digital currencies, such as Bitcoin and

Ethereum. It can also be used in supply chain management to track the movement of goods and ensure the authenticity of products. In healthcare, blockchain can be used to securely store and share patient data. The idea of blockchain in healthcare in particular has garnered several studies e.g. implementation of blockchain in the electronic health records [5].

B. Statement of the Problem

Maramba [6] proposed an access control system for pharmacogenomic data storage and sharing using InterPlanetary File Storage (IPFS), a decentralized and distributed system designed for storing and accessing files in a peer-to-peer network, as an off-chain storage and Solidity Smart Contracts from Ethereum as the blockchain. The issue with this proposed system is, since the doctor is both the data provider and requester, anyone with Ethereum address can register as a doctor. This means anyone can fake being a legitimate doctor conducting a research or clinical trial and request data from patients. Additionally, there is no purpose for every request sent by the doctors. This could pose as a problem for the patient to give consent for sharing his/her data since there is no way for them to verify the doctors requesting access or know the purpose for the request.

C. Objectives of the Study

The system aims to improve upon the work of [6] by having five users instead of a 2 users: doctor and patient, and by using private permissioned blockchain called MultiChain.

The system has the following functionalities:

1. Administrative entity
 - (a) Sets access permission to joining organizations (e.g. genomic laboratory hub)
 - (b) Adds Data Requester to the system

(c) Uploads the patient's pharmacogenomic data into the MultiChain stream.

2. Organizational entity

(a) Adds Data Requester to the system

(b) Uploads the patient's pharmacogenomic data into the MultiChain stream.

3. Patient

(a) Registers into the system

(b) Access and view their account

(c) Grant or Revoke Access

(d) Remove/hide their data

(e) View accounts who have access to their data

4. Data Requester (e.g. pharmacists, physicians)

(a) Access and view their account

(b) Request access to data while stating the purpose of the request.

(c) View data upon approval

(d) Download data into a CSV file

5. Auditor

(a) View the chain details

D. Significance of the Project

Addressing the concerns regarding consent and security is detrimental to the progress of pharmacogenomics. Using a private blockchain ensures that not anyone can get into the system and request data from the patient. Organizations must first request to join, or be invited into the system. Data requesters are then added by the organizational or administrative entity. This would give patients

the security that their data will not be easily requested or accessed by anyone. The system would implement the access control mechanism similar [6], but with added new features e.g. citing the research or clinical trial that intends to use the data, downloadable table of approved data, and access permission set by the administrator to the organization. Using MultiChain instead of Ethereum also means that there will be no fee using the blockchain.

E. Scope and Limitations

1. The project will only use the community version of MultiChain
2. Due to the limitations of using the community version, there will be functions missing unless the Enterprise version is used.
3. The drugs are limited to those with pharmacogenomic labeling.
4. The project conforms to the GDPR and HIPAA privacy rules.

F. Assumptions

1. The users of the system are knowledgeable in technology.
2. The patients have agreed for the organizations to upload their data into the system.
3. Other organization are invited or authorized by the administrator into the system
4. Gene and drug names, interaction scores, and annotation follow the Pharmacogenomics Knowledge Base (PharmGKB) format.
5. Organizations have a secure way of giving the data requesters their accounts
6. Data requesters given access do not share the data to others.

II. Review of Related Literature

A. Pharmacogenomics

Clinical DNA-based testing began with the diagnosis of sickle cell disease in 1978. Because of discoveries pertaining to genes linked to some common diseases, this opened up the possibility of predictive genetic testing to evaluate personalized disease risk. With the completion of the Human Genome Project and the availability of genome-wide sequence data, pharmacogenomics (PGx), the study and application of genetic factors relating to the body's response to drugs, was born[7, 8].

A.1 Challenges in Pharmacogenomics

However, there are a lot of ethical and regulatory issue, and potential risks associated with the use of pharmacogenomic data, including privacy concerns and the potential for malicious attacks.. Gershon et. al. [9] further analyzed these issues in their study. Pharmacogenomic data is highly sensitive and could be used to identify individuals or reveal personal information. The data is often collected and stored in a centralized database, which could be vulnerable to hacking or other malicious attacks. In 2021, over 45 Million protected health information (PHI) have been breached by attackers [10]. There are also questions regarding the identification of genetic variants leading up to individuals being subjected to discriminatory practices (e.g. being denied of insurance) [11].

Furthermore, testing personalized treatments on patients with rare diseases also opens up a lot of problems. The standard process of drug development is simply not yet catered for rare variant diseases. There is also the problem of consent; data mining of hospital records of anonymous individuals is common in research, however when consent is needed, broad-based consent is not possible under the current standards [9]. Gershon et. al. [9] states that the solution should lie in some combination of modifying the rules on consent to research to

allow open-ended consent and increasing the deterrent penalties for violations of privacy.

B. Blockchain

Blockchain was first introduced in 2008 by an entity under the pseudonym Satoshi Nakamoto by proposing Bitcoin, an electronic coin that doesn't rely on trust [12]. Blockchain is a distributed ledger where a chain of blocks contains the records of all committed transactions [13]. When more blocks are added, the chain continuously expands [13]. It provides, immutability, transparency, traceability, decentralization, and automation [14, 4]. One of the benefits of blockchain is its immutability. However, it also means storing large amounts of data in the blockchain would pose problems such as scalability and increased transaction fees, hence, blockchains alone would pose as a challenge in developing a data storage and sharing system. Which is why many researches have been experimenting on using an on-chain and off-chain system, wherein off-chain would store the data and the on-chain would handle all the transactions such as access control to view the data.

B.1 Data Repository, Data Sharing, and Access Control Using Blockchain

Since large files is not suitable to be stored in a blockchain, [15] used the Interplanetary File System (IPFS) as off-chain storage to compensate for blockchain inefficiency to store large files. The InterPlanetary File System (IPFS) is a file-sharing system designed to efficiently store and share large files. It can safeguard sensitive and personal data by not allowing other users to share data with unauthorized parties. To leverage access control in the blockchain, an Ethereum smart contract with 12 functions was used: (1) amIOwner and (2) amIOwnerMultiple, checks if the sender is the owner data, (3) checkAccess and (4) checkAccessMultiple, checks if access has been granted, (5) addBlock and (6) AddBlockMultiple, new data is added to storage, (7) grantAccess and (8) grantAccessMultiple, grants access to requester, (9)removeAccess and (10) removeAccessMultiple, removes ac-

cess to requester, (11) deleteBlock and (12) deleteBlockMultiple, all accesses and owner are removed.

Another approach is by Naz et. al. [16] where they implemented a system that uses Interplanetary File System (IPFS) for storing research files, and Ethereum Smart Contracts for the uploading or sharing of the file, review system, and the incentives of the data requester. Their system consists of 4 actors a Data owner, Data requester (Customer), Middleware (Workers), and an Arbitrator. They use the Shamir's Secret Sharing algorithm to encrypt the hashes generated by the IPFS. Middleware's private-public key pair are first initialized in the Smart Contract, only the middleware or workers can decrypt the data uploaded by the owner. Data owners uploads the meta data of the file into the IPFS. Once a file is uploaded, IPFS returns a hash which will then be encrypted using Shamir's Secret Sharing. Owners can select verified workers or middleware within the smart after receiving the encrypted hash. The encrypted hash is then uploaded into the system. Data requestors can then send request access to a file along with their RSA signature to all workers. Once workers verify the requestor, data requestors can then submit a deposit. Workers will then search and decrypt the requested file. Once the file is successfully downloaded by the requestors, they will submit a review which upon completion will receive 10 percent refund of their deposit. If the download fails, the arbitrator will then verify if it was a requestor mistake. If it is not a requestor mistake, requestor gets a full refund.

Shrestha et. al. [17] proposed a blockchain data-sharing framework using an on-chain and off-chain system in a travel industry. They used MultiChain both as the off-chain storage, by the uploading the user data as object in the MultiChain stream, and the blockchain for the user data. An enterprise is first initialized into the MultiChain. It serves as the administrator to grant other enterprises access permissions. These enterprises serves as the hosts or node. Depending on the administrator permissions, the enterprises can access data, or publish data within the stream or both. Only the participating nodes or enterprises can set

permissions provided they are given permission to do so by the administrator. All participants must have an Ethereum address to interact with the smart contract. Users registers into the enterprise by giving their information and choosing what data can be shared. User data is converted into a JSON format which is then published as items into the MultiChain stream. Users can decide which consumers can access their data. Only selected enterprises who agrees to the terms in the smart contracts can access or view the data published in the stream.

Similarly, Prata et. al [18] and Ismailisufi et. al. [19] also both implement MultiChain in their system albeit only using it merely as a data repository. [19] uses MultiChain as a storage for information on product items in food supply chain, [18] on the other hand created a framework based on it to store certificates issued by Federal University of Tocantins. Both studies used MultiChain in their studies because it is a private blockchain. The main advantage of a private blockchain is it is used for specific people/organization, which means only the necessary entities are allowed to access the data within the blockchain. So for a repository where a there is need for privacy within a certain network of enterprise or organization, private blockchains would prove to be beneficial. However, since the data can be accessed by anyone that is subscribed to the blockchain, there must be a need for access control. MultiChain in particular allows for access permission to be given by administrators to other nodes within the network, which gives a sense of access control on the part of the data provider. Private blockchains are also faster than public ones since there is no need for real decentralization, and participants do not need mine blocks, only validate transactions [18].

Chakraborty et. al. [20] implemented a similar approach with [18] wherein they also used MultiChain for certificates in the Education sector. The main difference is that [20] utilized more of MultiChain's networking capability by having the course providers as the hosts/nodes in the system. Their system is also more focused on the access control of the data owner. Employers can register in any of the course providers to request for student data. The student will then be able

to grant access to the employer by encrypting his/her certificate, student id, and employers public key into another stream within the MultiChain network.

MultiChain is also adopted in the medical field. [5] proposed a hybrid system approach of a secure Electronic Medical Record (EMR) storage using asymmetric keys supported by the Public Key Infrastructure (PKI), and secret session keys to store medical records in the MultiChain. The patient first creates a session key in the blockchain. Access to the patient's health records during a medical appointment requires patients to retrieve the session key addressing it with his/her public key then decrypt it with his/her private key. The patient will then publish the session key in the blockchain encrypted with the physician's public key, enabling the physician to access the health records using the patient's public key. After the medical appointment, the physician can then create the patient's medical record, encrypt it with the patient's session key, and sign it with his private key. The encrypted and signed record will then be published into the blockchain.

B..2 Application of Blockchain on Pharmacogenomic Data

Gürsoy et. al. [21] aims to solve the security and privacy challenges in pharmacogenomics by proposing to use blockchain technology in storing and querying pharmacogenomics data. They entered their proposed solution at the 2019 Integrating Data for Analysis, Anonymization, and Sharing (iDASH) competition as part of the Secure Genome Analysis Challenge. In their solution, the storing and querying of the data is done through Ethereum Smart-contracts created using Solidity programming language. They tested the speed of storing and querying of data using two methods: Challenge solution, and fastQuery solution. During their testing, they found that the fastQuery method is the fastest in terms of query and insertion. The system protects data from loss in a single point of failure scenario. However in their findings, their system is only scalable up to 10,000 entries due to limitations imposed by Solidity which poses problems as the number of pharmacogenomic data being used increases.

[22] identified 5 requirements in designing a data storage and sharing system for pharmacogenomic data: (1) Security and privacy, (2) Accessibility, (3) Interoperability, (4) Traceability, (5) Legal Compliance. They satisfied all these requirements by proposing an on-chain and off-chain hybrid system with a focus on access control. On-chain is implemented through Ethereum Smart Contracts while the off-chain is done through the Oracle server. There are three actors in the system: data creator, patient, and data requester. Access control is implemented in the smart contracts wherein the data creator can grant or revoke access to data requestors and patients can set access permission preferences.

A similar approach is proposed by Maramba [6]. In her proposed system, she used Interplanetary File System (IPFS) as the off-chain while also using Ethereum blockchain as Smart Contracts same as [21, 22]. There are only 2 actors in the system: a patient, and a doctor. The doctor however acts as the data creator and data requestor. The access control mechanism is much more data owner friendly as it allows the patient to grant and revoke access directly to the data requester.

C. Synthesis

Pharmacogenomics is an important part of drug development because it aims to personalize drug treatment in order to avoid adverse reactions that could harm the patient. Regrettably, concerns about consent, unauthorized access of data, data security, and regulatory requirements have slowed its progress. [21] used blockchain in order to securely store PGx data. Despite its security it still does not address the other concerns. [22] solved this by implementing an access control mechanism by allowing patients to set permission preferences, and allowing data creators to grant and revoke data access while using an off-chain storage to compensate for the blockchain's scalability issues. Maramba [6] also used this approach albeit implementing a similar architecture proposed by [15]. However, [6]'s system does not employ user authentication or verification since anyone with an ethereum address can register and pose as a legitimate doctor and request data from the patient, and does not provide information on the doctor's purpose of requesting. Chakraborty et. al. [20] used MultiChain as the blockchains in their system. Being a private blockchain means that it is faster and more cost effective since they have fewer nodes participating in the network and have a more centralized structure. This allows for faster consensus and fewer network bottlenecks. Entities will also not be easily able to access the blockchain unless they are verified by an administrator. [20] was able to implement an access control mechanism for the data owner but in a much more complicated way that only allows for granting access. [17] managed to implement an access control mechanism using Ethereum smart contracts. The smart contracts allow the data owners to set what data they want to share, and to set which data requestor can access their data. This approach can be used to solve the problems in [6, 20]. MultiChain is already both an off-chain and on-chain, and external encryption is not needed as it already provides the encryption for the data. MultiChain could be used to as a data storage and blockchain for pharmacogenomic data, and set access permissions to other organizations while the smart contracts called "smart filters" from the

MultiChain itself will be used to give patients access control, and to set the terms and conditions for the organizations to join into the network.

III. Theoretical Framework

A. Pharmacogenomics

Pharmacogenomics is defined as the study of drug interaction in relation to the genetic properties of an individual [23, 8]. It examines the theory that genomic variability underpins drug response variability [11].

A.1 Genetic Polymorphism

The term "polymorphism" refers to a mutation in the genome that varies among people in a significant portion of the population. Researchers who study pharmacogenomics focus on two determinants regarding polymorphism: pharmacokinetics and pharmacodynamics [24, 8].

1. Pharmacokinetics relates to the absorption, distribution, metabolism, and excretion (ADME) processes of the drug. It examines what the body does to a drug, and how much of a drug is required to reach its target in the body.
2. Pharmacodynamics is the opposite of pharmacokinetics as it relates to how well a target (e.g. receptors, ion channels, enzymes, or immune system) reacts to the drug. Or in short, what the drug does to the body.

The parameters of these determinants can be altered by a genetic alteration known as single nucleotide polymorphism (SNP) [1]. SNP is the most common type of polymorphism in pharmacogenomics [7]

B. Blockchain

A blockchain is a distributed ledger of transactions or digital events that have been shared among involved parties [25]. The transaction is verified through the consensus of the system's users. As the data or information enters the block, it will become immutable, meaning it cannot be changed or deleted within the blockchain

system. Every transaction ever made is contained in a certain, verifiable record on the blockchain.

B..1 Workflow

The transaction within the blockchain works by [25]:

1. The transaction is first stored in a block.
2. The transaction is then broadcast to every party in a peer-to-peer (P2P) network
3. Every party in the network with then verify the validity of the transaction
4. Once transaction is approved, it can now be added to the chain.

B..2 Architecture

The blockchain architecture consists of six layers as defined by [26]:

1. Data Layer: The data layer is the lowest layer of the blockchain protocol stack. It creates the data structure responsible for the storage and organization of data. The header is used to store the meta-information of the block, while the body uses the integrity of the data stored in the block.
2. Network Layer: The network layer's primary responsibility is to support information exchange between peer nodes while protecting the data's security and privacy.
3. Consensus Layer: The blockchain's consensus algorithm is contained in the consensus layer. Consensus algorithms such as:
 - (a) Proof-of-Work
 - (b) Proof-of-Stake
 - (c) Proof-of-Authority

4. Incentive Layer: The mechanism for issuing and distributing tokens is created by the incentive layer, and this mechanism establishes the overall number and distribution of tokens.
5. Contracts Layer: Smart contracts are commitments and rules that are predefined in blockchain systems and are automatically carried out when certain conditions are met. Without the involvement of a third party, smart contracts allow for traceable and irreversible trusted transactions. Different Opcodes, Chaincodes, and smart contracts are all contained in the contract layer. Opcodes and Chaincodes specify the specifics of trading and processes, enhancing the network's autonomy and programmability.
6. Application Layer: The application layer acts as the frontend of the blockchain. It facilitates the use of blockchain in different kinds of application.

B..3 Blocks

According to [27] the block has two parts: the Block header and the Block data.

1. Block Header consists of:
 - (a) The block number
 - (b) Previous hash value
 - (c) Hash representation of the block data
 - (d) Timestamp
 - (e) Size of Block
 - (f) Nonce Value
2. Block Data consists of:
 - (a) A list of transactions.
 - (b) Ledger of events.

B..4 Blockchain Categorization

There are three categories of blockchain [27, 28]:

1. **Permissioned:** Also known as private blockchains. In a permissioned blockchain network, only a specific group of people or organizations can publish within the blockchain system as long as a central figure gives them permission.
2. **Permissionless:** Also known as public blockchains. In a permissionless blockchain network, anyone can publish within the blockchain system.
3. **Consortium:** Consortium blockchain is a combination of both permissioned and permissionless blockchain. Only a specific group of people or organizations can publish within the blockchain system without a central figure. This means that the organizations can give permissions to each other.

C. MultiChain

MultiChain is private blockchain developed by Dr. Gideon Greenspan [29]. It allows users to control the maximum block size and configure every aspect of the blockchain in a configuration file. Administrative permissions are automatically given to the miner of the first block.

MultiChain has three main objectives [17]:

1. Blockchain's activity should be shown only to chosen participants.
2. Only selected transactions should be permitted.
3. Securely conduct mining without proof of work and its associated costs.

C..1 Smart Filters

The term smart contracts was coined by Nick Szabo in 1994. He defines it as "a computerized transaction protocol that executes the terms of a contract..." [27]. In Ethereum, smart contract is a collection of code and data deployed in a blockchain and is written using the Solidity programming language. It stores the rules for contract negotiations, automatic contract verification, and contract execution. [17] It can only be executed once and all nodes that execute it must derive the same results from the execution. The results of each execution are recorded on the blockchain [27]. In MultiChain 2.0, it allows for the creation of smart contracts as well in the form of "Smart Filters". It allows for custom rules to be defined regarding the validity of transactions or stream items [30].

C..2 Stream

The MultiChain data stream is an append only storage or database. A MultiChain blockchain can contain any number of streams. Each stream consists of an ordered list of items that has the following features:

1. one or more publishers who have digitally signed the item,
2. one or more keys with a length of 0 to 256 bytes to enable effective retrieval,
3. some data, either on-chain or off-chain, in JSON, text, or binary format,
4. stream filters that are used to define custom validation rules for the data,
5. and details about the transaction and block for the item, including its txid, blockhash, blocktime, etc.

Data can be published in stream either on-chain or off-chain. On-chain data is directly embedded into the blockchain, while off-chain data can embed a hash up to 1GB per item.

IV. Design and Implementation

A. Use-case Diagram

The system has four users: Administrative Entity, Organizational Entity, Patient, and Data Requester.

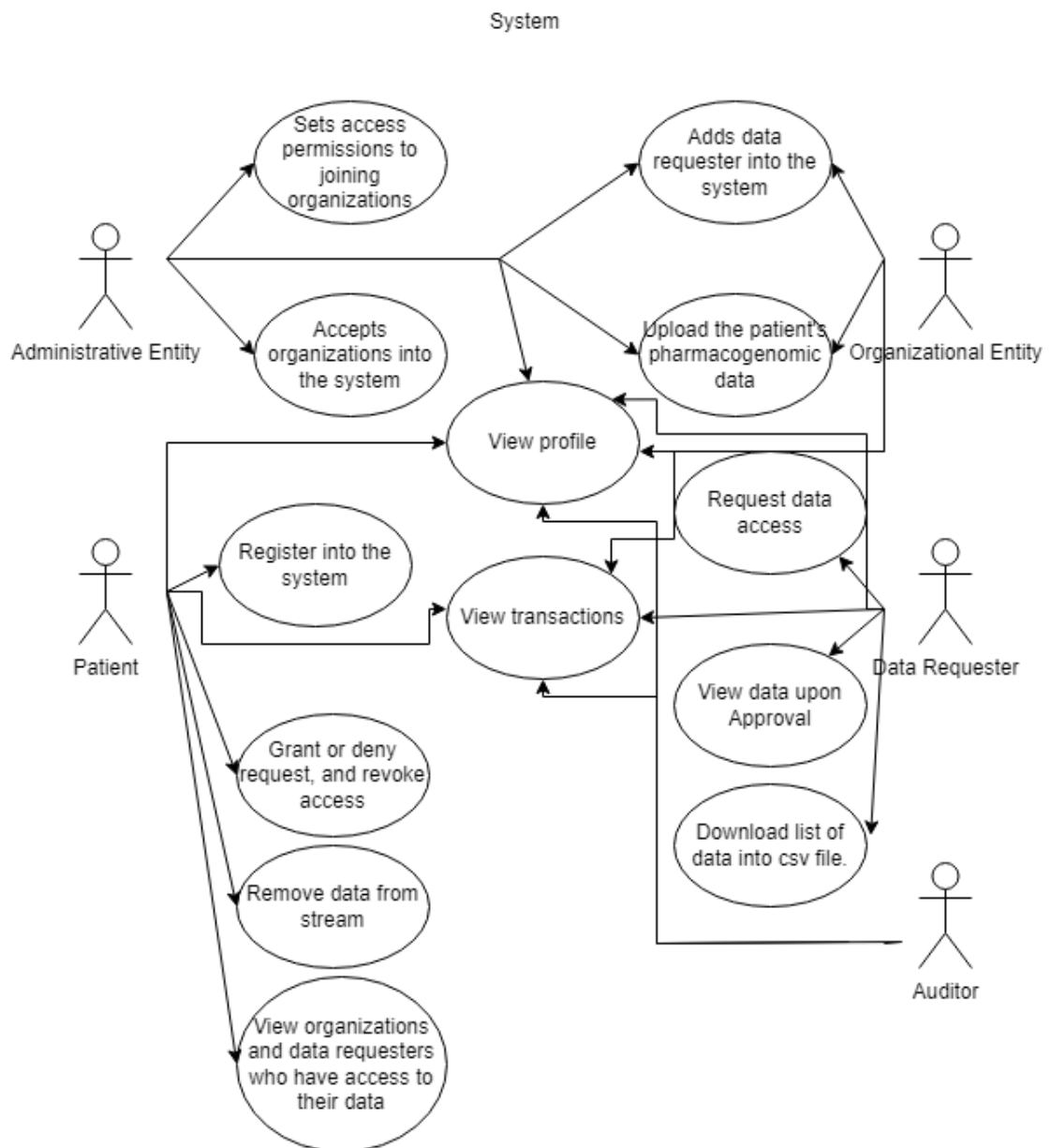


Figure 1: Use-Case Diagram

Patients can create their account in the system. They have control on granting and revoking access on their data. They would also be able to see who has access on their data. Patient data can only be uploaded into the system by either the administrative entity or an organizational entity.

For data requesters to get into the system, they must first be added by an administrative or an organizational entity. Only then can data requesters request access to the data.

B. Data Storage

Using the MultiChain blockchain platform, the developed system incorporates off-chain storage functionality for pharmacogenomic data. When patient data is uploaded, it is published as off-chain by including the keyword "offchain" in the publish function. This designates the data as suitable for off-chain storage. Before the data is fully published, it must be first be encrypted using the "binascii" library of Python. The encrypted data is then fragmented into smaller chunks, and each chunk is stored as an individual transaction in the MultiChain stream.

A mapping is established to sustain the connection between the patient's data and the corresponding MultiChain transactions. This mapping associates the identifier or key of the patient with the specific transaction IDs or references in MultiChain. For most stream items, the patient's address acts as the identifier or the key e.g. when the patient's pharmacogenomic data is uploaded, the patient's address acts as the key, or when publishing for the access control stream, the patient's address acts as the publisher of that stream item while the key for that item is the organization's address and this will be shown in the transaction when the list of stream items is queried.

By employing MultiChain as the off-chain storage solution, the system prevents the blockchain from becoming overburdened with massive data files. This method optimizes the system's storage capacity and scalability, ensuring that the blockchain remains lightweight and effective.

C. Access Control

At the administrative level, the system allows administrative entities to set access permissions for joining organizations. This step ensures that only trusted organizations are granted access to the system and that proper vetting processes are in place. As the organization creates an account, a request, which contains the organization’s address and the status ‘No grants yet’ will be passed on to the ‘*org_request*’ stream, using the organization’s address as the key, which will then be viewed by the admin. After granting, a new item will be published in the stream with the status ‘permissions granted’, indicating that the organization can now add data requesters and upload patient data. For patients, the access con-

Name	Data Type	Description
patient_address	address	Patient’s address
org	address	Organization’s address
data_id	string	The data’s identifier.
purpose	string	Research/Clinical Study.
timestamp	date	Time/Date published
access_level	string	‘grant’, ‘deny’, or ‘revoke’.

Table 1: Data structure of the access control stream

control function provides them with the ability to manage and control access to their own pharmacogenomic data. Patients can selectively grant or deny the request of the data requesters, and revoke access to their data to granted requesters. When granting, denying, or revoking access to a data requester, the system updates the access control list for the relevant data stream. This is achieved by adding the requester’s and patient’s addresses to the ‘*access_control*’ stream, along with the status ‘grant’, ‘deny’, or ‘revoke’, indicating that they do or don’t have permission to access the data.

Furthermore, data requesters are also subjected to the access control mechanisms implemented in the system. To request access to patient data, data requesters must explicitly state the purpose of their access. This request is passed on to the ‘*request-data*’ stream with the patient’s address as it’s key.

D. System Architecture

The system is a semi-decentralized application developed on the MultiChain blockchain platform. MultiChain provides a comprehensive set of built-in functions, including access permission management and off-storage capabilities, which were utilized in the development process. Python programming language was utilized to handle the interactions with MultiChain, utilizing the "savour" package to call JSON-RPC API commands. The application architecture incorporates the Django framework as the backend, enabling seamless integration with the MultiChain network. Additionally, Bootstrap was employed to design and create the frontend of the application, ensuring a user-friendly and visually appealing interface.

E. Technical Architecture

Below are the system requirements to run MultiChain [31]:

1. Linux: 64-bit, supports Ubuntu 12.04+, CentOS 6.2+, Debian 7+, Fedora 15+, RHEL 6.2+.
2. Windows: 64-bit, supports Windows 7, 8, 10, 11, Server 2008 or later.
3. Mac: 64-bit, supports OS X 10.11 or later.
4. 512 MB of RAM
5. 1 GB of disk space

V. Results

All users' initial landing pages are the login pages. Users must register and specify their roles if they do not already have an account in order to access the system; otherwise, they must enter their username and password.

Log in

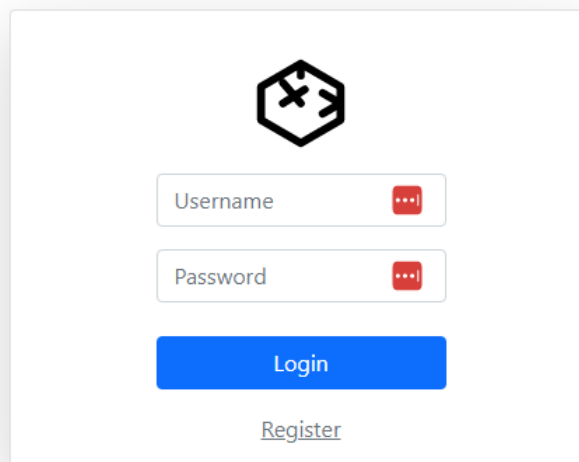
A screenshot of a login page. At the top center is a hexagonal logo containing a stylized 'X' and a right-pointing arrow. Below the logo are two input fields: 'Username' and 'Password'. Each field has a red eye icon on the right side, indicating a toggle for password visibility. Below the password field is a blue 'Login' button. At the bottom center is a blue link labeled 'Register'.

Figure 2: Login Page

Users can access the registration page by clicking the link beneath the login button on the login page in order to create a new account. The names, roles, usernames, and passwords of all users must be provided.

Create an Account!

Select Designation ▼

Username [Red eye icon]

Name

Password [Red eye icon] Repeat Password [Red eye icon]

Register Account

[Already have an account? Login!](#)

Figure 3: Register Page

Once registered, patients and auditors are redirected to their respective home-pages, the check request page for patients and the transaction view page for auditors. However, although organizations can access the system immediately, their functionality is limited because the administrative entity has not yet granted them permissions, so their default home page is their account page. This limitation is in place to ensure the security of the system and prevent unauthorized access.

PGChain Account Logout

X You are now logged in as 1EBmGqURqTnZKYs1oqK66R5WE/fqUysvmE3yv9

Profile

Name: Mercury Drug Corporation **Status:** No grants yet

Address: 1EBmGqURqTnZKYs1oqK66R5WE/fqUysvmE3yv9

Role: organization

Figure 4: Organization not granted

To grant permissions to the organizations, the administrative entity must check the join requests.

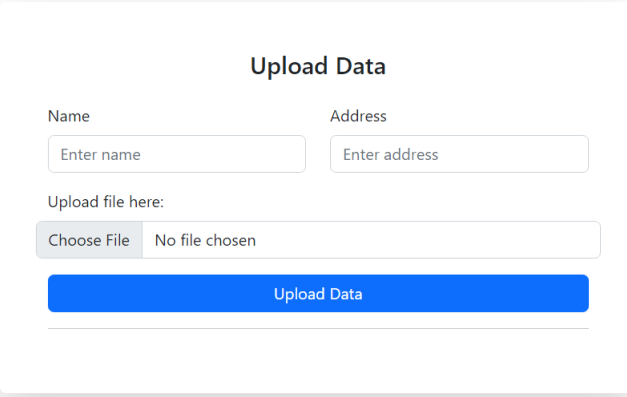
Check Requests

Organization

Mercury Drug Corporation Grant Permissions

Figure 5: Check Join Requests

Once permissions are granted, organizations can fully utilize the system's features, such as being able to create data requesters, and add patient pharmacogenomic data. The organizations must ensure that the name and address of the patient are registered in the system when adding the data, otherwise, they won't be able to add the data. Organizations must upload a csv file with



Upload Data

Name Address

Upload file here:

Figure 6: Upload Data Page

Organizations must upload a csv file with the pharmacogenomic data of the patient.

	A	B	C	D	E	F	G	H	I	J
1	Gene	Drugs	Interactioi Annotation							
2	CYP2D6	Codeine	100	Reduced efficacy of codeine due to impaired conversion to its active form,						
3	VKORC1	Warfarin	201	Increased sensitivity to warfarin, lower dose requirement.						
4	SLCO1B1	Simvastatin,Amiodarone	250	Increased risk of myopathy and rhabdomyolysis when simvastatin is co-ad						
5										
6										
7										
8										
9										

Figure 7: Example CSV File

Once data has been added, data requesters, whether from the same or a different organization, can request access to the data owner their respective researches or clinical trials. Data requesters can also request multiple data. The data to be requested can be filtered by gene or drug ID.

Request Data

Gene:

Drug:

Filter

- 1MruSBu5RxbwJqpHMLfe8XqQsLiLdtNFV7T8LR_atomoxetine
Gene: BDNF-CC

- 1CtrqffQ8uco6VnuYng3rc43PF1mMCHcbibXv_warfarin
Gene: CYP2C9

Request Access

Figure 8: Request Page

Patients would also be able to view their data in the View Data page upon data being added.

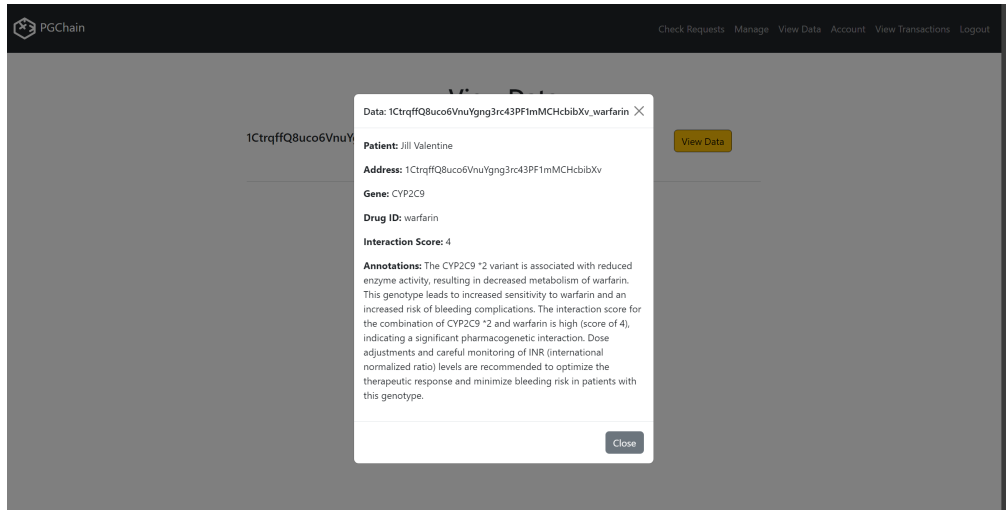


Figure 9: View Data (Patient)

After the request is sent, the patient owning the data should be able to see the requests on the check requests page. Before granting the request, the patient will be shown a consent form. Patients can also deny the request if they choose to do so.

Check Requests

Name	Address	Organization	Data	Purpose	
Juan Miguel	1SDBguiidU8tc2im54bA5EAdXijjDBC6d1xKkz	Mercury Drug Corporation	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MfFLK5Y_Warfarin	MD Clinical Trial #2	<input type="button" value="Grant"/> <input type="button" value="Deny"/>

Figure 10: Check Request Page

Consent ×

I hereby grant access to my pharmacogenomic data to **Mercury Drug Corporation** for the research/clinical trial "**MD Clinical Trial #2**". I understand that my data will be used in accordance with all applicable privacy laws and regulations, and will only be accessed by authorized individuals involved in my healthcare.

I acknowledge that granting this access is voluntary, and I have been provided with sufficient information about the purpose and potential risks of sharing my data. I understand that I have the right to revoke this access at any time.

By clicking the "Grant Access" button below, I consent to the above terms and conditions.

Cancel Grant Access

Figure 11: Patient Consent

Upon approval, the data should be accessed by all the data requesters, from the same organization, on the view data page similar to Figure 9. However, the data requester has the option to download all their accessed data into a csv file.

View Data

1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MFfLK5Y_mercaptopurine

Gene: TPMT

View Data

Click the button below to download the accessed data as a PDF:

Download Data

Figure 12: View Data (Data Requester)

Patients should now be able to see which organization has access to their data and the list of users under that organization on the Manage Access page. They should also be able to revoke access to each research/clinical study of the

organization thereby revoking all user's access within that organization.

Manage Access

Organization	Data	Purpose		
Umbrella Corporation	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MFfLK5Y_mercaptapurine	Research	View Users	Revoke

Figure 13: Manage Access Page

The transactions of the access control should be visible to the patient, organization, data requester, and auditor. They would be able to view them on the View Transactions page.

Transaction View

Patient Address: Access Level:

[Filter](#)

Txid	Patient Address	Requester Address	Data
770e5be6176e99cf2146c6423a623eb67b640d4107cf797805b326f41240d23e	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MFfLK5Y	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc	
78a9cbece2767f7fe492be04e58fa9622740a69083442efddc8c08cf2b90401	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MFfLK5Y	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc	
34b05aa1fe27e48562c657253efbe43a8fb169c37a5deb2b5baea699694325e3	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MFfLK5Y	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc	
05c4a88f5e1130d8edc76d6352223b4788a235c01ab931179e30f0539c00b05	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MFfLK5Y	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc	
da8974bbca508ea6ddafe6480079795644dc5706e7db8d4f94af853b024a1267	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc4MFfLK5Y	1RkoFVmZNgPkDcR5si3qhr4jThe1Lc	

Figure 14: Auditor's Transaction Page

Transaction View

Requester Address: Access Level:

Txid	Requester	Org	Data	Purpc
d3c63bb050cef0f9a50722c52b6615e060629675a4d7d98edc37f2270cae7d352	Albert Wesker	Umbrella Corporation	1MruSBu5RxbuJqpHMLfe8XqQsLiLdtNFV7T8LR_atomoxetin	Resea
3c48aa25f810c814fbd1633b2ea3e4375de534ac6a8b8c44a6e3684569bc4ec	John Doe	AdminOrg	1MruSBu5RxbuJqpHMLfe8XqQsLiLdtNFV7T8LR_atomoxetin	Resea
7b7a3ed3c933254463da32c0aed52eed5cac50fb2dcd36219ed0d915c06b4973	John Doe	AdminOrg	1MruSBu5RxbuJqpHMLfe8XqQsLiLdtNFV7T8LR_atomoxetin	Resea
0a25f701cd988c50898b1f8509d6007fa654585831c894ebbb4bad005aadaa05	John Doe	AdminOrg	1MruSBu5RxbuJqpHMLfe8XqQsLiLdtNFV7T8LR_atomoxetin	Resea
c496fc2c8baa1bfcee387ac7f75f84e10e92370bd8b443d0a25f435a97c52cdd	John Doe	AdminOrg	1MruSBu5RxbuJqpHMLfe8XqQsLiLdtNFV7T8LR_atomoxetin	Resea
22r61bh77783605fe708eac806092e59284f63e54hr033a48608a3971b415e6f	John	AdminOrg	1MruSBu5RxbuJqpHMLfe8XqQsLiLdtNFV7T8LR_atomoxetin	Resea

Figure 15: Patient's Transaction Page

Transaction View

Patient: Access Level:

Txid	Patient	Data	Purpose	Acce
05c4a88f55e1130d8edc76d6352223b4788a235c01ab931179e30f0539c00b05	Ronald McDonald	1RKoFVmZNgPkDcR5si3qhr4jThe1Lc4MFILK5Y_mercaptopurine	Research	grar

Figure 16: Organization and Data Requester's Transaction Page

These access control transactions can also be viewed in the multichain-cli. Here transaction shows the patient's address, which is the 'publisher' in this case, the name of the organization, which is the key for the stream item, and the data, which is encoded in hexadecimal.

```

{
  "publishers": [
    "1RkoFVmZNgPkDcR5si3qhr4jThE1Lc4MFfLK5Y"
  ],
  "keys": [
    "Umbrella Corporation"
  ],
  "offchain": false,
  "available": true,
  "data": "7b2270617469656e745f61646472657373223a202231524b6f46566d5a4e67506b44635235736933716872346a546845314c63344d46664c4b3559222c20226f7267223a2022556d6272656c6c6120436f72706f726174696f6e222c2022646174615f6964223a202231524b6f46566d5a4e67506b44635235736933716872346a546845314c63344d46664c4b35595f6d6572636170746f707572696e65222c2022707572706f7365223a20225265736561726368222c202274696d657374616d70223a2022323032332d30362d31365432313a33313a31382e3430343335322c20226163636573735f6c6576656c223a20226772616e74227d",
  "confirmations": 43,
  "blocktime": 1686922291,
  "txid": "05c4a88f55e1130d8edc76d6352223b4788a235c01ab931179e30f0539c00b05"
},

```

Figure 17: Transactions in multichain-cli

We can decode the hex data in python to see if transactions match the transactions in the Transactions page

```

>>> data = "7b2270617469656e745f61646472657373223a202231524b6f46566d5a4e67506b44635235736933716872346a546845314c63344d46664c4b3559222c20226f7267223a2022556d6272656c6c6120436f72706f726174696f6e222c2022646174615f6964223a202231524b6f46566d5a4e67506b44635235736933716872346a546845314c63344d46664c4b35595f6d6572636170746f707572696e65222c2022707572706f7365223a20225265736561726368222c202274696d657374616d70223a2022323032332d30362d31365432313a33313a31382e3430343335322c20226163636573735f6c6576656c223a20226772616e74227d"
>>> data = bytes.fromhex(data).decode('utf-8')
>>> print(data)
{"patient_address": "1RkoFVmZNgPkDcR5si3qhr4jThE1Lc4MFfLK5Y", "org": "Umbrella Corporation", "data_id": "1RkoFVmZNgPkDcR5si3qhr4jThE1Lc4MFfLK5Y_mercaptopurine", "purpose": "Research", "timestamp": "2023-06-16T21:31:18.404353", "access_level": "grant"}
>>>

```

Figure 18: Decoded Hex Data

VI. Discussions

The developed application system aimed to fulfill several objectives related to administrative, organizational, patient, data requester, and auditor entities. Upon evaluation, it can be concluded that the system has successfully achieved these objectives. The administrative entity is able to set access permissions for joining organizations and accept nodes into the system. Additionally, the entity can add data requesters to the system and upload patients' pharmacogenomic data into the MultiChain stream. Similarly, the organizational entity can add data requesters and upload patient data into the stream. Patients can register into the system, access and view their accounts, grant or deny requests, revoke access to their data, and view the accounts that have access to their data. Data requesters, such as pharmacists and physicians, can access their accounts, request data access, view data upon approval, and download the data as csv file. Auditors have the ability to view chain details, ensuring transparency and accountability.

Throughout the development process, various challenges were encountered such as User-related issues, framework limitations, programming language challenges, and technology constraints. Despite the successful achievement of the outlined objectives, one limitation of the current system is the lack of the "remove access" feature. Due to the utilization of the open-source version of MultiChain, this functionality is not available. However, future iterations or the adoption of the MultiChain Enterprise version could enable the implementation of this feature. Incorporating the "remove access" capability would grant patients greater control over their data privacy and security.

To optimize query performance, MultiChain initially limited data fetching to 10 items per stream. However, this imposed limitations when presenting information such as transaction lists, especially for auditors who require a comprehensive view. In order to address this constraint and enhance the user experience, solution was implemented that allows the maximum number of items to be returned per stream, providing a more comprehensive and detailed overview. By setting the limit to 256

items, we ensure that auditors and other viewers have access to a broader range of data, enabling them to perform thorough analysis and gain valuable insights. This enhancement not only improves the usability of the system but also enables auditors to effectively evaluate the chain details.

Comparing this work with the main references, notable differences emerge. The referenced works utilize Smart Contracts for on-chain data management for access control, and IPFS for off-chain data storage. In contrast, this system leverages MultiChain for stream-based data management. Employing MultiChain in the system brings significant benefits to data privacy, scalability, and interoperability in the healthcare ecosystem. The use of blockchain technology ensures secure and tamper-evident storage, making it nearly impossible to modify or tamper with stored pharmacogenomic data without detection. MultiChain's access control mechanisms provide granular control over data access, allowing only authorized entities to interact with the data. This enhances privacy and compliance with data protection regulations. Furthermore, MultiChain's decentralized nature improves data privacy by distributing data across multiple nodes and granting certain entities control over their own data. The system can scale to handle large volumes of data and promotes interoperability with other healthcare systems, facilitating seamless data sharing and collaboration. By leveraging MultiChain, the system establishes a strong foundation for secure and efficient pharmacogenomic data management.

The developed system also goes beyond the previous references by introducing the purpose of data requests. This addition promotes transparency and empowers patients to make informed choices regarding their pharmacogenomic data sharing. By requiring requesters to explicitly state their access purpose, patients gain a clear understanding of how their data will be utilized. Moreover, implying the purpose ensures compliance with data protection laws and regulations, reinforcing the system's commitment to privacy and ethical data practices.

A dedicated page is provided for concerned users to view the access control

transactions associated with their data. This page is where users can access and review the detailed information about the access control transactions in a user-friendly format. By having a dedicated page for transactions, users can easily and track the history of the access control stream.

To ensure the legitimacy and integrity of these transactions, users are provided with two options for verification. The first option is to use the 'multichain-cli' command-line interface, which allows users to interact directly with the MultiChain blockchain network. By utilizing the 'multichain-cli' tool, users can query and verify the access control transactions by retrieving the transaction details and confirming their validity against the blockchain records using *liststreamitems *stream_name**. Alternatively, users can leverage the MultiChain Explorer, a web-based interface that provides a visual representation of the blockchain network. The MultiChain Explorer allows users to explore the blockchain, view transaction details, and verify the access control transactions associated with their data.

The main contributions of this work are twofold. Firstly, the developed application system addresses the need for effective pharmacogenomic data management and access control. By providing a user-friendly interface and comprehensive functionalities, it empowers administrative entities, organizations, patients, data requesters, and auditors to efficiently manage and access relevant data. Secondly, the system introduces an innovative approach to data management by utilizing MultiChain. This novel implementation provides a robust and secure platform for storing and sharing pharmacogenomic data, addressing privacy concerns and enhancing data integrity.

VII. Conclusions

The developed application system effectively fulfills its objectives, catering to the needs of administrative entities, organizations, patients, data requesters, and auditors. It enables administrators to set access permissions, organizations to submit patient data, and patients to control access to their own data.

The inclusion of a data request's intent within the system enhances transparency and enables patients to make informed decisions regarding the sharing of their pharmacogenomic data. By explicitly stating the purpose of their access request, data requesters demonstrate transparency and accountability, aligning with data protection regulations. This feature ensures that patients have a clear understanding of how their data will be utilized and allows them to assess the relevance and potential benefits of sharing their data for a specific purpose. It promotes a culture of trust, privacy, and compliance, strengthening the overall ethical framework of the system.

By incorporating the concept of organizations joining the system and creating data requesters, the developed application ensures that requesters are legitimate entities, subject to background checks and approval by the administrative entity. This approach transfers the liability for data privacy breaches from the system owner to the organization, enhancing accountability and reinforcing data protection measures. It establishes a robust framework where organizations play a crucial role in safeguarding patient data, fostering trust, and ensuring compliance with privacy regulations. This organizational structure strengthens the overall security and privacy framework of the system, mitigating potential risks and ensuring the responsible handling of sensitive pharmacogenomic data.

Utilizing MultiChain technology, the system guarantees secure storage and enables seamless integration with other healthcare systems, thereby fostering collaboration and coordinated patient care. Overall, the system represents a significant improvement in pharmacogenomic data administration, transforms data sharing practices, and advances precision medicine.

VIII. Recommendations

In order to enhance the overall system's implementation, there are several key recommendations to consider. Firstly, a better implementation of MultiChain streams could be explored to optimize data management and retrieval. This could involve adding stream per patient, or requester, in order to not overload the system when for example fetching all the access control transaction.

Secondly, integrating Metamask as a user authentication solution could greatly enhance security and user convenience. Metamask is a widely used Ethereum wallet and browser extension that provides secure management of private keys. By incorporating Metamask into the system, users can securely authenticate themselves without relying on traditional username-password combinations. This not only improves security by eliminating the need to store sensitive login credentials but also enhances the user experience by offering a seamless and intuitive authentication process.

Lastly, considering the use of the enterprise version of MultiChain could provide additional benefits, particularly in terms of managing and removing stream items. The enterprise version offers advanced functionalities and capabilities specifically designed for enterprise-grade applications. Leveraging the enterprise version could enable smoother removal of stream items, ensuring efficient data management and maintenance within the system.

IX. Bibliography

- [1] J. Oates and D. Lopez, “Pharmacogenetics: An important part of drug development with a focus on its application,” *International Journal of Biomedical Investigation*, vol. 1, pp. 1–16, May 2018.
- [2] K. Bienfait, A. Chhibbe, J. Marshall, M. Armstrong, C. Cox, P. M. Shaw, and C. Paulding, “Current challenges and opportunities for pharmacogenomics: perspective of the industry pharmacogenomics working group (i-pwg),” *Human Genetics*, vol. 141, pp. 1165–1173, 2022.
- [3] M. E. Klein, M. M. Parvez, and J.-G. Shin, “Clinical implementation of pharmacogenomics for personalized precision medicine: Barriers and solutions,” *Journal of Pharmaceutical Sciences*, vol. 106, pp. 2368–2379, 2017.
- [4] N. Darlington, “What is blockchain technology? a step-by-step guide for beginners.” <https://blockgeeks.com/guides/what-is-blockchain-technology/>, 2022.
- [5] M. T. de Oliveira, L. H. A. Reis, R. C. Carrano, F. L. Seixas, D. C. M. Saade, C. V. Albuquerque, N. C. Fernandes, S. D. Olabbarriaga, D. S. V. Medeiros, and D. M. F. Mattos, “Towards a blockchain-based secure electronic medical record for healthcare applications,” *IEEE International Conference on Communications (ICC)*, May 2019.
- [6] M. A. F. M. Maramba, “A blockchain-based access control system for storing and sharing pharmacogenomic data,” June 2022.
- [7] S. A. Scott, “Personalizing medicine with clinical pharmacogenetics,” *Genet Med*, vol. 13, pp. 978–995, December 2011.
- [8] K. J. Karczewski, R. Daneshjou, and R. B. Altman, “Chapter 7: Pharmacogenomics,” *PLOS Computational Biology*, vol. 8, p. e1002817, December 2012.

- [9] E. S. Gershon, N. Alliey-Rodriguez, and K. Grennan, “Ethical and public policy challenges for pharmacogenomics,” *Dialogues Clin Neurosci*, vol. 16, pp. 567–574, 2014.
- [10] H. Landi, “Healthcare data breaches hit all-time high in 2021, impacting 45m people,” *Health Tech*, February 2022.
- [11] D. M. Roden, R. B. Altman, N. L. Benowitz, D. A. Flockhart, K. M. Giacomini, J. A. Johnson, R. M. Krauss, H. L. McLeod, M. J. Ratain, M. V. Relling, H. Z. Ring, A. R. Shuldiner, R. M. Weinshilboum, and S. T. Weiss, “Pharmacogenomics: Challenges and opportunities,” *Ann Intern Med*, vol. 145, pp. 749–757, 2006.
- [12] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.” <https://bitcoin.org/bitcoin.pdf/>, 2008.
- [13] Z. Zheng, H. Wang, S. Xie, H.-N. Dai, and X. Chen, “Blockchain challenges and opportunities: A survey,” *International Journal of Web and Grid Services*, vol. 14, 2018.
- [14] IBM, “Benefits of blockchain.” <https://www.ibm.com/topics/benefits-of-blockchain>.
- [15] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State, “Blockchain-based, decentralized access control for ipfs,” *IEEE Xplore*, 2018.
- [16] M. Naz, F. A. Al-zahrani, R. Khalid, N. Javaid, A. M. Qamar, M. K. Afzal, and M. Shafiq, “A secure data sharing platform using blockchain and interplanetary file system,” *Sustainability*, vol. 11, p. 7054, December 2019.
- [17] A. K. Shrestha, J. Vassileva, and R. Deters, “A blockchain platform for user data sharing ensuring user control and incentives,” *Frontiers in Blockchain*, vol. 3, October 2020.

- [18] D. N. Prata, H. X. de Araujo, and C. Santos, “Blockchain technology applied to education,” *International Journal of Advanced Engineering Research and Science*, vol. 6, p. 295–298, 2019.
- [19] A. Ismailisufi, T. Popovic, N. Gligoric, S. Radonjic, and S. Sandi, “A private blockchain implementation using multichain open source platform,” *2020 24th International Conference on Information Technology (IT)*, February 2020.
- [20] S. Chakraborty, K. Dutta, and D. Berndt, “Blockchain based resource management system,” January 2018.
- [21] G. Gürsoy, C. M. Brannon, and M. Gerstein, “Using ethereum blockchain to store and query pharmacogenomics data via smart contracts,” *BMC Medical Genomics*, vol. 13, June 2020.
- [22] F. Albalwy, J. H. McDermott, W. G. Newman, A. Brass, and A. Davies, “A blockchain-based framework to support pharmacogenetic data sharing,” *The Pharmacogenomics Journal*, vol. 22, pp. 264–275, July 2022.
- [23] P. J. Carabello, J. A. Sutton, J. Giri, J. A. Wright, W. T. Nicholson, I. J. Kullo, M. A. Parkulo, S. J. Bielinski, and A. M. Moyer, “Integrating pharmacogenomics into the electronic health record by implementing genomic indicators,” *Journal of the American Medical Informatics Association*, vol. 27, pp. 154–158, October 2019.
- [24] J. U. Adams, “Pharmacogenomics and personalized medicine,” *Nature Education*, vol. 1, p. 194, 2008.
- [25] M. Crosby, Nachiappan, P. Pattanayak, S. Verma, and V. Kalyanaraman, “Blockchain technology beyond bitcoin,” *Sutardja Center for Entrepreneurship and Technology Technical Report*, October 2015.
- [26] X. Li, Z. Wang, V. C. M. Leung, H. Ji, Y. Liu, and H. Zhang, “Blockchain-empowered data-driven networks: A survey and outlook,” *ACM Computing Surveys*, vol. 54, pp. 1–38, April 2022.

- [27] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain technology overview,” *National Institute of Standards and Technology*, October 2018.
- [28] B. Shrimali and H. B. Patel, “Blockchain state-of-the-art: architecture, use cases, consensus, challenges and opportunities,” *Journal of King Saud University - Computer and information Sciences*, August 2021.
- [29] G. Greenspan, “Multichain private blockchain — white paper.” <https://www.multichain.com/download/MultiChain-White-Paper.pdf>.
- [30] MultiChain, “Working with smart filters.” <https://www.multichain.com/developers/smart-filters/>.
- [31] MultiChain, “Download multichain community.” <https://www.multichain.com/download-community/>.

X. Appendix

A. Source Code

A..1 Django Files

Settings

```
"""
Django settings for multi_pgx project.

Generated by 'django-admin startproject' using Django 4.1.7.

For more information on this file, see
https://docs.djangoproject.com/en/4.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.1/howto/deployment/
# checklist/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = "django-insecure--2+e)v+#d@h7s+u+41o9jo3k3ujxn$_
_zhpe62fhnw5xwaw9u"

# SECURITY WARNING: don't run with debug turned on in
# production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "pgx",
]

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]

ROOT_URLCONF = "multi_pgx.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.
            DjangoTemplates",
        "DIRS": [],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.
                    messages",
            ],
        },
    },
]

WSGI_APPLICATION = "multi_pgx.wsgi.application"

# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases
```

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-
# password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.
            UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.
            MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.
            CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.
            NumericPasswordValidator",
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.1/topics/i18n/

LANGUAGE_CODE = "en-us"

TIME_ZONE = "UTC"

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.1/howto/static-files/

STATIC_URL = "static/"

# Default primary key field type
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-
# auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

MULTICHAIN_RPC = {
    "rpchost": "127.0.0.1", # Change to the IP address of your
        MultiChain node, if needed
    "rpcport": "6732", # Change to the port number specified in
        the 'params.dat' file
    "rpcuser": "multichainrpc",
    "rpcpasswd": "DLEBEN97YUVnn4psEojR49ZkEESoYBjZANGX4AQmM28",
        # Replace with the rpcpassword found in the '
        multichain.conf' file
    "chainname": "mychain",
}

from django.http import HttpResponseRedirect
from django.urls import reverse

def role_required(*roles):
    def decorator(view_func):
        def _wrapped_view(request, *args, **kwargs):
            user_role = request.session.get('role')
            if user_role not in roles:
                return HttpResponseRedirect(reverse('home')) #
                    redirect to home if the user doesn't have
                    the required role
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator
```

Decorators

```

    return _wrapped_view
return decorator

```

Utilities

```

import binascii
from datetime import datetime
import hashlib
import secrets
from Savoir import Savoir
from django.conf import settings
import json

rpcuser = settings.MULTICHAIN_RPC["rpcuser"]
rpcpasswd = settings.MULTICHAIN_RPC["rpcpasswd"]
rpchost = settings.MULTICHAIN_RPC["rpchost"]
rpcport = settings.MULTICHAIN_RPC["rpcport"]
chainname = settings.MULTICHAIN_RPC["chainname"]

multichain_api = Savoir(rpcuser, rpcpasswd, rpchost, rpcport,
                        chainname)

def publish_to_stream_with_offchain_data(stream_name, key,
                                         hex_data):
    options = "offchain"
    return multichain_api.publish(stream_name, key, hex_data,
                                  options)

def get_all_patient_data(stream_name):
    try:
        # get all items from the stream
        stream_items = multichain_api.liststreamitems(
            stream_name, False, 256)

        patients = []
        for item in stream_items:
            data_hex = item["data"]
            data_str = binascii.unhexlify(data_hex).decode()
            name, address, gene, drugid, iscore, annot, upload =
                data_str.split(":")
            dataid = f"{address}_{drugid}"
            patient_data = {
                "name": name,
                "address": address,
                "gene": gene,
                "drugid": drugid,
                "iscore": iscore,
                "annot": annot,
                "upload": upload,
                "dataid": dataid,
            }

            patients.append(patient_data)
        return patients
    except Exception as e:
        print(f"Error fetching patients: {str(e)}")
        return None

def grant_perm(address, name):
    multichain_api.grant(address, "activate")
    multichain_api.grant(address, "requester-test.write")
    multichain_api.grant(address, "user_credentials.write")
    multichain_api.grant(address, "pgx_data.write")
    access_data = {
        "org": name,
        "address": address,
        "timestamp": datetime.now().isoformat(),
        "status": "permissions granted", # or whatever access
        level is appropriate
    }
    # Convert to hex to publish on the blockchain
    data_hex = binascii.hexlify(json.dumps(access_data).encode()
                                ).decode()
    return multichain_api.publish("org_request", address,
                                  data_hex)

def grant_requester_perm(address):
    multichain_api.grant(address, "activate")
    return multichain_api.grant(address, "request-data.write")

def grant_patient_perm(address):
    multichain_api.grant(address, "activate")

    return multichain_api.grant(address, "access_tx.write")

def get_join_requests(stream_name):
    try:
        # get all items from the stream
        stream_items = multichain_api.liststreamitems(
            stream_name, False, 256)
        prev_name = ''
        prev_status = ''
        requests = []
        for item in reversed(stream_items):
            data_hex = item["data"]
            address = item["keys"][0]

```

```

data_str = binascii.unhexlify(data_hex).decode()
data_str = json.loads(data_str)

```

```

name = data_str.get("name", "org")
status = data_str["status"]
if (
    status == "No grants yet"
    and prev_status != "permissions granted"
    and prev_status != "No grants yet"
    and name != prev_name
):
    request = {
        "name": name,
        "address": address,
        "status": status,
    }
    requests.append(request)

prev_name = name
prev_status = status

```

```

return requests
except Exception as e:
    print(f"Error fetching patients: {str(e)}")
return None

```

```

def get_all_requests(stream_name, patient_address):
    try:
        # get all items from the stream
        stream_items = multichain_api.liststreamkeyitems(
            stream_name, patient_address)
        requests = []
        for item in reversed(stream_items):
            data_hex = item["data"]
            data_str = binascii.unhexlify(data_hex).decode()
            data_str = json.loads(data_str)
            status = data_str.get("status")
            name = data_str.get("name")
            organization = data_str.get("organization")
            address = data_str.get("requester")
            data_id = data_str.get("data")
            if not data_str.get("purpose"):
                purpose = "Research"
            else:
                purpose = data_str.get("purpose")

            if check_grant(organization, data_id) or data_id ==
                '':
                status = 'grant/deny'

            if status == "waitlisted":
                request = {
                    "name": name,
                    "organization": organization,
                    "address": address,
                    "data": data_id,
                    "purpose": purpose,
                    "status": status,
                }

                requests.append(request)

        return requests
    except Exception as e:
        print(f"Error fetching patients: {str(e)}")
        return None

```

```

def get_all_data(address):
    patient_data = multichain_api.liststreamkeyitems('pgx_data',
                                                    address)
    data_array = []
    for d in patient_data:
        d_hex = d["data"]
        d_str = binascii.unhexlify(d_hex).decode()
        name, address, gene, drug_id, iscore, annot, uploadedby = d_str
            .split(":")
        data_dict = {
            "name": name,
            "address": address,
            "gene": gene,
            "drugid": drug_id,
            "iscore": iscore,
            "annot": annot,
            "uploadedby": uploadedby,
        }
        data_array.append(data_dict)
    return data_array

```

```

def check_request(purpose, patient_address):
    request = multichain_api.liststreamkeyitems("request-data",
                                                patient_address)
    if request:
        for r in request:
            data = r["data"]
            d_str = binascii.unhexlify(data).decode()
            d = json.loads(d_str)
            p = d.get('purpose')
            if p == purpose:
                return True

```



```

return False

def check_grant(organization, dataid):
    grants = multichain_api.liststreamkeyitems("access_tx",
        organization)
    if grants:
        for grant in grants:
            grant = grant['data']
            grant = binascii.unhexlify(grant).decode()
            grant = json.loads(grant)
            data = grant.get('data_id')
            if dataid == data:
                return True
        return False
    else:
        return False

def check_deny(organization, purpose):
    access = multichain_api.liststreamkeyitems("access_tx",
        organization)
    if access:
        for deny in access:
            deny = deny['data']
            deny = binascii.unhexlify(deny).decode()
            deny = json.loads(deny)
            p = deny.get('purpose')
            status = deny.get('access_level')
            if p == purpose and status == 'deny':
                return True
        return False
    else:
        return False

def check_address(address):
    stream_items = multichain_api.liststreamitems("
        user_credentials", False, 256)

    for item in stream_items:
        data_hex = item["data"]
        data_str = binascii.unhexlify(data_hex).decode()
        add = data_str.split(":")[2]
        if add == address:
            return True

    return False

def check_name(name):
    stream_items = multichain_api.liststreamitems('patients',
        False, 256)
    isnamesame = False
    for item in stream_items:
        data_hex = item["data"]
        data_str = binascii.unhexlify(data_hex).decode()
        patient_name= data_str.split(":")[0]
        print(patient_name,name)
        if patient_name == name:
            isnamesame = True
    if isnamesame:
        return True
    else:
        return False

def publish_to_stream_from_address(org, stream_name, key,
    hex_data):
    options = "offchain"
    return multichain_api.publishfrom(org, stream_name, key,
        hex_data, options)

def publish_request(requester, stream_name, key, hex_data):
    options = "offchain"
    return multichain_api.publishfrom(requester, stream_name,
        key, hex_data, options)
    #return multichain_api.publish(stream_name, key, hex_data,
    options)

def get_all_granted(requester_address):
    req = multichain_api.liststreamkeyitems('requester-test',
        requester_address)
    req = req[0]['data']
    req = binascii.unhexlify(req).decode()
    org = req.split(":")[1]
    grant = multichain_api.liststreamkeyitems('access_tx', org)
    data_map = {}
    for item in grant:
        grant_data = item["data"]
        data_str = binascii.unhexlify(grant_data).decode()
        data_str = json.loads(data_str)
        status = data_str["access_level"]
        data = data_str["data_id"]
        patient_address = data_str["patient_address"]
        drugid = data.split("-")[1]
        patient_data = multichain_api.liststreamkeyitems('
            pgx_data', patient_address)
        for d in patient_data:
            d_hex = d["data"]
            d_str = binascii.unhexlify(d_hex).decode()
            name,address,gene,drug_id,iscore,annot,uploadedby=
                d_str.split(":")
            if drug_id == drugid:
                data_dict = {
                    "name": name,
                    "address":address,
                    "gene":gene,
                    "drugid":drug_id,
                    "iscore":iscore,
                    "annot":annot,
                    "uploadedby":uploadedby,
                    "status": status
                }
                if status == "grant":
                    data_map[data] = data_dict
                elif status == "revoke" and data in data_map:
                    del data_map[data]

    p_data = list(data_map.values())
    return p_data

def getallrequesterswithaccess(patient_address):
    grant = multichain_api.liststreamitems('access_tx', False,
        256)
    data_map = {}
    temp = ''
    for item in reversed(grant):
        grant_data = item["data"]
        data_str = binascii.unhexlify(grant_data).decode()
        data_str = json.loads(data_str)
        address = data_str["patient_address"]
        requesters = []
        if address == patient_address:
            status = data_str["access_level"]
            data = data_str["data_id"]
            if not data_str.get("purpose"):
                purpose = "Research"
            else:
                purpose = data_str.get("purpose")
            org = data_str["org"]
            reqorg = multichain_api.liststreamitems('requester-
                test', False, 256)
            for req in reqorg:
                reqhex = req['data']
                req = binascii.unhexlify(reqhex).decode()
                organization = req.split(':')[1]
                name = req.split(':')[0]
                if organization == org:
                    requesters.append(name)
            if status == "grant" and org!=temp:
                data_dict = {
                    "organization": org,
                    "requesters": requesters,
                    "data": data,
                    "purpose":purpose,
                    "status": status,
                }
                data_map[org] = data_dict
            elif status == "revoke":
                temp = org

    access_data = list(data_map.values())
    print(access_data)
    return access_data

def grant_access(patient_address, org, data_id, purpose):
    # Prepare the data
    access_data = {
        "patient_address": patient_address,
        "org": org,
        "data_id": data_id,
        "purpose": purpose,
        "timestamp": datetime.now().isoformat(),
        "access_level": "grant",
    }
    json_data = json.dumps(access_data)
    hex_data = json_data.encode().hex()
    return multichain_api.publishfrom(patient_address,"access_tx
        ", org, hex_data)

def deny_access(patient_address, org, data_id, purpose):
    # Prepare the data
    access_data = {
        "patient_address": patient_address,
        "org": org,
        "data_id": data_id,
        "purpose":purpose,
        "timestamp": datetime.now().isoformat(),
        "access_level": "deny",
    }
    # Convert to hex to publish on the blockchain
    json_data = json.dumps(access_data)
    hex_data = json_data.encode().hex()
    return multichain_api.publishfrom(patient_address,"access_tx
        ", org, hex_data)

def revoke_access(patient_address, org, data_id, purpose):
    # Prepare the data
    access_data = {
        "patient_address": patient_address,
        "org": org,
        "data_id": data_id,

```

```

        "purpose":purpose,
        "timestamp": datetime.now().isoformat(),
        "access_level": "revoke",
    }
    # Convert to hex to publish on the blockchain
    json_data = json.dumps(access_data)
    hex_data = json_data.encode().hex()
    return multichain_api.publishfrom(patient_address,"access_tx",
        org, hex_data)

def generate_salt():
    return secrets.token_hex(32)

def create_org_address():
    address = multichain_api.getnewaddress()
    permissions = "connect,send,receive"
    multichain_api.grant(address, permissions)
    return address

def create_address():
    address = multichain_api.getnewaddress()
    permissions = "connect"
    multichain_api.grant(address, permissions)
    return address

def hash_password(password, salt):
    return hashlib.sha256((password + salt).encode()).hexdigest()

def get_user_data(stream_name, key):
    items = multichain_api.liststreamkeyitems(stream_name, key)
    if items:
        items = list(reversed(items))
        data = items[0]["data"]
        return data
    return None

def get_publisher_address(stream_name, key):
    items = multichain_api.liststreamkeyitems(stream_name, key)
    if items:
        publisher = items[0]['publishers'][0]
        return publisher
    return None

def get_status(stream_name, key):
    items = multichain_api.liststreamkeyitems(stream_name, key)
    if items:
        return items
    return None

def get_access_control_tx():
    tx = multichain_api.liststreamitems('access_tx', False, 256)
    datalist = []
    for item in tx:
        txid = item['txid']
        data = item['data']
        data = bytes.fromhex(data).decode('utf-8')
        data = json.loads(data)
        p_address = data['patient_address']
        r_address = data['org']
        dataid = data['data_id']
        if not data.get("purpose"):
            purpose = "Research"
        else:
            purpose = data.get("purpose")
            timestamp = data['timestamp']
            timestamp = datetime.fromisoformat(timestamp)
            timestamp = timestamp.strftime("%B %d, %Y, %H:%M:%S")
            access_level = data['access_level']
            data_dict = {
                "txid": txid,
                "patient_address": p_address,
                "org": r_address,
                "dataid":dataid,
                "purpose": purpose,
                "timestamp":timestamp,
                "access_level":access_level,
            }
            datalist.append(data_dict)
    return datalist

def get_tx_org(name):
    tx = multichain_api.liststreamkeyitems('access_tx', name)
    print(tx)
    datalist = []
    for item in tx:
        txid = item['txid']
        data = item['data']
        data = bytes.fromhex(data).decode('utf-8')
        data = json.loads(data)
        p_address = data['patient_address']
        dataid = data['data_id']
        timestamp = data['timestamp']
        timestamp = datetime.fromisoformat(timestamp)
        timestamp = timestamp.strftime("%B %d, %Y, %H:%M:%S")
        access_level = data['access_level']
        if not data.get("purpose"):
            purpose = "Research"
        else:
            purpose = data.get("purpose")
        patient = get_user_data('patients', p_address)

    if not patient:
        patient=get_user_data('patients', 'gab')
        patient_str = binascii.unhexlify(patient).decode()
        name = patient_str.split(':')[0]
        data_dict = {
            "txid": txid,
            "patient_name": name,
            "dataid":dataid,
            "purpose": purpose,
            "timestamp":timestamp,
            "access_level":access_level,
        }
        datalist.append(data_dict)
    return datalist

def get_tx_patient(address):
    tx = multichain_api.liststreamitems('access_tx', False, 256)
    datalist = []
    for item in tx:
        txid = item['txid']
        data = item['data']
        data = bytes.fromhex(data).decode('utf-8')
        data = json.loads(data)
        print(data)
        p_address = data['patient_address']
        r_address = data['org']
        requester_data = multichain_api.liststreamkeyitems('requester-test', r_address)
        if not requester_data:
            requester_data = multichain_api.liststreamkeyitems('requester-test', 'requester1')
            requester_hex = requester_data[0]["data"]
            requester_str = binascii.unhexlify(requester_hex).decode()
            name, org = requester_str.split(":")
        elif requester_data:
            requester_hex = requester_data[0]["data"]
            requester_str = binascii.unhexlify(requester_hex).decode()
            name, org, addr = requester_str.split(":")
        dataid = data['data_id']
        timestamp = data['timestamp']
        if not data.get("purpose"):
            purpose = "Research"
        else:
            purpose = data.get("purpose")
            timestamp = datetime.fromisoformat(timestamp)
            timestamp = timestamp.strftime("%B %d, %Y, %H:%M:%S")
            access_level = data['access_level']
            if address == p_address:
                data_dict = {
                    "txid": txid,
                    "name": name,
                    "org": org,
                    "dataid":dataid,
                    "purpose": purpose,
                    "timestamp":timestamp,
                    "access_level":access_level,
                }
                datalist.append(data_dict)
    return datalist

def org_status(address):
    items = multichain_api.liststreamkeyitems('org_request', address)
    if items:
        items = list(reversed(items))
        data = items[0]["data"]
        data = binascii.unhexlify(data).decode()
        data = json.loads(data)
        status = data['status']
        return status
    return None

```

Views

```

import binascii
import json
import pandas as pd
import csv
from django.http import JsonResponse, HttpResponseRedirect, HttpResponse
from django.urls import reverse
from django.contrib import messages
from django.shortcuts import redirect, render
from .multichain_utils import *
from .decorators import role_required

from .forms import *

def index(request):
    return redirect('login')

def home(request):
    user = request.session.get('user')
    role = request.session.get('role')

    if user is None or role is None:

```

```

        # User is not authenticated, redirect them to the login
        page
        return redirect('login')

    if role == 'admin':
        return redirect('join_request')
    elif role == 'organization':
        return redirect('profile')
    elif role == 'Patient':
        return redirect('patient_request')
    elif role == 'Auditor':
        return redirect('view_trans')
    elif role == 'requester':
        return redirect('request_view')

def register_user(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        name = request.POST.get('name')
        role = request.POST.get('role')
        salt = generate_salt()
        password_hash = hash_password(password, salt)

        # Create a new MultiChain address for this user
        address = create_address()

        # Store the username, hashed password, and address
        together
        data = f"{salt}:{password_hash}:{address}:{role}"
        data_hex = binascii.hexlify(data.encode()).decode()

        # Publish the user's credentials and address to the
        # user_credentials' stream
        publish_to_stream_with_offchain_data('user_credentials',
            username, data_hex)

        # Store the user's name and role on the 'user_profiles'
        stream
        if role == 'Patient':
            profile_data = f"{name}:{address}"
            profile_data_hex = binascii.hexlify(profile_data.
                encode()).decode()
            grant_patient_perm(address)
            publish_to_stream_with_offchain_data('patients',
                address, profile_data_hex)
            request.session['user'] = username
            request.session['role'] = role
            request.session['address'] = address
            messages.success(request, "Your account has been
                successfully registered.")
            return redirect('home')
        elif role == 'Auditor':
            profile_data = f"{name}:{address}"
            profile_data_hex = binascii.hexlify(profile_data.
                encode()).decode()
            publish_to_stream_with_offchain_data('auditors',
                address, profile_data_hex)
            request.session['user'] = username
            request.session['role'] = role
            request.session['address'] = address
            messages.success(request, "Your account has been
                successfully registered.")
            return redirect('home')
        elif role == 'organization':
            profile_data = f"{name}:{address}"
            profile_data_hex = binascii.hexlify(profile_data.
                encode()).decode()
            req_data = {'name':name,'status': 'No grants yet'}
            req_hex = binascii.hexlify(json.dumps(req_data).
                encode()).decode()
            publish_to_stream_with_offchain_data('organizations',
                address, profile_data_hex)
            publish_to_stream_with_offchain_data('org_request',
                address, req_hex)
            request.session['user'] = username
            request.session['role'] = role
            request.session['address'] = address
            messages.success(request, "Your account has been
                successfully registered.")
            return redirect('home')
        else:
            messages.error(request, "Please select a role.")
            return redirect('register_user')
    return render(request, 'register.html')

def profile(request):
    role = request.session.get('role')
    user = request.session.get('user')
    address = request.session.get('address')
    status = ""
    if role == 'admin':
        user_data = get_user_data('organizations', user)
        if not user_data:
            user_data = get_user_data('organizations', address)
        orgrequest = get_user_data('org_request', address)
        status_hex = binascii.unhexlify(orgrequest).decode()
        status_hex = json.loads(status_hex)
        status = status_hex['status']

        data_str = binascii.unhexlify(user_data).decode()
        name = data_str.split(':')[0]

        context = {'role': role, 'status': status, 'name': name,
            'address': address}
    elif role == 'Patient':
        user_data = get_user_data('patients', user)
        if not user_data:
            user_data = get_user_data('patients', address)
        data_str = binascii.unhexlify(user_data).decode()
        name = data_str.split(':')[0]

        context = {'role': role, 'status': status, 'name': name,
            'address': address}
    elif role == 'Auditor':
        user_data = get_user_data('auditors', user)
        if not user_data:
            user_data = get_user_data('auditors', address)
        data_str = binascii.unhexlify(user_data).decode()
        name = data_str.split(':')[0]

        context = {'role': role, 'status': status, 'name': name,
            'address': address}
    elif role == 'requester':
        user_data = get_user_data('requester-test', user)
        if not user_data:
            user_data = get_user_data('requester-test', address)
        data_str = binascii.unhexlify(user_data).decode()
        name = data_str.split(':')[0]
        organization = data_str.split(':')[1]
        context = {'role': role, 'status': status, 'name': name,
            'address': address, 'organization': organization}

    return render(request, 'profile.html', context)

@role_required('admin')
def join_request_view(request):
    role = request.session.get('role')
    address = request.session.get('address')
    status = ""
    requests = get_join_requests('org_request')
    return render(request, 'org_request.html', {'requests':
        requests, "role":role, 'status': status, 'address':
        address})

@role_required('admin')
def grant_permissions(request, name, address):
    if request.method == 'POST':
        try:
            grant_perm(address, name)
            messages.success(request, f"Permissions have been
                granted to {address}")
            return redirect('join_request')
        except Exception as e:
            messages.error(request, str(e))
            return redirect('join_request')
    else:
        messages.error(request, "Invalid request")
        return redirect('join_request')

@role_required('admin', 'organization')
def create_data_requester(request):
    role = request.session.get('role')
    user = request.session.get('user')
    orgadd = request.session.get('address')
    status = ""
    if role == "organization":
        status = org_status(orgadd)
    if role == "admin":
        user_data = get_user_data('organizations', user)
    else:
        user_data = get_user_data('organizations', orgadd)
    org = ""
    if user is None:
        return redirect('login')
    if user_data:
        if role == "admin":
            org = binascii.unhexlify(user_data).decode()
        else:
            org = binascii.unhexlify(user_data).decode()
            org = org.split(':')[0]
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        name = request.POST.get('name')
        organization = org
        role = "requester"
        salt = generate_salt()
        password_hash = hash_password(password, salt)

```

```

# Create a new MultiChain address for this user
address = create_address()

# Store the username, hashed password, and address
together
data = f"{salt}:{password_hash}:{address}:{role}"
data_hex = binascii.hexlify(data.encode()).decode()

profile_data = f"{name}:{organization}:{address}"
profile_data_hex = binascii.hexlify(profile_data.encode()
()).decode()

grant_requester_perm(address)

publish_to_stream_from_address(orgadd, 'requester-test',
address, profile_data_hex)

# Publish the user's credentials and address to the '
user_credentials' stream
publish_to_stream_from_address(orgadd, 'user_credentials
', username, data_hex)

messages.success(request, f"User with address {address}
has been created.")
return redirect('create_user')

return render(request, 'createrequester.html', {"role":role,
'status': status})

def authenticate_user(request):
if request.method == 'POST':
username = request.POST.get('username')
password = request.POST.get('password')

data_hex = get_user_data('user_credentials', username)
if data_hex:
data_str = binascii.unhexlify(data_hex).decode()
stored_salt, stored_password_hash, address, role =
data_str.split(':')
password_hash = hash_password(password, stored_salt)

if stored_password_hash == password_hash:
request.session['user'] = username
request.session['role'] = role
request.session['address'] = address
# Redirect user to home page
messages.success(request, "You are now logged in
as " + str(address))
return HttpResponseRedirect(reverse('home'))
else:
messages.error(request, "Invalid password!")
return redirect('login')

else:
messages.error(request, "Username not found!")
return redirect('login')

return render(request, 'login.html')

@role_required('admin', 'organization')
def upload_data(request):
role = request.session.get('role')
user = request.session.get('user')
orgadd = request.session.get('address')
status = ""
org = get_user_data('organizations', user)
if org:
org = binascii.unhexlify(org).decode()
else:
org = get_user_data('organizations', orgadd)
org = binascii.unhexlify(org).decode()
org = org.split(":")[0]
if role == "organization":
status = org_status(orgadd)
if request.method == 'POST' and request.FILES['csv_file']:
csv_file = request.FILES['csv_file']
if not csv_file.name.endswith('.csv'):
return HttpResponseRedirect('Invalid file format.
Please upload a CSV file.')

# Process the CSV file
file = pd.read_csv(csv_file).iloc[:, [0,1,2,3]]
gene = file['Gene'].tolist()
drugid = file['Drugs'].tolist()
iscore = file['Interaction Score'].tolist()
annot = file['Annotation'].tolist()
name = request.POST.get('name')
address = request.POST.get('address')

uploaded_by = org
if not check_address(address) and not check_name(name):
messages.error(request, "Address and name not found
.")
return redirect('upload_data')
elif not check_address(address):
messages.error(request, "Address not found.")
return redirect('upload_data')
elif not check_name(name):
messages.error(request, "Name not found.")
return redirect('upload_data')
else:
for i in range(len(file)):
data = f"{name}:{address}:{gene[i]}:{drugid[i]}:{
iscore[i]}:{annot[i]}:{uploaded_by[i]}"
data_hex = binascii.hexlify(data.encode()).decode()
publish_to_stream_from_address(orgadd, 'pgx_data',
address, data_hex)
messages.success(request, f"Patient {address} pgx
data was uploaded.")
return HttpResponseRedirect(reverse('upload_data'))

return render(request, 'createdata.html', {"role":role, '
status': status})
@role_required("Patient")
def patient_request_view(request):
role = request.session.get('role')
address = request.session.get('address')
status = ""
requests = get_all_requests('request-data', address)
return render(request, 'patient_request_view.html', {'
requests':requests, "role":role, 'status': status})

@role_required("Patient")
def manage_access_view(request):
role = request.session.get('role')
address = request.session.get('address')
status = ""
requesters = get_all_requesters_with_access(address)
return render(request, 'manage_access.html', {'requesters':
requesters, "role":role, 'status': status})

@role_required("Patient")
def grant_data_access(request, organization, data_id, purpose):
address = request.session.get('address')
if request.method == 'POST':
try:
grant_access(address, organization, data_id, purpose)
messages.success(request, f"You have granted {
organization} access to your data for the
research/clinical trial {purpose}.")
return redirect('manage_access')
except Exception as e:
messages.error(request, str(e))
return redirect('patient_request')
else:
messages.error(request, "Invalid Request")
return redirect('patient_request')

@role_required("Patient")
def revoke_data_access(request, organization, data_id, purpose):
address = request.session.get('address')
if request.method == 'POST':
try:
revoke_access(address, organization, data_id, purpose)
messages.success(request, f"You have revoked access
of {organization} for {purpose}")
return redirect('manage_access')
except Exception as e:
messages.error(request, str(e))
return redirect('patient_request')
else:
messages.error(request, "Invalid Request")
return redirect('patient_request')

@role_required("Patient")
def deny_data_access(request, organization, data_id, purpose):
address = request.session.get('address')
if request.method == 'POST':
try:
deny_access(address, organization, data_id, purpose)
messages.success(request, f"You have denied access to
{organization} for {purpose}")
return redirect('manage_access')
except Exception as e:
messages.error(request, str(e))
return redirect('patient_request')
else:
messages.error(request, "Invalid Request")
return redirect('patient_request')

@role_required("requester")
def request_view(request):
status = ""
role = request.session.get('role')
address = request.session.get('address')
patients = get_all_patient_data('pgx_data')
filtered_patients = patients

select_gene= []
select_drug= []

for p in patients:

```

```

gn = p.get('gene')
dg = p.get('drugid')
if gn not in select_gene:
    select_gene.append(gn)
if dg not in select_drug:
    select_drug.append(dg)

gene = request.GET.get('gene_id')
drug = request.GET.get('drug_id')
if gene:
    filtered_patients = [t for t in filtered_patients if t.get('gene') == gene]
if drug:
    filtered_patients = [t for t in filtered_patients if t.get('drugid') == drug]
patients = filtered_patients
for patient in patients:
    data = patient['dataid']
    items = get_status('access_tx', address)
    in_list = False
    if items:
        for item in items:
            item = item['data']
            item = binascii.unhexlify(item).decode()
            item = json.loads(item)
            if item['data_id'] == data:
                in_list = True
                continue
    if in_list:
        patients.remove(patient)
context = {'pd':patients, 'role':role, 'status': status, 'drugs':select_drug, 'genes':select_gene}
return render(request, 'request_view.html', context)

@role_required("requester")
def request_data(request):
    user = request.session.get('user')
    address = request.session.get('address')
    user_data = get_user_data('requester-test', user)
    if not user_data:
        user_data = get_user_data('requester-test', address)
    data_str = binascii.unhexlify(user_data).decode()
    name, org, add = data_str.split(':')
    else:
        data_str = binascii.unhexlify(user_data).decode()
        name, org = data_str.split(':')
    if request.method == 'POST':
        purpose = request.POST.get('purpose')
        data_list = request.POST.getlist('selected_patients')
        already_requested = len(data_list)
        for data in data_list:
            if data != '':
                patient_address, data_id = data.split('|')
                if check_request(purpose, patient_address):
                    messages.error(request, f"Your request of {data_id} for {purpose} has already been sent")
                    already_requested -= 1
                    continue
                if check_deny(org, purpose):
                    messages.error(request, f"Your request of {data_id} for {purpose} has been denied")
                    already_requested -= 1
                    continue
                req_data = {'name':name, 'organization':org, 'requester':address, 'data':data_id, 'purpose':purpose, 'status': 'waitlisted'}
                data_hex = binascii.hexlify(json.dumps(req_data).encode()).decode()
                publish_request(address, 'request-data', patient_address, data_hex)
            if already_requested > 1:
                messages.success(request, f"Your request/s has/have been sent")
            return redirect('request_view')
        else:
            messages.error(request, "Invalid request")
            return redirect('request_view')

@role_required("requester")
def view_data_table(request):
    status = ""
    role = request.session.get('role')
    data = get_all_granted(request.session.get("address"))
    return render(request, "data_table.html", {'data':data, "role":role, 'status': status})

@role_required("requester")
def download_data(request):
    # Get the accessed data for the requester
    accessed_data = get_all_granted(request.session.get("address"))

# Prepare the data as a table
table_data = []
headers = ["Name", "Address", "Gene", "Drug/s", "Interaction Score", "Annotation"]
table_data.append(headers)
for data in accessed_data:
    table_data.append([data['name'], data['address'], data['gene'], data['drugid'], data['iscore'], data['annot']]) # Replace with your actual data fields

response = HttpResponse(content_type='text/csv')
response['Content-Disposition'] = 'attachment; filename="pgx_data.csv"'

writer = csv.writer(response)
writer.writerows(table_data)

return response

@role_required("Patient")
def view_data_table_patient(request):
    status = ""
    role = request.session.get('role')
    data = get_all_data(request.session.get("address"))
    return render(request, "data_table_patient.html", {'data':data, "role":role, 'status': status})

@role_required("Auditor")
def transaction_view(request):
    status = ""
    transactions = get_access_control_tx()
    role = request.session.get('role')
    transactions = transactions[::-1]
    select_patient = []
    select_access = []
    for t in transactions:
        patient = t.get('patient_address')
        access = t.get('access_level')
        if patient not in select_patient:
            select_patient.append(patient)
        if access not in select_access:
            select_access.append(access)
    filtered_transactions = transactions
    patient_address = request.GET.get('patient_address')
    access_level = request.GET.get('access_level')
    if patient_address:
        filtered_transactions = [t for t in filtered_transactions if t.get('patient_address') == patient_address]
    if access_level:
        filtered_transactions = [t for t in filtered_transactions if t.get('access_level') == access_level]

    context = {
        'transactions':filtered_transactions,
        'role':role,
        'status': status,
        'patients':select_patient,
        'access':select_access,
    }
    return render(request, 'transactions.html', context)

@role_required("Patient")
def patient_transaction_view(request):
    status = ""
    address = request.session.get('address')
    transactions = get_tx_patient(address)
    role = request.session.get('role')
    transactions = transactions[::-1]
    select_requester = []
    select_access = []
    select_org = []
    for t in transactions:
        requester = t.get('name')
        access = t.get('access_level')
        org = t.get('org')
        if requester not in select_requester:
            select_requester.append(requester)
        if access not in select_access:
            select_access.append(access)
        if org not in select_org:
            select_org.append(org)

    filtered_transactions = transactions
    requester_name = request.GET.get('name')
    access_level = request.GET.get('access_level')
    organization = request.GET.get('org')
    if requester_name:
        filtered_transactions = [t for t in filtered_transactions if t.get('name') == requester_name]
    if organization:
        filtered_transactions = [t for t in filtered_transactions if t.get('org') == organization]
    if access_level:

```

```

        filtered_transactions = [t for t in
            filtered_transactions if t.get('access_level') ==
            access_level]

context = {
    'transactions':filtered_transactions,
    'role':role,
    'status': status,
    'names':select_requester,
    'access':select_access,
    'orgs':select_org,
}
return render(request, 'transactions_patient.html', context)

@role_required("organization", "requester")
def org_transaction_view(request):
    status = ""
    user = request.session.get('user')
    role = request.session.get('role')
    address = request.session.get('address')
    if role == 'requester':
        req = get_user_data('requester-test', address)
        req_str = binascii.unhexlify(req).decode()
        name = req_str.split(':')[1]
        add = get_publisher_address('requester-test', address)
        address = add
        print(user,address)
    else:
        user_data = get_user_data('organizations', user)
        if not user_data:
            user_data = get_user_data('organizations', address)
            data_str = binascii.unhexlify(user_data).decode()
            name = data_str.split(':')[0]
            orgrequest = get_user_data('org_request', address)
            status_hex = binascii.unhexlify(orgrequest).decode()
            status_hex = json.loads(status_hex)
            status = status_hex['status']
            transactions = get_tx_org(name)
            role = request.session.get('role')
            transactions = transactions[::-1]
            select_patient= []
            select_access = []
            for t in transactions:
                patient = t.get('patient_name')
                access = t.get('access_level')
                if patient not in select_patient:
                    select_patient.append(patient)
                if access not in select_access:
                    select_access.append(access)
            filtered_transactions = transactions
            patient_name = request.GET.get('patient_name')
            access_level = request.GET.get('access_level')
            if patient_name:
                filtered_transactions = [t for t in
                    filtered_transactions if t.get('patient_name') ==
                    patient_name]
            if access_level:
                filtered_transactions = [t for t in
                    filtered_transactions if t.get('access_level') ==
                    access_level]

context = {
    'transactions':filtered_transactions,
    'role':role,
    'status': status,
    'patients':select_patient,
    'access':select_access,
}
}

}
return render(request, 'transactions_requester.html',
context)

def logout(request):
    request.session.flush() # This will delete all current
    session data
    return redirect('login') # Redirect to the login page

```

URLS

```

from django.urls import path

from django.conf.urls.static import static
from django.conf import settings

from . import views

urlpatterns = [
    path("", views.index , name="index"),
    path("home/", views.home, name="home"),
    path("login/", views.authenticate_user, name="login"),
    path("logout/", views.logout, name="logout"),
    path("profile/", views.profile, name="profile"),
    path("register/", views.register_user, name="register_user")
    ],
    path("check_join_requests/", views.join_request_view, name="
join_request"),
    path("grant_permissions/<str:name>/<str:address>/", views.
grant_permissions, name="grant_permissions"),
    path("create_user/", views.create_data_requester, name="
create_user"),
    path("upload_data/", views.upload_data, name="upload_data"),
    path("request_view/", views.request_view, name="request_view
"),
    path("request_data/", views.request_data, name="request_data
"),
    path("data_table/", views.view_data_table, name="data_table
"),
    path("download_data/", views.download_data, name="
download_data"),
    path("data_table_patient/", views.view_data_table_patient,
name="data_table_patient"),
    path("patient_request_data/", views.patient_request_view,
name="patient_request"),
    path("manage_access", views.manage_access_view, name="
manage_access"),
    path("grant_access/<str:organization>/<str:data_id>/<str:
purpose>", views.grant_data_access, name="
grant_access"),
    path("deny_access/<str:organization>/<str:data_id>/<str:
purpose>", views.deny_data_access, name="deny_access
"),
    path("revoke_access/<str:organization>/<str:data_id>/<str:
purpose>", views.revoke_data_access, name="
revoke_access"),
    path("viewtransactions/", views.transaction_view, name="
view_trans"),
    path("patient_transactions/", views.patient_transaction_view
, name="patient_trans_view"),
    path("org_transactions/", views.org_transaction_view, name="
org_trans_view"),
]

```

A..2 Templates

Base Template

```

<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0, shrink-to-fit=no">
<title>{% block title %}{% endblock title %}</title>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
bootstrap@5.2.3/dist/css/bootstrap.min.css">
{% block extracss %}
{% endblock extracss %}
</head>

<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-dark py
-3">
<div class="container-fluid">
<a class="navbar-brand" href="{% url 'home' %}">
<svg xmlns="http://www.w3.org/2000/svg" viewBox
="0 0 24 24" fill="none" stroke="white"
stroke-width="2" stroke-linecap="round"
stroke-linejoin="round" class="feather
feather-blockchain" style="width: 40px;
height: 40px;">
<path d="M12 2L2 7.8v8.4L12 22l10-5.8V7.8L12
22M22 12l-4.5 2.6m0-5.2L22 12m-9-3.5L7
.5 12m0-5.2L12 12m0-9.3V5"></path>
</svg>
PGChain
</a>
<button data-bs-toggle="collapse" class="navbar-
toggler" data-bs-target="#navcol-1"><span
class="visually-hidden">Toggle navigation</
span><span class="navbar-toggler-icon"></span
></button>
<div class="collapse navbar-collapse" id="navcol-1">
<ul class="navbar-nav ms-auto">
{% if role == "Patient" %}
<li class="nav-item"><a class="nav-link"
href="{% url 'patient_request' %}"><span
class="visually-hidden">Toggle navigation</
span><span class="navbar-toggler-icon"></span
></button>
<div class="collapse navbar-collapse" id="navcol-1">
<ul class="navbar-nav ms-auto">
{% if role == "Patient" %}
<li class="nav-item"><a class="nav-link"
href="{% url 'patient_request' %}">
Check Requests</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'manage_access' %}">
Manage</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'data_table_patient'
%}">View Data</a></li>

```

```

<li class="nav-item"><a class="nav-link"
href="{% url 'profile' %}">Account
</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'patient_trans_view'
%}">View Transactions</a></li>
{% elif role == "requester" %}
<li class="nav-item"><a class="nav-link"
href="{% url 'data_table' %}">View
Data</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'request_view' %}">
Request</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'org_trans_view' %}">
View Transactions</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'profile' %}">Account
</a></li>
{% elif role == "Auditor" %}
<li class="nav-item"><a class="nav-link"
href="{% url 'view_trans' %}">View
Transactions</a></li>
{% else %}
{% if role == "admin" %}
<li class="nav-item"><a class="nav-
link" href="{% url 'join_request
' %}">Check Join Requests</a></
li>
{%endif%}
{% if status == "permissions granted" or
role == "admin" %}
<li class="nav-item"><a class="nav-link"
href="{% url 'upload_data' %}">
Upload Data</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'create_user' %}">
Create User</a></li>
<li class="nav-item"><a class="nav-link"
href="{% url 'org_trans_view' %}">
View Transactions</a></li>
{% endif %}
<li class="nav-item"><a class="nav-link"
href="{% url 'profile' %}">Account
</a></li>
{% endif %}
<li class="nav-item"><a class="nav-link" href
="{% url 'logout' %}">Logout</a></li>
</ul>
</div>
</nav>
<main>
{% include 'message.html' %}
{% block content %}
{% endblock content %}
</main>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/
dist/js/bootstrap.bundle.min.js"></script>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></
script>
<script>
$(document).ready(function() {
// Close the alert when the close button is clicked
$(".msg .close").on("click", function() {
$(this).closest(".msg").alert("close");
});
// Automatically disappear after 5 seconds
$(".msg").delay(5000).fadeOut("slow");
});
</script>
{% block extrascripts %}
{% endblock extrascripts %}
</body>
</html>

```

```

<link href="https://fonts.googleapis.com/css?family=Poppins
:300,400,500,600,700,800,900" rel="stylesheet">
<script src="https://kit.fontawesome.com/70e2a3091b.js"
crossorigin="anonymous"></script>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
@fontawesome/fontawesome-free@6.1.1/css/fontawesome.
min.css">
<title>Register</title>
<style>
</style>
</head>
<body>
<div class="container" style="position: absolute;left: 0;
right: 0;top: 50%;transform: translateY(-50%);-ms-
transform: translateY(-50%);-moz-transform:
translateY(-50%);-webkit-transform: translateY(-50%)
;-o-transform: translateY(-50%);">
<div class="row d-flex d-xl-flex justify-content-center
justify-content-xl-center">
<div class="col-sm-12 col-lg-10 col-xl-9 col-xxl-7 bg
-white shadow-lg" style="border-radius: 5px;">
<div class="p-5">
<div class="text-center">
<h4 class="text-dark mb-4">Create an
Account!</h4>
</div>
<form method="POST" class="user">
{% csrf_token %}
<div class="mb-3">
<select class="form-select" name="role"
aria-label="Default select
example">
<option value = "None" selected
disabled hidden>Select
Designation</option>
<option value="Patient">Patient</
option>
<option value="Auditor">Auditor</
option>
<option value="organization">
Organization</option>
</select>
</div>
<div class="mb-3"><input class="form-
control form-control-user" type="
text" name="username" placeholder="
Username" required /></div>
<div class="mb-3"><input id="email" class
="form-control form-control-user"
name="name" type="text" placeholder
="Name" required /></div>
<div class="row mb-3">
<div class="col-sm-6 mb-3 mb-sm-0"><
input id="password" class="form-
control form-control-user" type
="password" name="password1"
placeholder="Password" required
/></div>
<div class="col-sm-6"><input id="
verifyPassword" class="form-
control form-control-user" type
="password" name="password2"
placeholder="Repeat Password"
required /></div>
</div>
<div class="row mb-3">
<p id="passwordErrorMsg" class="text-
danger" style="display: none;">
Password does not match</p>
</div><button id="submitBtn" class="btn
btn-primary d-block btn-user w-100"
type="submit">Register Account</
button>
<hr />
</form>
<div class="text-center"><a class="small"
href="{% url 'login' %}">Already have
an account? Login!</a></div>
</div>
</div>
</div>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/
dist/js/bootstrap.bundle.min.js"
integrity="sha384-ka7Sk0l4gamtz2M1QnikT1wXgYs0g+OMhuP+
1LRH9sENB00Lrn5q+8nbTov4+1p"
crossorigin="anonymous"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery
/3.4.1/jquery.min.js"></script>
<script>
// Function to check if passwords match and display
error message
function checkPasswordMatch() {
var password = document.getElementById("password").
value;
var verifyPassword = document.getElementById("

```

Register

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/
dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-1
BmE4kWBq78iYhF1dvKuhfTAU6au08tT94rWfhtjDbrCEXSU1oBoqyl2QvZ6j1W3
" crossorigin="anonymous">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
bootstrap-icons@1.3.0/font/bootstrap-icons.css">

```



```

        <button type="button" class="btn-close" data
        -bs-dismiss="modal" aria-label="Close
        "></button>
    </div>
    <div class="modal-body">
    <form id="requestAccessForm" method="post"
    action="{% url 'request_data' %}">
    {% csrf_token %}
    <div class="form-group">
    <label for="purpose">Purpose:</label>
    <input type="text" class="form-control" id
    ="purpose" name="purpose" required>
    </div>
    <input type="hidden" name="selected_patients
    " value="" id="selectedPatientsInput
    ">
    <div class="modal-footer">
    <button type="submit" id = "btn" class="
    btn btn-primary">Request Access</
    button>
    <button type="button" class="btn btn-
    secondary" data-dismiss="modal">
    Cancel</button>
    </div>
    </form>
    </div>
</div>
</div>
</div>
</form>

    </div>
    <div class="row justify-content-end">
    <div class="col">
    <h3>No data to request</h3>
    </div>
    </div>
    {%endif%}
</div>

{% endblock content %}

{% block extrascripts %}
<script>
$(document).ready(function() {
    $('#requestAccessModal').on('shown.bs.modal', function() {
        $('#purpose').focus(); // Focus on the purpose input field
        when the modal is shown
        updateRequestButton(); // Update the state of the request
        button on modal shown
    });

    // Update the state of the request button whenever the
    purpose input value changes
    $('#purpose').on('input', function() {
        updateRequestButton();
    });

    $('#requestAccessForm').submit(function(e) {
        e.preventDefault(); // Prevent the form from submitting
        normally

        var selectedPatients = [];
        $('#input[name="selected_patients"]:checked').each(function
        () {
            selectedPatients.push($(this).val());
        });

        var purpose = $('#purpose').val();

        // Set the selected patients value in the hidden input
        field
        $('#selectedPatientsInput').val(selectedPatients.join(', '))
        );

        // Perform any validation or additional checks on the
        purpose if needed

        // Update the form action URL with the purpose value
        $(this).attr('action', $(this).attr('action') + '?purpose
        =' + encodeURIComponent(purpose));

        // Submit the form
        this.submit();
    });

    // Function to update the state of the request button based
    on the purpose input value
    function updateRequestButton() {
        var purpose = document.getElementById("purpose").value;
        var requestButton = $('#requestAccessForm button[type="
        submit"]');

        if (purpose == '') {
            document.getElementById("btn").disabled = true;
        } else {
            document.getElementById("btn").disabled = false; //
            Disable the request button
        }
    }
});
</script>

{% endblock extrascripts %}

Check Request

{% extends 'index.html' %}
{% block title %}Request View{% endblock title %}
{% block content %}

<div class = "container-xl px-1 pt-4">
<h1 class="text-center pb-4">Check Requests</h1>
{% if requests %}
<table id ='booking-table' class="table ">
<thead>
<tr>
<th scope="col" class = "text-center">Name</th>
<th scope="col" class = "text-center">Address</th>
<th scope="col" class = "text-center">Organization</
th>
<th scope="col" class = "text-center">Data</th>
<th scope="col" class = "text-center">Purpose</th>
<th scope="col"></th>
<th scope="col"></th>
</tr>
</thead>
<tbody>
{% for req in requests%}
<tr>
<td class = "py-3 text-center">{{req.name}}</td>
<td class = "py-3 text-center">{{req.address}}</td>
<td class = "py-3 text-center">{{req.organization}}</
td>
<td class = "py-3 text-center">{{req.data}}</td>
<td class = "py-3 text-center">{{req.purpose}}</td>
<td class = "py-3 pl-5 text-center">
<a class="btn btn-success text-center border
border-dark" href="#" role="button" data-bs
-toggle="modal" data-bs-target="#grantModal
">Grant</a>
</td>
<td class = "py-3 pl-5 text-center">
<a class="btn btn-danger text-center border
border-dark" href="#" role="button" data-bs
-toggle="modal" data-bs-target="#denyModal
">Deny</a>
</td>
</tr>
</tbody>
<!-- Request modal -->
<div class="modal fade" id="grantModal" tabindex="-1"
role="dialog" aria-labelledby="grantModalLabel"
aria-hidden="true">
<div class="modal-dialog" role="document">
<div class="modal-content">
<div class="modal-header">
<h5 class="modal-title" id="
grantModalLabel">Consent</h5>
<button type="button" class="btn-close"
data-bs-dismiss="modal" aria-label
="Close"></button>
</div>
<div class="modal-body">
<p> I hereby grant access to my
pharmacogenomic data to <strong
>{{req.organization}}</strong>
for the research/clinical trial
"<strong>{{req.purpose}}</strong
">. I understand that my data
will be used in accordance with
all applicable privacy laws and
regulations, and will only be
accessed by authorized
individuals involved in my
healthcare.</p>
<p>I acknowledge that granting this
access is voluntary, and I have
been provided with sufficient
information about the purpose
and potential risks of sharing
my data. I understand that I
have the right to revoke this
access at any time.</p>
<p>By clicking the "Grant Access"
button below, I consent to the
above terms and conditions.</p>
</div class="modal-footer">

```



```

        <a class="btn btn-danger text-center border
        border-dark" href="#" role="button" data-bs-
        toggle="modal" data-bs-target="#"
        revokeModal">Revoke</a>
    </td>
</tr>
<div class="modal fade" id="revokeModal" tabindex="-1"
role="dialog" aria-labelledby="revokeModalLabel"
aria-hidden="true">
<div class="modal-dialog" role="document">
<div class="modal-content">
<div class="modal-header">
<h5 class="modal-title" id="
revokeModalLabel">Revoke Access</h5
>
<button type="button" class="btn-close"
data-bs-dismiss="modal" aria-label
="Close"></button>
</div>
<div class="modal-body">

    <p> Are you sure you want to revoke
    access for this data?</p>
<div class="modal-footer">
<form method="post" action="{% url '
revoke_access' req.organization
req.data req.purpose %}">
    {% csrf_token %}
<button type="button" class="btn
btn-danger" data-bs-dismiss
="modal">No</button>
<button class="btn btn-success text
-center" type="submit">Yes</
button>

    </form>
</div>
</div>
</div>
</div>
</div>
<div class="modal fade" id="viewrequesters" tab-index
="-1">
<div class="modal-dialog modal-dialog-centered modal-
dialog-scrollable">
<div class="modal-content">
<div class="modal-header">
<h5>Users who have access under {{req.
organization}}</h5>
<button type="button" class="btn btn-close"
data-bs-dismiss="modal"></button>
</div>
<div class="modal-body">
    {% for user in req.requesters %}
    <p>{{user}}</p>
    {%endfor%}
</div>
<div class="modal-footer">
<button type="button" class="btn btn-
secondary" data-bs-dismiss="modal"><i>
class="fa-solid fa-x"></i> Exit</button>
>
</div>
</div>
</div>
</div>
    {% endfor %}

</tbody>
</table>
</div>
{% endblock content %}

```

View Transactions

```

{% extends 'index.html' %}
{% block title %}Transaction View{% endblock title %}
{% block content %}

<div class = "container-xl pt-5 mx-auto">
<h1 class="text-center pb-4">Transaction View</h1>
    {% if transactions %}
    <form method="get" action="{% url 'view_trans' %}">

```

```

<div class="row mx-5 mx-5">
<div class="col">
<label for="patient_address">Patient Address:</
label>
<select class="form-select" name="patient_address"
aria-label="Default select example">
<option value = "None" selected disabled
hidden>-----</option>
    {%for patient in patients %}
    <option value="{{patient}}">{{patient}}</
option>
    {%endfor%}
</select>
</div>

<div class="col">
<label for="access_level">Access Level:</label>
<select class="form-select" name="access_level"
aria-label="Default select example">
<option value = "None" selected disabled
hidden>-----</option>
    {%for a in access %}
    <option value="{{a}}">{{a}}</option>
    {%endfor%}
</select>
</div>
</div>
<div class = "row-3 mx-3">
<button class="btn btn-success text-center mx-5 my-2"
type="submit">Filter</button>
</div>
</form>

<div class = "table-responsive mb-4">
<table id ='tx_table' class="table" >
<thead>
<tr>
<th scope="col" class = "text-center">Txid</th>
<th scope="col" class = "text-center">Patient
Address</th>
<th scope="col" class = "text-center">
Organization</th>
<th scope="col" class = "text-center">Data</th>
<th scope="col" class = "text-center">Access</th>
<th scope="col" class = "text-center">Timestamp</
th>
</tr>
</thead>
<tbody>
    {% for tx in transactions%}
    <tr>
<td class = "py-3 px-3 text-center">{{tx.txid}}</
td>
<td class = "py-3 px-3 text-center">{{tx.
patient_address}}</td>
<td class = "py-3 px-3 text-center">{{tx.org}}</
td>
<td class = "py-3 px-3 text-center">{{tx.dataaid
}}</td>
<td class = "py-3 px-3 text-center">{{tx.purpose
}}</td>
<td class = "py-3 px-3 text-center">{{tx.
access_level}}</td>
<td class = "py-3 px-3 text-center">{{tx.
timestamp}}</td>

    </tr>
    {% endfor %}
</tbody>
</table>
</div>
    {%else%}
<div class="row justify-content-end">
<div class="col">
<h3>No transactions to view</h3>
</div>
</div>
    {%endif%}
</div>
{% endblock content %}

```

XI. Acknowledgment

I would like to express my deepest gratitude to the Lord for His unwavering guidance and blessings throughout this journey.

I am incredibly thankful to my family for their endless love, support, and encouragement. Their patience and understanding during long hours of work and dedication to this project are deeply appreciated.

I would like to extend my heartfelt appreciation to my girlfriend for her unwavering support, understanding, and encouragement. Her love, patience, and belief in me have been a constant source of inspiration.

I am also grateful to my friends for keeping me sane during dark times, and helping me laugh and be positive when things seem to not work out.

Lastly, I would like to extend my appreciation to all the individuals, mentors, and colleagues who have provided guidance, assistance, and valuable insights during the development of this project. Your expertise and constructive feedback have immensely contributed to this project.