University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

A Blockchain-based Patient-centered Patient Generated Health Data Sharing Decentralized App

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

Julius Allen A. Reyes

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 "A Blockchain-based Patient-centered Patient Generated Health Data Sharing Decentralized App" prepared and submitted by Julius Allen A. Reyes 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, M.Sc. (*cand.*)	_____	_____
3. Perlita E. Gasmen, M.Sc. (*cand.*)	_____	_____
4. Ma. Sheila A. Magboo, Ph.D. (*cand.*)	_____	_____
5. Vincent Peter C. Magboo, M.D.	_____	_____
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.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences and Mathematics

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

Abstract

With the advancements in wearable technology, use of Patient Generated Health Data has been on the verge of making it to the normal. They provide a way for patients to generate their own health data, which gives them awareness on the state of their health, and communicate it to doctors for diagnosis. However, with traditional systems, sharing data that includes private and sensitive data can be unsafe and their data might be targeted by malicious entities. Thus, a need to develop a system where secure data sharing can be done is needed. Blockchain technology has been on a steady rise in the past few years, boasting security and performance advantages compared to traditional systems. To address these, this paper proposes a patient-centered blockchain-based system that takes advantage of the built-in security and privacy features of Blockchain.

Keywords: Patient-Centered, Patient Generated Health Data, Blockchain, Ethereum

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

With the advancements to healthcare and wireless technologies, smartphones, and wearable technology, and with how the world continues to go digital [1], Patient generated health data have been a thing of interest for healthcare institutions. Patient generated health data are basically health-data that are generated by the patients themselves [2] and is often used to recognize and diagnose the patient based from the vital data collected in their PGHD. These can include health and treatment history, biometric data from sensors, symptoms, and lifestyle choices. PGHD collection is mostly done through the use of home health equipment and/or mobile and wearable devices, often accompanied by an application on mobile devices [3].

Patients themselves are the ones who are responsible for capturing, recording, and/or collecting these data correctly. As a result, patients have full control on their own data and it is in their discretion on how they will share the data with their doctors [4]. From a research standpoint, it can be seen that PGHD of the greatest interest often include vital signs, stress levels, mood, physical activity, weight, diet, blood levels, medications, sleep patterns, tabacco and alcohol use, as well as environmental exposures.

As interest in PGHD grows, even tech giants such as Samsung and Apple have adopted the use of it [4]. These companies have since integrated PGHD in their devices and with it developed major digital health initiatives. One such example is Apple's "HealthKit" features, which is now integrated into their operating system, the iOS, as of iOS8. Apple also have partnerships with the Mayo Clinic and the EPIC electronic health record with the aim of further improving and developing digital health.

Blockchain is a decentralized and distributed ledger that handles and records

transactions between members on a ledger, often without a central authority governing over it. Blockchain technology, in recent years, have successfully penetrated different fields, mostly due to how popular cryptocurrencies are. Over the years, due to the need of a more patient-centered approach when it comes to healthcare systems, blockchain have been a topic of interest [5]. Medical data is very important in healthcare, these data are very sensitive, and often confidential, which makes it a prime target for malicious attacks. It has been more than ever important to secure all of these data. Here is where blockchain comes in, as blockchain is very robust against attacks and failures, and it also provides different methods of access control, making blockchain a good framework for healthcare data.

B. Statement of the Problem

According to a study done by Deering, et al. [6] the privacy and security related issues and concerns of the usage of Patient Generated Health Data can be narrowed down to the following: Authenticity of the patient, provenance of the data, tracking of the data from source going to providers and staff, patient authorization of the data, and the patient knowing the identity of providers and staff accessing the data.

C. Objectives of the Study

The objective of the study is to create a patient-centered patient generated health data sharing system with the following specifications:

1. The patient can:
    (a) Create an account on the system.
    (b) Log-in and access their account.
    (c) Upload their PGHD.

(d) Access their PGHD.

(e) Edit/Update their PGHD.

(f) Access their PGHD's history.

(g) Grant access/Revoke access to their PGHD to other people.

2. The doctor can:

(a) Create an account on the system.

(b) Log-in and access their account.

(c) Request for access to the PGHD of their patients.

(d) Access the PGHD of their patients.

(e) Access the PGHD history of their patients

D. Significance of the Project

Secure usage of digital data can be tricky, especially in healthcare scenarios, where data is sensitive and confidential. One thing that prevents healthcare institutions from developing health informatics are the patients' concerns with regards to privacy and security. Blockchain-based systems, on the other hand, by nature, include security properties and features that essentially makes it so that data stored in the blockchain are impervious to tampering. Adoption of blockchain into healthcare systems may prove to be the answer to the privacy and security concerns of these systems, allowing widespread usage of digital data in healthcare systems without the drawbacks in security and privacy.

E. Scope and Limitations

1. The patient generated health data only includes biometric data, symptoms, and lifestyle choices and health and treatment history.

2. The system only focuses on the privacy and security between the sharing of patient generated health data.

3. Communication between the patient and the doctors, consultations, and/or check-ups are not handled by the system and is to be done outside the system.

4. Other documents other than patient generated health data such as x-rays, prescriptions, etc. are not supported by the system.

F. Assumptions

1. Doctors registered in the system are real doctors.

2. The study assumes that the patient inputs true and correct data.

3. Both the patients and the doctors are in healthcare institutions that use the system.

4. Only patient generated health data and no other documents are stored into the blockchain network.

5. Due to the lack of accessible Wearable API, PGHD is to be inputted to the system manually.

II. Review of Related Literature

A. Electronic Health Records

Electronic Health Records, or EHRs in short, is simply a comprehensive, agglomeration of a patient's health data, as well as past healthcare data, albeit electronically [7]. It not only contains data that are relevant to the patient's health, but also to that certain patient's treatment as well. Due to the electronic nature of EHRs, it can be made easily and readily available to other people [8]. EHRs are also much more effective than just being a medium of information. According to Carey, et al. [9] EHRs have more benefits such as lower costs, provides an advantage in record keeping, ensures the mobility of the records, as well as having an impact in improving healthcare quality.

This technology, however, does not come without concerns. Several studies have shown that the use of EHRs can raise privacy, security, and even ethical issues. Keshta, et al. [10] found out through certain studies that when asked whether they feel safe and secure with their health data, close to half of the respondents stated that they were worried about the privacy and security of their data, while the remaining half were either satisfied with the security or had no care at all about the subject. There have also been ethical concerns with the use of EHRs. Time-saving measures, such as copy-and-pasting of previous entries into the current EHRs can lead into useless and repetitive entries or even inaccuracies, as well as bloat in information [11]. Due to EHRs being mostly computer-assisted, physicians can lose focus and blindly follow the information as they appear on the computer, without actually catering to the patient's current demands or needs.

B. Patient Generated Health Data

Patient Generated Health Data, or PGHD in short, are basically health-related data that is generated and gathered from the patient themselves, that is often used to help address a health concern [2]. These PGHDs include, but are not limited to, health and treatment histories, patient-reported outcomes, and biometric sensor data. This has become possible with the recent advancements in the field of wireless technology, smartphone, and IoTs.

Usage of PGHD have resulted in a lot of benefits for patients using them [12]. Having the patients generate their health data themselves gives the patient a lot more awareness on the state of their health. This awareness more often that not can result into the patient being more engaged in taking care of their health. PGHDs also prove to be a game changer in improving health outcomes of patients.

PGHDs, however, does not come without its problems [12]. Due to PGHDs being a relatively new technology, not a lot use them, be it doctors or patients. This results into a limited widespread use and acceptance among doctors and patients. Technical issues also come into play, as not everyone has a means to connect to the internet, a major requirement for PGHDs, as well as the lack of applications that supports PGHD in other languages, which might pose a language barrier problem to some patients. As with all things digital, patients also reported privacy and confidentiality concerns on their collected data on how it is being shared and basically just where their data goes through.

C. Security in Traditional Health Data Sharing Systems

One particular approach in applying security into EHR systems is through the use of cryptography. In 2013, the US Department of Health and Human Services released the Omnibus Final Rule [13]. In it, significant modifications were made to the privacy and security standards under the Health Insurance Portability and Accountability

Act, or HIPAA for short. These new standards and regulations were motivated by the need to ensure the security, confidentiality, integrity, and privacy of patients' data in electronic health records, as well as in other data formats.

The HIPAA designated ways in which health information can be secured through the use of cryptography. The use of encryption has proven to improve the security of electronic health records, especially when electronic health records are shared between a patient and their doctor. This exchange has some policies which it needs to adhere to, as well as being needed to be recorded [13]. Security breach concerns has also been tackled and prevented with the use of digital signatures. Digital signatures are applied to electronic health records as another layer of security and effectively prevents breaches of security.

Other approaches to security in EHR systems consists of using more traditional techniques, such as usage of antivirus software, cloud computing, preliminary risk assessment, use of RFIDs, [13] and even the usage of Firefox, as Firefox have been found out to be very effective in securing the network of an organization, as well as being able to ensure that health information is protected on the network [14]. Firefox has been used widely to provide security and protection to information technology systems of healthcare organizations.

D. Blockchain Technology

The term "blockchain" has more often than not, a lot of different definitions. One widely used definition for blockchain is that of a tamper resistant digital ledger implemented in a distributed network, often referred to as a "distributed ledger", and usually does not have a central governing authority in the system [15]. Due to the distributed nature of blockchain, each member of the blockchain network has a copy of the ledger. Blockchain enables a group of users to records transactions in a distributed ledger within that community. This makes it so that no transaction can be

modified or tampered with once it is published.

In 2008, a user by the name of Satoshi Nakamoto posted about Bitcoin in a cryptography mailing list. The idea of blockchain was combined with other technologies and concepts to create cryptocurrencies, with Bitcoin being the first recorded blockchain-based cryptocurrency in existence [15]. In 2009, the first functional version of Bitcoin was announced. Main characteristics of Bitcoin include faster transaction speeds, low cost, and anonymity. Bitcoin can be acquired through different means, such as through bitcoin mining, purchasing bitcoin, and trading other things for them. Being a product of blockchain, bitcoin is considered to be the world's first decentralized currency, where the currency is not issued by a central authority (banks) and is not governed by government rules. Nowadays, Bitcoin have been widely accepted as a form of payment by a lot of companies [16].

After Bitcoin, Ethereum was first introduced in a paper by Buterin, with the goal of addressing the limitations of Bitcoin. Ethereum presents a blockchain network with a Turing-complete language. It allows anyone to create their own set of rules for ownership, and transactions. This is done through the utilization of smart contracts, a set of rules that are only executed if certain criteria are met [17].

Potential fields of uses for Ethereum include token systems, insurance, identity and reputation systems, cloud computing, etc. The most important use of Ethereum, however, is in decentralized applications, or dApps for short. Many dApps are prevalent today such as Golem, Augur, and Civic, to name a few [10].

A decentralized application, simply, is a type of distributed open-source software application that runs on a peer-to-peer blockchain network, instead of a single computer. dApps are free from the control of a central governing authority. Nowadays, dApps can be a variety of applications, ranging from finance, to gaming, and to social media.

Throughout the years, blockchain has been applied to several fields, mostly due to

its properties [18]. Its immutable property ensures that the data within the blockchain network are irreversible and cannot be modified or tampered with once it is uploaded to the blockchain. This ensures the integrity and authenticity of the data. Its distributed nature means that every member of the network has their own copy of the ledger. This is important because in a distributed ledger, every member will have to participate in the changes within the network. This is important for its decentralized nature, as in a blockchain network, there is no centralized governing authority. This promotes transparency in the network, meaning every change in the network can be traced, this also eliminates a single point of failure in the network. This also leads into the consensus mechanism in blockchain. This simply means that for something to happen in the network, a consensus must first be reached. A certain number of members (or everyone) must first approve that something before it can happen. All in all, several, if not all of these properties are taken advantage of in the application of blockchain technology in healthcare systems.

E. Applications of Blockchain Technology in Health Data Sharing Systems

With the advancements on blockchain technology throughout the years, several studies now aim to address the security and privacy concerns that comes with dealing with health data. With this, several studies have come up with proposed systems using different blockchain platforms and techniques with the aim of combating the security and privacy concerns of traditional health data sharing systems.

In order to combat the privacy and security concerns that come with traditional EHR sharing systems, Tanwar, et. Al [19] proposes a hyperledger-based electronic health records sharing system that uses blockchain in order address security and privacy concerns that comes with EHR sharing. The study also includes a proposed Access Control Policy Algorithm to aid in data accessibility in between healthcare

providers, as well as to address the concerns of the patients when it comes to giving doctors access to their EHR. The Access Control Policy Algorithm makes it so that the doctors and/or the laboratories have to ask for access to the patient when they need to access their EHR, the patient can now then allow access to their EHR, and revoke access when their consultation period is done. The admin also has control on the members where they can remove members from the network when they are to do something malicious.

HealthChain is an EHR system built upon consortium blockchain technology made up of hospitals, insurance providers, and government agencies, with the aim of overcoming the disadvantages that comes with traditional EHR systems [20]. An instance of chaincode will regulate the permissions of the members, and a designated orderer establishes consensus which uses Proof of Authority as consensus protocol on transactions of EHRs and then disseminate blocks to peers. The HealthChain succeeds in creating a system where EHRs can be stored and shared to members, while also having features such as privacy preservation, improved security, and high throughput.

Shuaib, et al. [21] proposes a permissioned hyperledger-based healthcare data sharing system that integrates Blockchain technology, decentralized file system and threshold signature to address problems such as single point of failure and security problems which stems from the use of traditional database systems. The decentralized file system provides much better security than most traditional centralized file systems while still being able to provide the same level of performance, if not more. The system uses IBFT as a consensus protocol which is the basis of the threshold signature, while the decentralized file system is based on InterPlanetary File System (IPFS).

Niu, et al. [22] proposes a sharing scheme for electronic health records based on permission blockchain technology for the purpose of eliminating violations of patient privacy. The permission system consists of 4 different types of users namely: patient,

doctor, cloud server, and searcher, each with their own set of permissions and access. The encrypted electronic health records also are able to be searched in the system with the use of keywords and ciphertext.

Hashim, et al. [23] proposes yet another approach for a blockchain-based electronic health record sharing system, the MedShard. MedShard is a blockchain-based electronic health records sharing system which aims to address the scalability issues that comes with most blockchain systems due to the consensus algorithm and ledger replication using blockchain sharding. It uses transaction-based sharding that uses Proof of Authority for consensus, with the previously visited caregivers of the patient as the validating nodes. This makes it so that when a patient visits a caregiver, the caregiver creates a digital signature, which it then broadcast to registered nodes in the network, caregivers that were previously visited by the patient will verify the node, and then respond by searching through their local database for any previous records, while still adhering to any signed consent by the patient. PoA consensus is then used to reach an agreement on the validity of the shard, and once a transaction has been reached, the shard is discarded.

Dubovitskaya, et al. [24] proposes ACTION-EHR. ACTION-EHR is a permission blockchain-based system for EHR data sharing and integration. The system is built upon the Hyperledger fabric and data sharing transactions are implemented using chaincode. Each member hospital will have to provide a blockchain node integrated with its own EHR system that will form the blockchain network. A web-based interface is also used to allow patients and doctors to have EHR sharing transactions. The actual EHR data is stored and encrypted in a HIPAA-compliant cloud-based storage. The system uses PKI-based asymmetric encryption as well as digital signatures to ensure the security of shared EHR data.

Ktari, et al. [25] proposes an IoMT-based Platform for E-health Monitoring based on Blockchain. The main objective of the system is to be able to directly and securely

share a patient's PGHD directly from their wearables, smartwatches, or smartphones to their doctors, hospitals, pharmacy, or even insurance companies. The system makes use of blockchain as a way to securely store the data gathered from the patients, as well as for encryption of the data about to be shared. The system also makes it so that only the patient and their attending doctor or physician have the rights to access the patient's data. The system makes use of Raspberry Pi 3 to facilitate the data transfer from a mobile app that collected the data from a patient's wearables, which is then sent to the blockchain system for storage and sharing.

F. Synthesis

With the rise of wireless technology and the Internet of Things, patient generated health data continues to be more and more used nowadays. With digital data however, comes the question of privacy and confidentiality. Several studies have utilized blockchain technology in order to combat these concerns. Blockchain was mostly used for its privacy-preserving and security capabilities, ensuring that the patient has the right and complete control of their data. Blockchain was also utilized for performance improvements. Several proposals each discussed their own approaches to combat the concerns that comes with the usage of PGHD. All in all, blockchain, with its improvements and advancements throughout the years, prove to be a big aid in addressing the security and privacy shortcoming of healthcare technology.

III. Theoretical Framework

A. Patient Generated Health Data

Patient Generated Health Data, or PGHD in short, is commonly defined as health data generated by and from the patient themselves. Nowadays, the vast majority of PGHD are generated through the use of mobile health applications or mHealth apps in short, with the use of wearables such as smartwatches such as Fitbit, or wearable medical devices such as a continuous glucose monitoring device [12].

Patient Generated Health Data contains, but is not limited to data about a patient's health and treatment history, symptoms, biometric data, lifestyle choices, and other clinically relevant information to assist in addressing health concerns.

Patient Generated Health Data used in the system include the age, height, weight, blood pressure (systolic and diastolic), blood sugar, symptoms, and diet of the patient.

A..1 Mobile Health Applications

Mobile Health Applications or mHealth apps, are a type of mobile applications that are related to health knowledge and research. They are often used by health care professionals as well as patients in order to improve health treatments and public health [26]. These applications use the data collected by certain sensors, such as those from wearable devices, to handle the identification of various health parameters related to a patient's health, such as physical activity, and health state. It is also possible to recognize vital parameters of a patient's health through mobile sensors, allowing the application to recognize a patient's daily activities and lifestyle. The applications are ready to use in order to help patients to regulate or prevent some chronic diseases, such as obesity, diabetes, and cardiovascular accidents as well as help them with the control of health-related habits such as diet, exercise, sleep, smoking cessation, relaxation, and medication adherence. All in all, mHealth apps have several

purposes, such as diagnosis of heart rate problems, and controlling a glucose meter used by diabetic patients. These applications provide easy access to information related to health conditions or treatment; describe, show, or communicate potential medical conditions to their health care provider.

A..2 Wearables

Wearables as medical technologies are becoming more and more an integral part of personal analytics, being able to measure physical status, physiological parameters. They do not only promise to help people pursue a healthier life style, but also provide continuous medical data that can be used to actively track metabolic status, diagnosis, and treatment [27]. Advances in the miniaturization of flexible electronics, biosensors, microfluidics, and AI algorithms have led to wearable devices that can generate real-time medical data within the Internet of things. Wearables have already been impacting health care and medicine by enabling health monitoring outside of health care institutions and prediction of health events [28].

Various kinds of wearables have now been developed for various parts of our bodies. Data are collected from these wearables to the Internet or mobile devices through Bluetooth, WiFi, LTE, 3G, 4G, or 5G connections. The medical data can be sent to a healthcare provider to receive therapeutic feedback or acted upon automatically by other devices in the network [27].

A popular type of wearables are smartwatches. Smartwatches are basically an extension of smartphones, having basically the same capabilities as them, albeit limited. Smartwatches nowadays are fitted with sensors that can collect health data from the user. This feature is usually accompanied by mHealth apps in the user's smartphones which can display the collected data, and even send them to a doctor or health care professionals in real time.

B. Blockchain

Blockchain, simply is a decentralized, shared, immutable data structure in a network [29]. Each and everyone in the network shares the blockchain and has their own copy of it. The blockchain assist in handling the process of recording transactions and assets in the network. An asset can be virtually anything, be it a tangible object such as material things, or an intangible object, such as intellectual properties or rights to a property. Being of immutable property, blockchain essentially makes it difficult to modify, change, or tamper the system, making it a lot more secure. Every recorded transaction in the blockchain is recorded as a block. This block contains all the details of the transaction.

B..1 Blockchain Transaction Process

The process of a blockchain handling transactions include steps such as [30]

1. The transaction is recorded, containing digital signatures from each party involved in the transaction and their relevant details.

2. The members of the network are notified of the transaction and examines it. The members will examine the transaction to make sure that it is a valid transaction, using different algorithms. This process occurs among all the members of the network.

3. Once a transaction is verified and accepted, the transaction will now be recorded into a block. The block contains its own unique hash, as well as the hash of the block before it, allowing the members of the network to identify where the block is located in the blockchain.

4. After the block is created and completed, the block is now added to the blockchain. The unique hash makes sure that the blocks are placed in their proper chronological order.

Afterwards, the transaction is now deemed as completed. Each member of the ledger will now have a copy of the block.

B..2 Blockchain Architecture

Blockchain, from its name, is simply a chain of blocks, which contains a complete record of transactions, which each block holding a number of transactions [31]. The blockchain consists of five layers, namely [32]:

1. **Hardware layer:** consists of hardware, i.e. computers in the network, network connections, data servers.

2. **Data layer:** made up of blocks containing information, each block is connected to the previous block. The genesis block, or the first block in the network, is not connected to the previous block, since it is the first block. This is the layer where data in the network is managed.

3. **Network layer:** handles the communication between the different members of the blockchain network. Also referred to as the propagation layer, due to this layer is where blocks are created and is added into the blockchain.

4. **Consensus layer:** a single member cannot add a transaction into the blockchain, as all members of the network need to validate and accept it first. This layer lowers the risk of fraudulent and/or malicious transactions from being recorded and added into the blockchain.

5. **Application layer:** allows the use of blockchain for a variety of purposes or applications. Allows users to interact with the blockchain network.

A block in the blockchain consists of two major parts: the block header, and the block body. The block header consists of [31]:

1. **Block version:** indicates which set of block validation rules to follow.

2. **Merkle tree root hash:** hash value of the transactions contained in the block.

3. **Timestamp:** the current time listed as seconds in the universal time, originating from January 1, 1970.

4. **nBits:** target threshold of a valid block hash.

5. **Nonce:** a 4-byte field, usually starts at 0 and increases for every hash calculation.

6. **Parent block hash:** 256-bit hash value that points to the previous block in the blockchain.

For the block body, it consists of two parts, namely [31]:

1. **Transaction Counter:** a number that represents the number of transactions stored in the block.

2. **Transactions:** the transactions themselves.

The maximum number of transactions that can be stored in a block depends on the size of the block, as well as the size of the transactions themselves.

B..3 Consensus Mechanisms

Blockchain uses consensus mechanisms as a way to identify which members of the network are allowed to added a block into the blockchain. Different types of consensus mechanisms use in blockchain are the following [33]:

1. **Proof of Work:** members are required to solve and give a solution for a complex cryptographic hash problem. Here, miners compete with each other to solve the problem first and get a reward. This consensus mechanism requires a lot of computing power and uses up a lot of resources.

2. **Proof of Stake:** the validator that validates the next block is chosen based on the number of stake they have in the network. The chance to become a validator is directly proportional to their stake.

3. **Proof of Activity:** a combination of the PoS and PoW algorithms. It starts with the PoW algorithm and once the block is almost mined, it switches to the PoS algorithm.

4. **Proof of Elapsed Time:** generates a random wait time for each node, where each node must sleep for that duration. The node with the shortest wait time wins the block, allowing it to add a new block to the blockchain.

5. **Delegated Proof of Stake:** an evolution of the Proof of Stake algorithm. Allows stockholders to vote those who want to add a block to the blockchain.

B..4 Characteristics of Blockchain

The blockchain architecture have the following characteristics [34]:

1. **Decentralization:** transactions in the blockchain network are validated by most of the members of the network, eliminating the need for a central governing authority.

2. **Immutability:** each block in the network are linked to the previous block, meaning that any attempt to modify or tamper a block will affect all succeeding blocks before being able to modify that certain block, making it hard for an attacker to do malicious attacks.

3. **Transparency:** changes in the network are publicly available and visible to ensure transparency and security within the members of the network.

4. **Traceability:** since all of the information about a transaction is easily available, each transaction is easily traceable.

B..5 Types of Blockchain Networks

There are three major types of blockchain networks [35]:

1. **Permissionless blockchain:** also know as a public blockchain, of which bitcoin is the best example of. Virtually anyone can access and use it. Anyone can write and do transactions as long as they are following the rules of the blockchain. This type of blockchain powers up most of the popular digital currency in the market right now.

2. **Permissioned blockchain:** also known as a private blockchain. These kinds of blockchain acts as such of a private ecosystem. Users cannot easily join the network or do transactions, until they receive some sort of permission to be able to do these things. More often than not, these kind of blockchain belongs to an individual or an organization, where they are the central governing authority in charge of the permissions.

3. **Consortium blockchain:** also known as a federal blockchain. Instead of giving power to a single person or entity, it is instead given to a group of people or organizations which forms a consortium or a federation. Main examples are Qourum, Hyperledger, and Corda.

C. Ethereum

Ethereum was first mentioned and describe by a cryptocurrency researcher named Vitalik Buterin in a proposal in 2013. The proposal suggested the addition of a scripting language for programming Bitcoin. The development of Ethereum was funded in a crowdfunding of cryptocurrency tokens. In July 30, 2015, the Etherium first came online [36].

Ethereum is an open-source, distributed ledger based on blockchain technology. Etherium has its own cryptocurrency named Ether and its own programming language

named Solidity. Ethereum enables developers to develop and build decentralized applications or dApps in short. Miners mine Ether tokens which serves as a currency to be used to pay for usage on the Ethereum network. Ethereum also supports the use of smart contracts, which are a type of digital contract. Ethereum is Bitcoin's main competitor.

C..1 Smart Contracts

Smart contracts were first proposed in the 1990s by Nick Szabo [37]. Smart contract clauses will be automatically be executed once predefined conditions are met. Smart contracts containing transactions are essentially stored, replicated, and updated in distributed blockchain networks. Its main difference from conventional contracts is its automation, resulting in much shorter execution time and lower costs.

Other advantages that smart contracts have when compared to conventional contracts include:

1. **Reducing risks:** stemming from the immutability property of blockchain, smart contracts cannot be altered once they are issued.

2. **Cutting down administration and service costs:** due to blockchains being decentralized by nature, it eliminates the need of an administration to verify smart contracts, as they can be automatically triggered once their conditions are met.

3. **Improving the efficiency of business procedures:** again, due to smart contracts' automation, turnaround time can be significantly reduced, resulting into more efficient business processes.

Smart contracts in the Ethereum platform are usually written in Turing-complete languages such as Solidity, Serpent, Low-level Lisp-like Language, and Mutan. Ethereum compiles smart contracts using these languages into machine code, which can then be

loaded into the Ethereum Virtual Machine and run. Ethereum uses an account-based model, where each member is identified by their digital wallet. Ethereum also adopts the use of PoW as its consensus mechanism.

C..2 Decentralized Applications

Decentralized Applications are a new type of application, that is decentralized in nature [38]. dApps interact with smart contracts by transactions, and provide services around them. In theory, a single smart contract can be considered as a dApp itself. dApps can be characterized by four properties as follows [39]:

1. **Open Source:** due to the nature of blockchain, dApps need to keep their source codes open source.

2. **Internal Cryptocurrency Support:** as cryptocurrency are required in a blockchain in order to be able to do transactions, dApps also uses them to quantify all credits and transactions among members of the system.

3. **Decentralized Consensus:** consensus is needed among the members to promote transparency.

4. **No Central Point of Failure:** A fully decentralized system should have no central point of failure due to all components of the application is hosted and executed in the blockchain.

IV. Design and Implementation

A. Use Cases

The system has two users: the patient and the doctor.

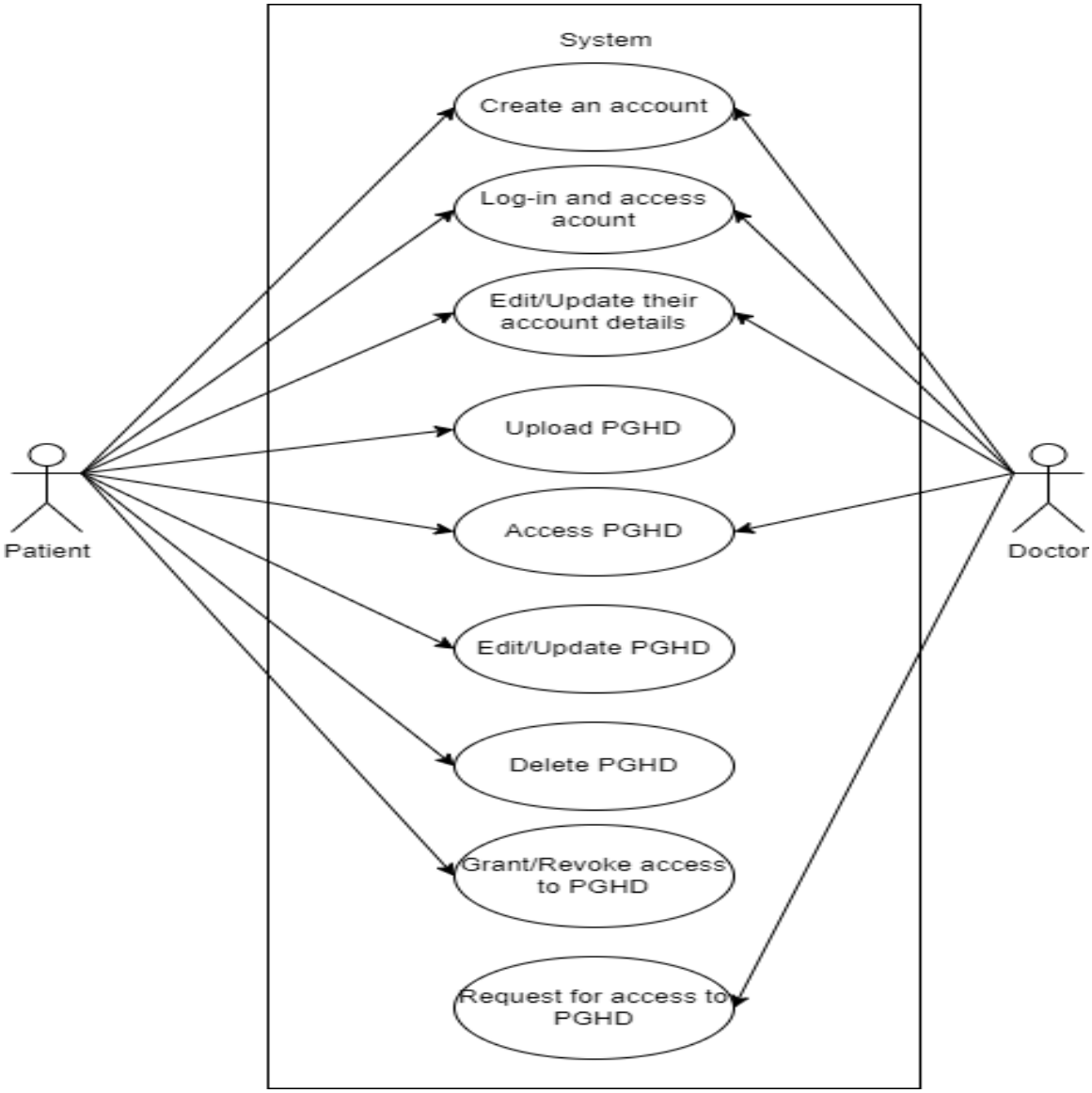

Figure 1: Use Case Diagram

The patient has full ownership of their data. Both the patient and the doctor can create an account in the system and access it. They can also update their account

details.

The patient can upload and access their PGHD into the system. The patient can also update it. The patient also has control on who can access their PGHD, allowing them to grant/revoke a doctor's access to their PGHD.

The doctor can view all of the patient's PGHD they have access to. If they want to access a certain patient's PGHD, they need to request for access first from the patient.

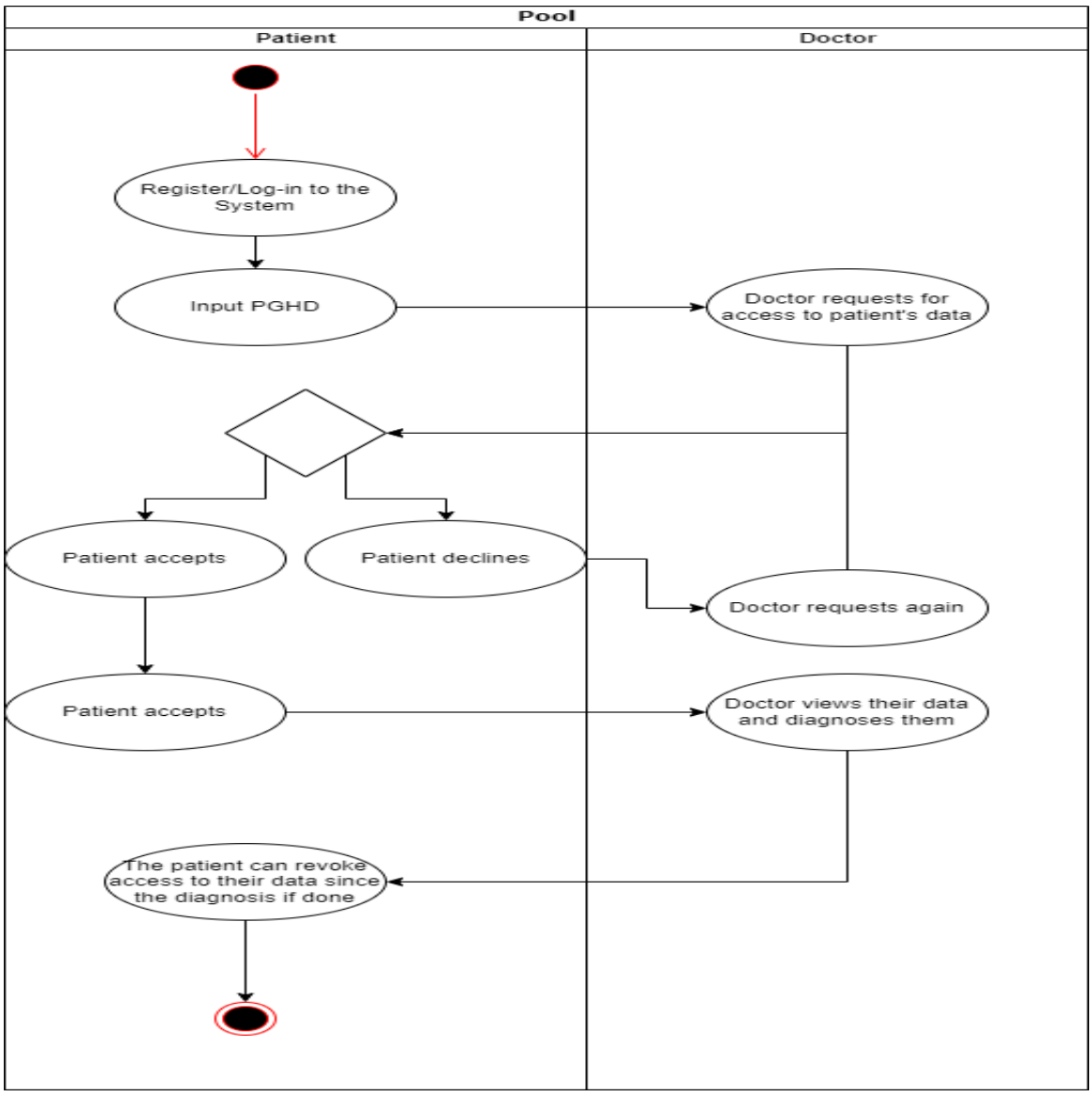

Figure 2: Activity Diagram

B. Database Design

Since the data inputted by the patients and viewed by the doctors are relatively small, the system opted to use on-chain blockchain database. The on-chain database is where patients can input their details and generated data, and update them. The doctors can also read and view the details of the patients that they have access to. Data was stored on-chain to solve interoperability issues.

The on-chain database is also where the smart contracts for access requests are stored.

The data structure for patient generated health data can be seen in the table below:

Field name	Data Type	Description
name	string	Patient's name
age	uint256	Patient's age
height	uint256	Patient's height (in kg)
weight	uint256	Patient's weight (in cm)
systolic	uint256	Patient's Systolic Blood Pressure (in mmHg)
diastolic	uint256	Patient's Diastolic Blood Pressure (in mmHg)
bloodsugar	uint256	Patient's Blood Sugar Level (in mg/dL)
symptoms	string	Symptoms the patient are experiencing
diet	string	Current diet of the patient
timestamp	uint256	Unix timestamp when the data was created

Table 1: Data Structure for Patient Generated Health Data

The data structure for access requests can be seen in the table below:

Field name	Data Type	Description
doctor	address	Doctor's address
patient	address	Patient's address
granted	bool	True if access is granted, False otherwise

Table 2: Data Structure for Access Requests

C. System Architecture

The system is a decentralized application built upon the Etherium blockchain network. Solidity was used as the programming language to write the smart contracts. Ganache was used to deploy a local blockchain network and Remix was used to test and deploy the smart contracts alongside Ganache. Web3.py was used for the system to interact with the Ethereum blockchain.

To build the web app, Django was used for the backend, while Bootstrap was used for the frontend.

D. Technical Architecture

To sync with the Ethereum blockchain, the following minimum requirements must be met by the computer:

- CPU with at least 2 cores
- 4GB RAM minimum with an SSD, at least 8GB RAM minimum with an HDD
- 8 Mbit/s bandwidth

The recommended requirements on the other hand are as follows:

- CPU with 4+ cores
- 16GB RAM
- SSD with at least 500GB space
- 25+ MBit/s bandwidth

V. Results

A. User Registration and Authentication

Once the user accesses the web app, they will be greeted with the log-in page. The log-in page works for both patients and doctors. In order to log-in to the system, the user simply has to provide their registered email address as well as the password to their account.

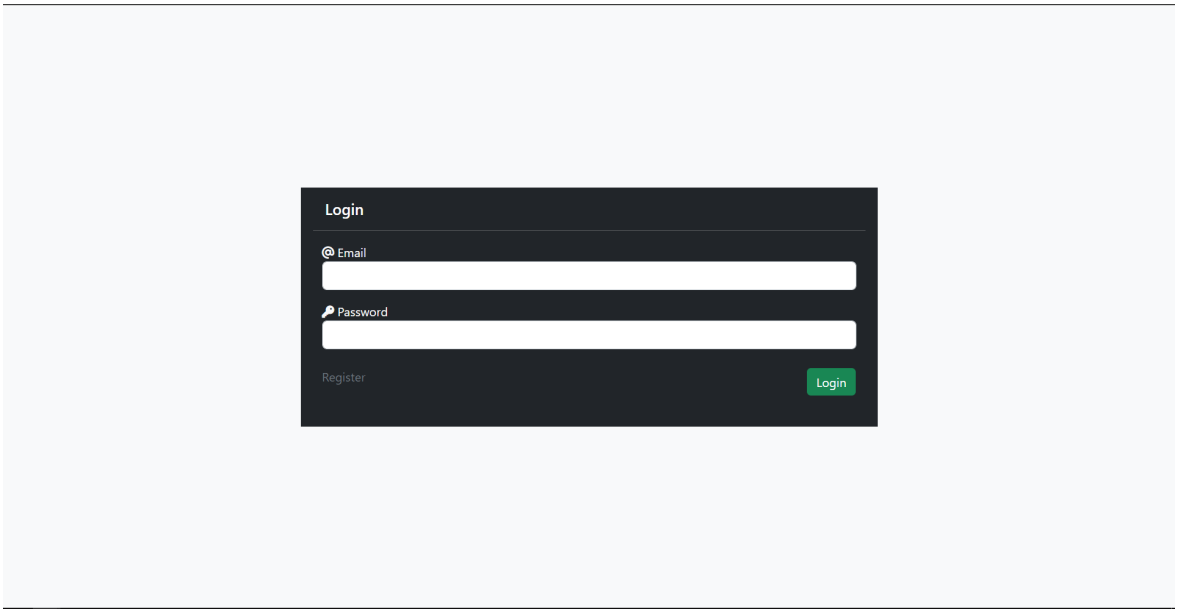

Figure 3: Log-in page

If the user does not have an account yet, they can instead proceed to the registration page. In the registration page, the user can input their first and last names, username, email address, ethereum address, and password. The user can also determine whether they are a patient or a doctor. Once the user submits their registration, their account is created and can proceed to logging in.

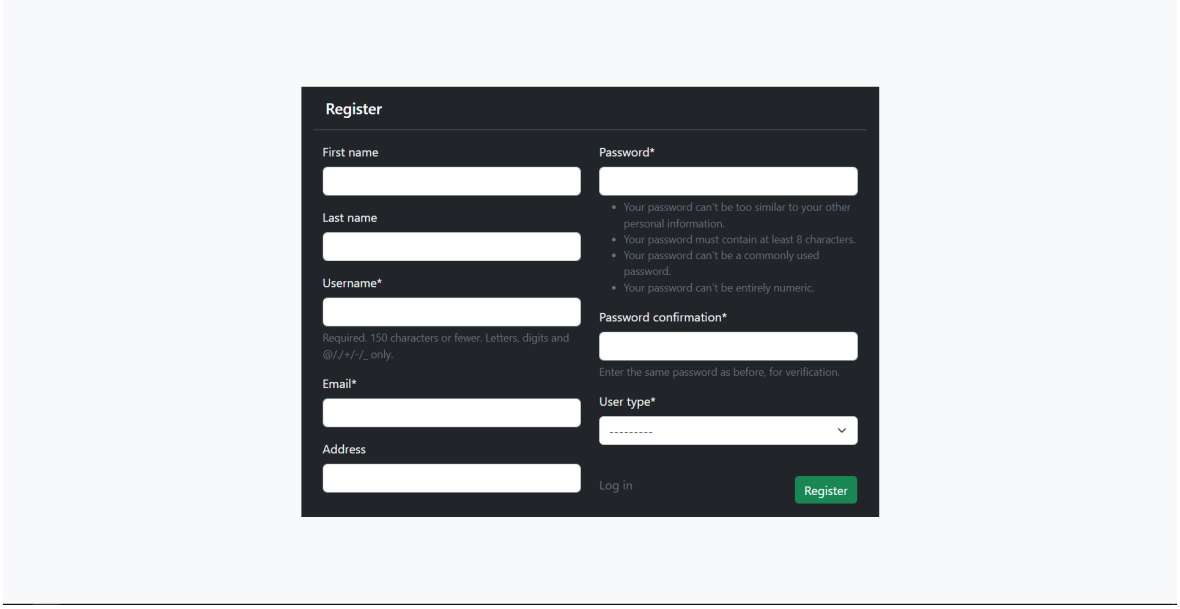

Figure 4: Registration page

B. Patients

B..1 Patient Data

Once a patient has logged in, they will be greeted with their dashboard. In their dashboard, they will be able to see their personal information, current health data, and the doctors with requests to their data.

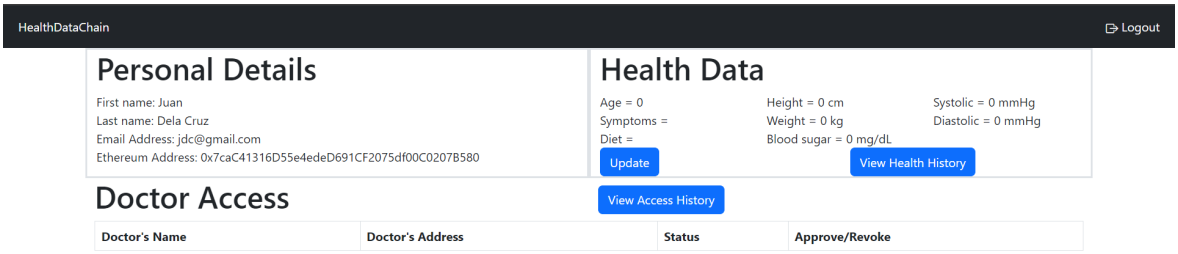

Figure 5: Patient Dashboard

Should the patient wish to update their health data, they can click the Update button. After clicking, a modal with a form will appear, in the form, the patient can input their health data.

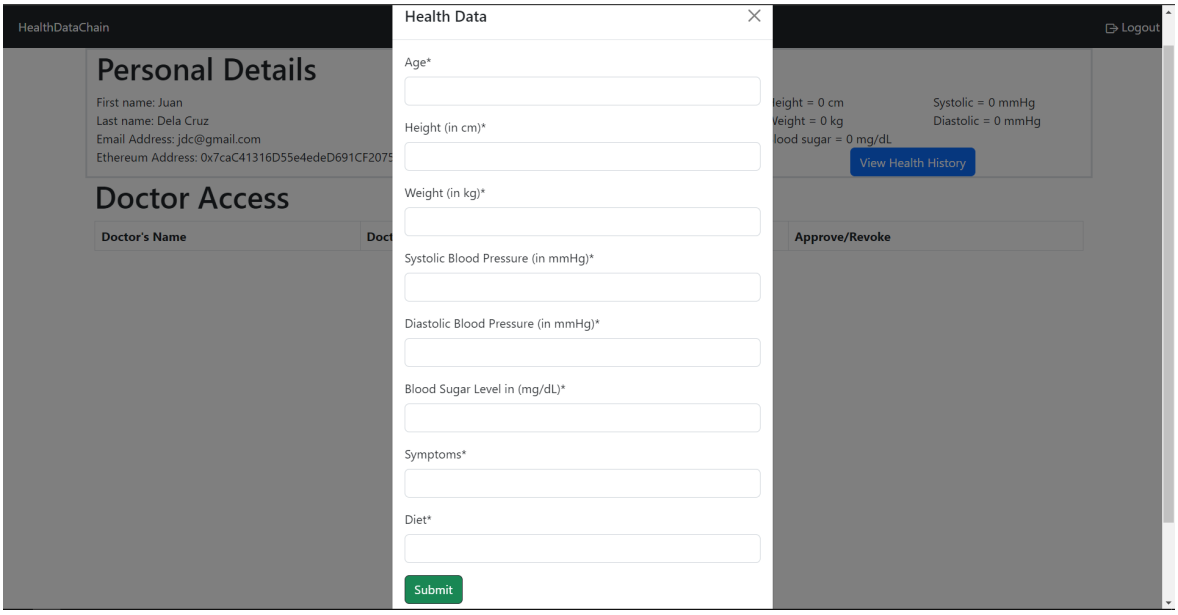

Figure 6: Patient Data Form

After submitting their data, the patient will be redirected back to their dashboard and their health data will show the updated values. A timestamp will automatically be added based on the time the patient submitted their data.

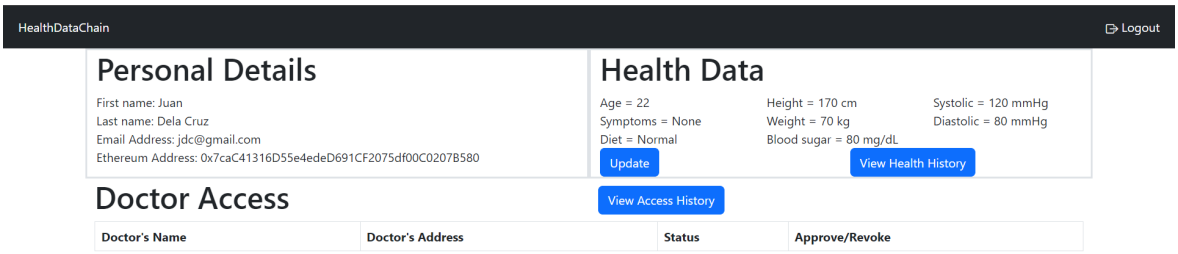

Figure 7: Patient Data Updated

When the patient already has existing health data and updates it, the old data will be stored as part of the patient's data history. Old data are kept in order to see the trend in a patient's health. The old data are stored such that the latest data are shown first. In order to access the patient's data history, the patient can click on the View Health History button

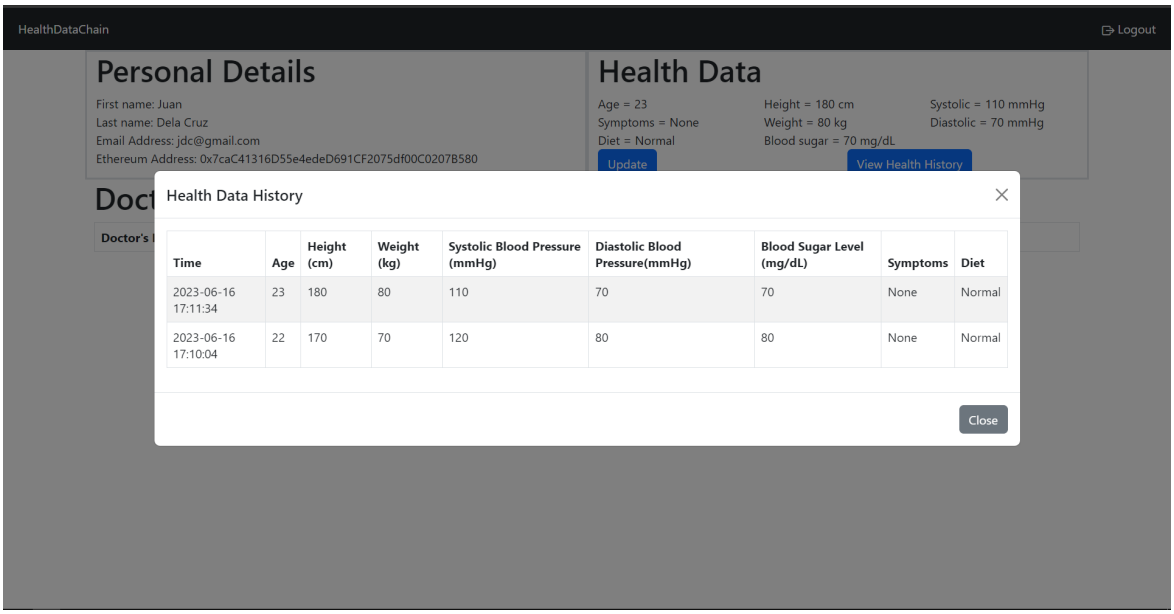

Figure 8: Patient Data History

B..2 Access Requests

Once a doctor has requested for access to a patient's data, the doctor's details will show up in the Doctor Access section of the dashboard. In here, the patient can see the doctor's name as well as their ethereum address, the request's status, and a button to approve/revoke access to the doctor. Access request's status will be pending by default.

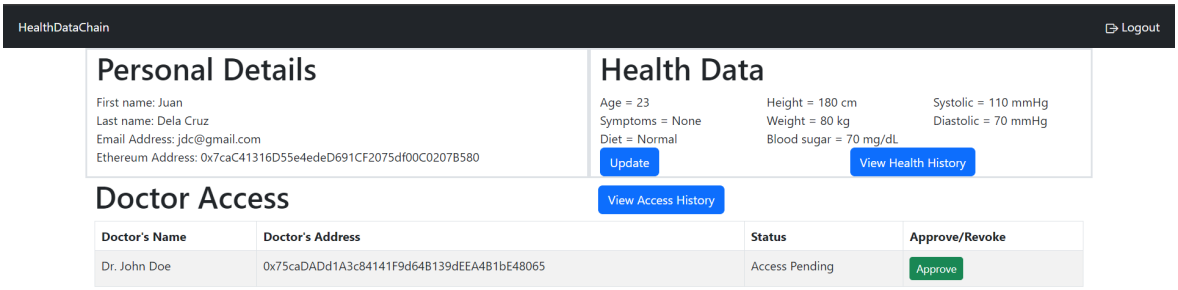

Figure 9: A Doctor's Access Request to the Patient's Data

Should the patient want to give the doctor access to their data, the patient can simply click the Approve button. The access request's status will be changed to granted, and the Approve button will change into a Revoke button. The doctor can now access their data and its history.

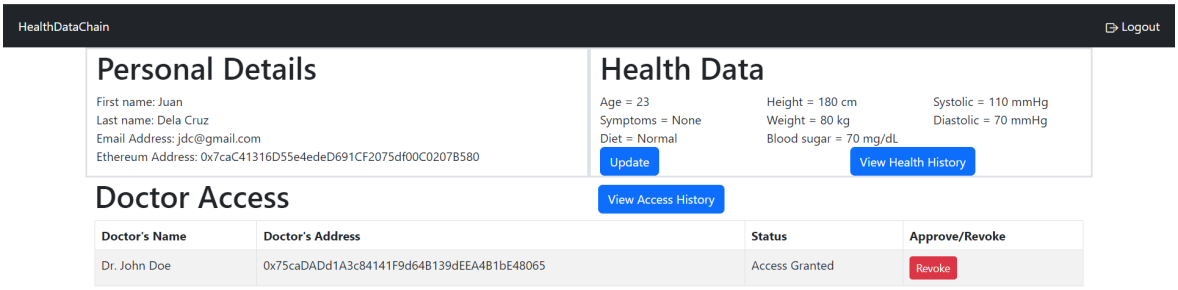

Figure 10: Access Request Approved

Once consultation with the doctor ends, the patient can now then revoke the doctor's access to their data, they can achieve this simply by clicking the Revoke button.

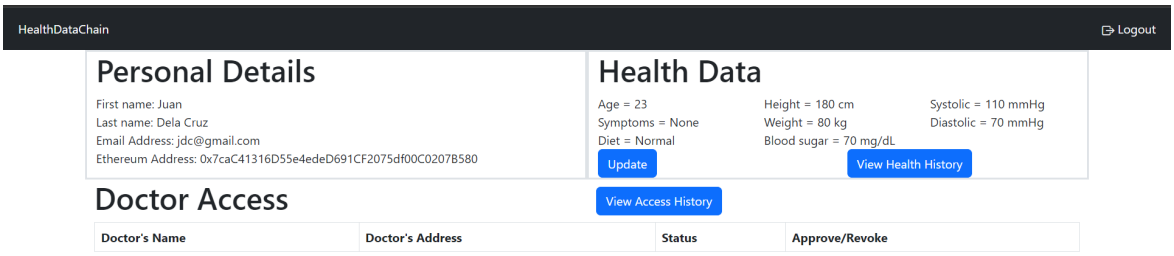

Figure 11: Access Request Revoked

Once an access request is revoked, the access requests will be stored as part of the patient's access request history. The access request history are kept for provenance purposes. In order to access the access request history, the patient can simply click on the View Access History button. Once clicked, a modal will appear with a table of previous access requests along with the requesting doctor's name and ethereum address.

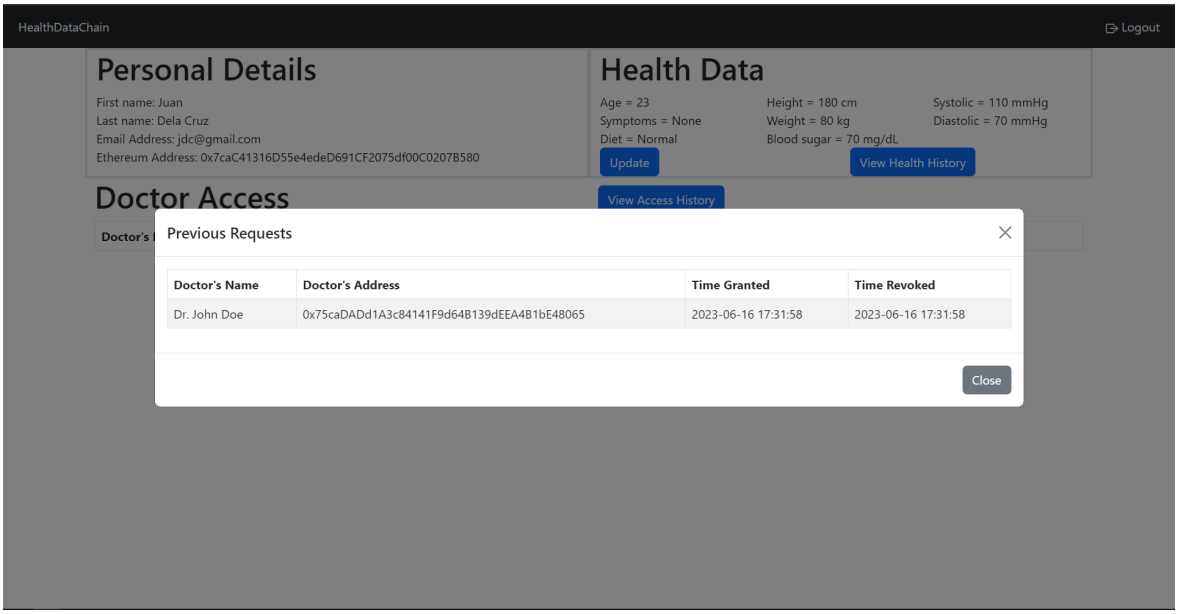

Figure 12: Access Request History

C. Doctors

Once a patient has logged in, they will be greeted with their dashboard. In their dashboard, they will be able to see their personal information, the access request form, and the patients they have access requests to.

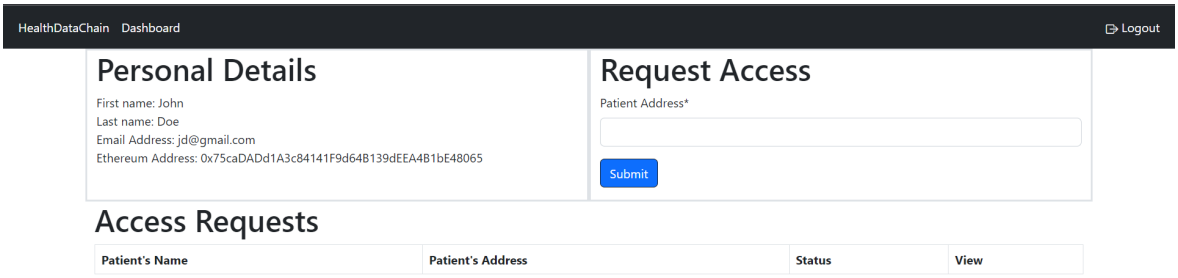

Figure 13: Doctor Dashboard

If the doctor wants to request access to one of their patients, they can simply input the patient's ethereum address to the access request form. The reason for using ethereum addresses for requesting is due to it being unique, and patients can have the same names.

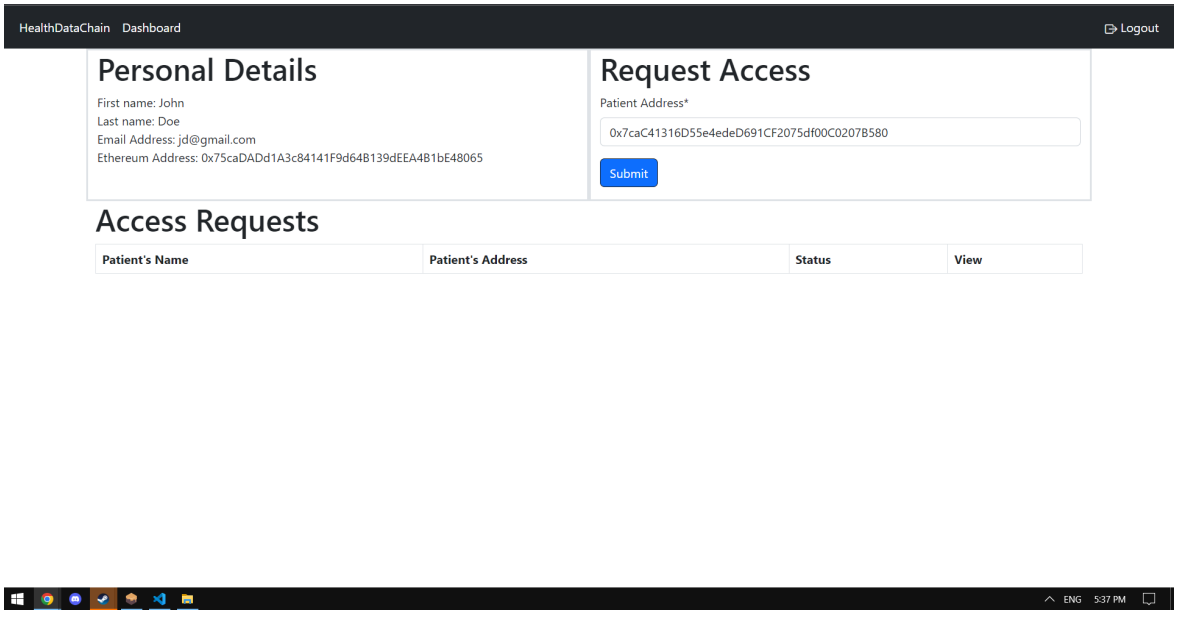

Figure 14: Requesting Access

Once the doctor clicks the submit button, their Access Requests section of the dashboard will be updated with the new access request. In here, the doctor can see the name and address of the patient they requested to, the request's status, which is pending by default, and a button to view the patient's data, that when pending will have a red color, signaling the doctor has no access.

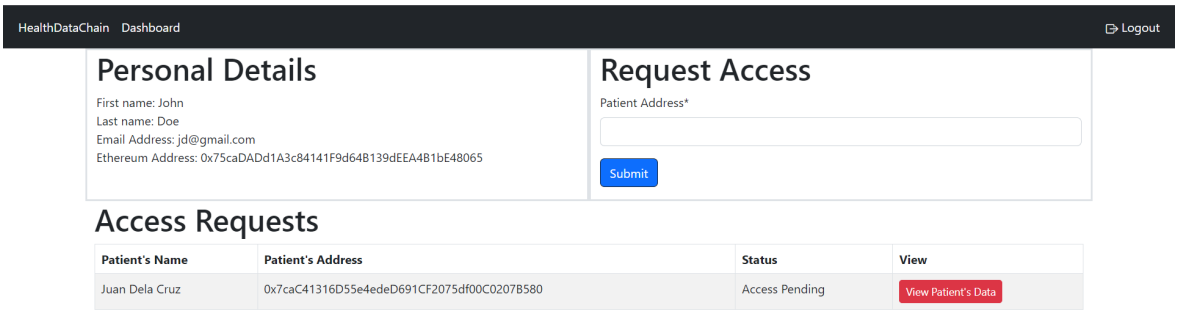

Figure 15: Access Requested

If the doctor presses the View Patient's Data button even if their request is still pending, a modal will appear that says they currently has no access to the patient's data.

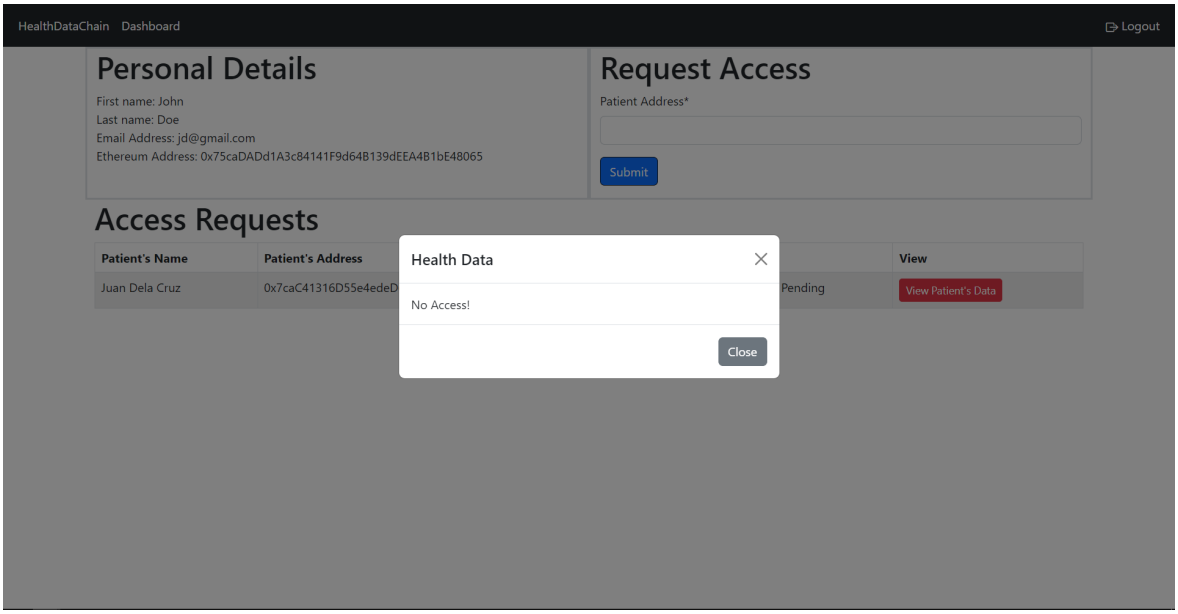

Figure 16: No Access!

However, if the doctor's request is approved, the status of the request will be updated to granted and the button to view the patient's data will turn green, signaling that the doctor now has access.

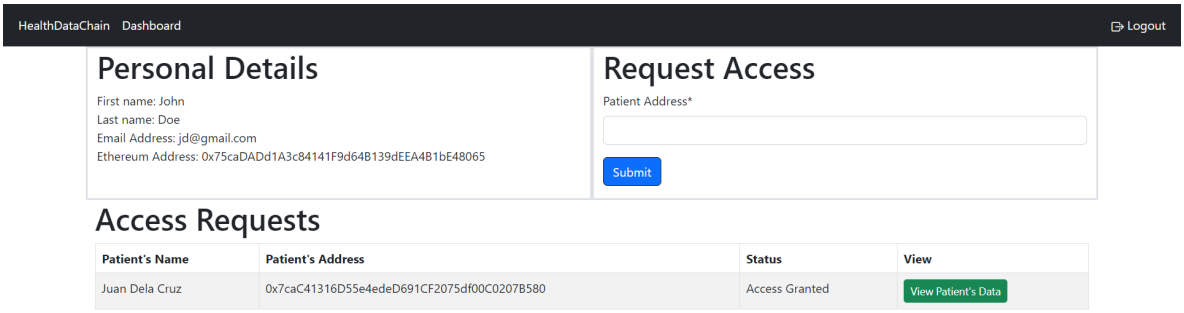

Figure 17: Access Request Approved

If the doctor wants to view the patient's data, they can simply click the View Patient's Data button, and they will be redirected to the patient's data and history page.

In there, the doctor can view the patient's current and previous health data, with the latest data showing up first.

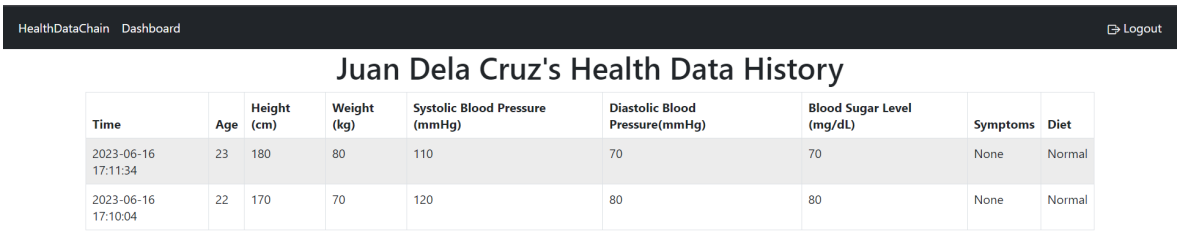

Figure 18: Patient's Data and History

Once the consultation with the patient ends, and the patient revokes the doctor's access to their data, the access request will be removed from the doctor's dashboard.

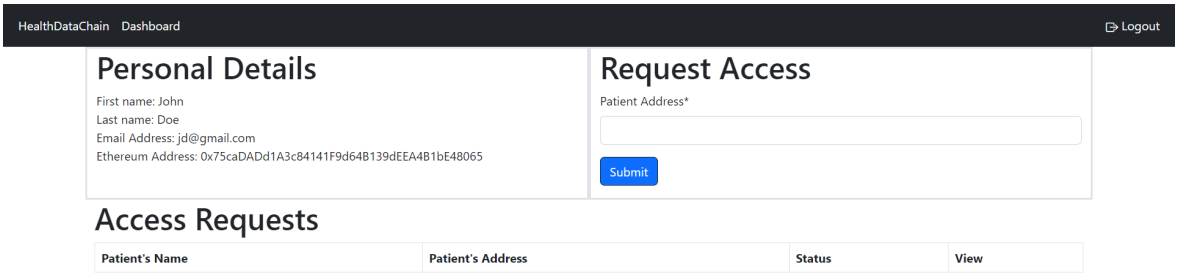

Figure 19: Request Removed

Should the patient have another consultation with the doctor, the doctor can request for access to the patient's data again.

D. Transaction Logs

Figure 20: Patient Data Transaction Log

After a patient has updated their data, their inputted data will be sent to the blockchain, and their corresponding data in the blockchain will be updated. The transaction log shows from whom the transaction is, in this case the patient, as well as the patient's inputs and the smart contract function that is called.

Figure 21: Access Request Transaction Log

After a doctor has requested access to the patient's data, their request will be sent to the blockchain. The transaction log shows that the transaction is from the doctor, as well as the patient's address from whom they want to request access to.

Figure 22: Access Granted Transaction Log

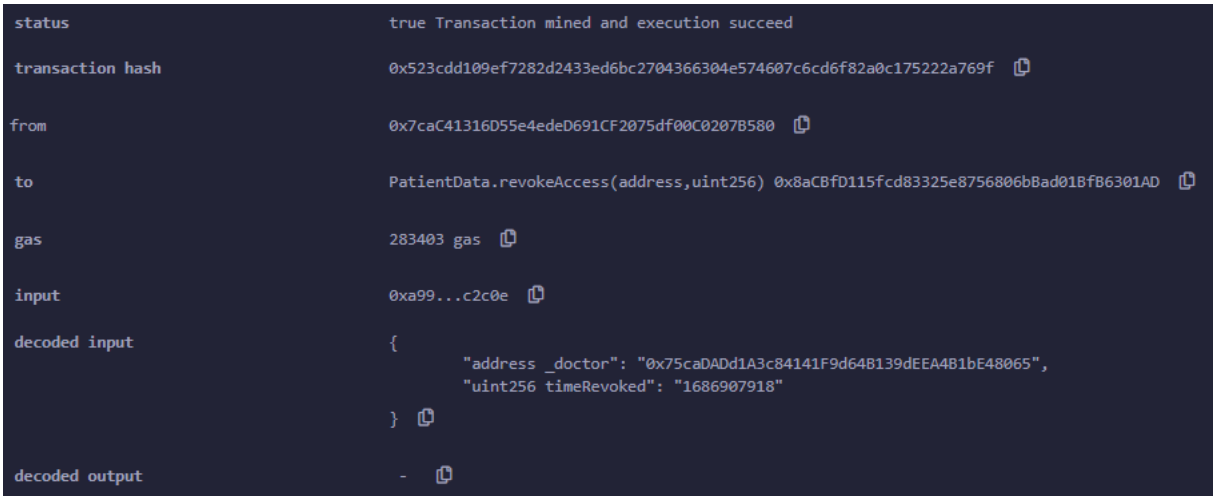

Figure 23: Access Revoked Transaction Log

The above figures both show the transaction logs that happens when a patient grants and revokes access to their data. In both logs, we can see the address of the doctor whom access is being granted and revoked, as well as a timestamp that contains the time when the access request is granted or revoked.

VI. Discussions

The blockchain application developed is a sharing platform for patients to share their patient-generated health data to their doctors, with the aid of the Ethereum blockchain for its access control capabilities. This section mainly discusses whether the developed application was able to fulfill all the objectives and address the stated problems.

A. Objectives and Addressed Problems

The developed application's main objective was to provide a platform for patient's to share their patient-generated health data to doctors, through an access control mechanism using Ethereum blockchain and Solidity smart contracts. For a doctor to access the patient's data, they first must request access to the patient. The patient can then approve the request, allowing the doctor to access their data. After the consultation is done, the patient can revoke the doctor's access to their data, preventing the doctor from accessing their data further.

The developed application was able to successfully meet the said objectives, along with other functionality such as user registration and authentication. Through the use of access control, the system was able to address concerns on the safety, security and privacy of their data, as they have full control on who is able to access their data.

B. Gas Cost Estimation

Gas consumption is often seen as a metric for the smart contract's efficiency. This means that more complex functions often leads to higher gas consumption when calling the function.

Figures 20, 21, 22, and 23 shows the estimated gas cost for each of the functions in the smart contract. However, these are in terms of Wei, in order to get the gas

cost in terms of Ether, they need to be converted first.

C. Security Analysis

- **Transparency** - One major concern of patients when it comes to using digital data like PGHD is knowing who can access and is accessing their data. This is addressed by recording access requests in the blockchain, that, with blockchain's immutable nature, guarantees that the patient can always see true information on who can access their data.

- **Provenance** - As said in the previous section, every access request is recorded in the blockchain, complete with information such as the doctor, as well as the timestamps of when it was granted or revoked. This allows the patients to always have a complete and accurate history in order to track who has previously accessed their data in the past.

- **Immutability** - The main reason why the blockchain platform was used. Blockchain's immutable nature basically makes the system immune to tampering or fabrication of data. This means that malicious doctors cannot simply manipulate access requests in order to gain access to patient's data without the patient knowing.

- **Integrity of Data** - Blockchain's immutability also guarantees that the patient data that both the patient and the doctor can see are true and accurate to what the patient inputted, since no malicious users can modify the patient's data, which can lead to misdiagnosis of the patient's condition by the doctor.

D. Significance

With advancements to wearable technology and more digitalization of data. It is inevitable that patient-generated health data will be used more and more. Once

more healthcare institutions and doctors adopt the use of PGHD, it will soon become the norm. The developed application provides a platform for the use of PGHD. With the implementation of Ethereum blockchain technology and smart contracts to the system, the application gives full control of a patient's data to the patient themselves. With the immutability and other properties of blockchain, the patients can be rest assured that their data is safe and secured with the system.

E. Issues and Challenges

With blockchain technology and smart contracts being relatively new technology to the developer, several challenges were faced during the development of the application. Major challenges include figuring out how to implement Ethereum blockchain into the web application, which was addressed through thorough learning of the Ethereum network and testing using libraries such as Web3.py, as well as programming in Solidity, as it is fundamentally different to other programming languages used by the developer in the past, there was a learning curve before the developer was able to program using solidity properly and be able to find and patch all possible loopholes in the smart contract as much as possible.

F. Differences with Main References

The main difference of the developed application with proposed systems used for reference is the use of patient-generated health data. PGHD was chosen to be used since in the Philippines, healthcare institutions are the ones who handles the patient's electronic health records, which prevents the patient to have full control over their data. Since the PGHD comes and is generated from patient themselves, the patient owns this data, and allows them to have full control over it. The application also introduces access requests, which the patient can approve and revoke, giving the control on who can access their health data.

G. Contributions

The main contributions of the developed application includes:

1. An Ethereum blockchain-based sharing platform where patients can securely share their PGHD to doctors.

2. An Access Control Scheme that focuses on the patient, allowing the patient to have full control on who can access their PGHD, as well as track those with access.

3. An application that enjoys Ethereum and blockchain's security and privacy-preserving features, which addresses concerns over use and sharing of digital data.

VII. Conclusions

The developed system was built as platform for patients to be able to share their PGHD to their doctors. Through the system, patients can input their PGHD, and doctors can request access to it. Patients can then approve and revoke access to their data. This gives them full control over who can access their data, as only doctors whose access requests are approved can access their data. The patient can also track which doctors can access their data, giving the full transparency.

In order to achieve a secure system, blockchain technology was integrated to the system. Blockchain allows data to be stored more securely compared to traditional centralized databases. With the nature of blockchain, tampering and malicious attempts to access data can prove to be close to or completely impossible.

Lastly, with the development of the application, due to the lack of healthcare systems that use PGHD, the developer hopes that the system can serve as a platform that allows PGHD to be used more and more in the future, and eventually being a norm, as well as to further showcase the security and privacy capabilities of blockchain, when it comes to healthcare systems.

VIII. Recommendations

Due to the nature of the Ethereum network being public and anyone has access to it, a private blockchain network such the Hyperledger Fabric can be used to further secure the application and the patient's data.

Other programming languages can be used for smart contract development can also be used, especially those with built-in and more robust access control features, such as OpenZeppelin.

For a more secure system, the use of digital identity for both patients and doctors can be used, especially for verifying if the doctors really are doctors, as the current system assumes that all doctors are real doctors.

Once more Wearable APIs are made publicly available, an automated way of inputting PGHD to the system can be implemented.

Lastly, for better user experience and UI, the frontend can be improved further.

IX. Bibliography

[1] S. Dua, "Digital communication management: The world is going digital," *International journal of recent research aspects*, vol. 4, no. 3, pp. 50–53, 2017.

[2] H. S. Jim, A. I. Hoogland, N. C. Brownstein, A. Barata, A. P. Dicker, H. Knoop, B. D. Gonzalez, R. Perkins, D. Rollison, S. M. Gilbert, *et al.*, "Innovations in research and clinical care using patient-generated health data," *CA: a cancer journal for clinicians*, vol. 70, no. 3, pp. 182–199, 2020.

[3] "What are patient-generated health data?," Jan 2018.

[4] W. A. Wood, A. V. Bennett, and E. Basch, "Emerging uses of patient generated health data in clinical research," *Molecular oncology*, vol. 9, no. 5, pp. 1018–1024, 2015.

[5] M. Hölbl, M. Kompara, A. Kamišalić, and L. Nemec Zlatolas, "A systematic review of the use of blockchain in healthcare," *Symmetry*, vol. 10, no. 10, p. 470, 2018.

[6] M. J. Deering, E. Siminerio, and S. Weinstein, "Issue brief: Patient-generated health data and health it," *Office of the National Coordinator for Health Information Technology*, vol. 20, 2013.

[7] A. Hoerbst and E. Ammenwerth, "Electronic health records," *Methods of information in medicine*, vol. 49, no. 04, pp. 320–336, 2010.

[8] "Benefits of ehrs," Oct 2017.

[9] D. J. Carey, S. N. Fetterolf, F. D. Davis, W. A. Faucett, H. L. Kirchner, U. Mirshahi, M. F. Murray, D. T. Smelser, G. S. Gerhard, and D. H. Ledbetter, "The geisinger mycode community health initiative: an electronic health record–linked

biobank for precision medicine research," *Genetics in medicine*, vol. 18, no. 9, pp. 906–913, 2016.

[10] I. Keshta and A. Odeh, "Security and privacy of electronic health records: Concerns and challenges," *Egyptian Informatics Journal*, vol. 22, no. 2, pp. 177–183, 2021.

[11] L. S. Sulmasy, A. M. López, and C. A. Horwitch, "Ethical implications of the electronic health record: in the service of the patient," *Journal of general internal medicine*, vol. 32, no. 8, pp. 935–939, 2017.

[12] A. Omoloja and S. Vundavalli, "Patient generated health data: Benefits and challenges," *Current Problems in Pediatric and Adolescent Health Care*, p. 101103, 2021.

[13] C. J. Wang and D. J. Huang, "The hipaa conundrum in the era of mobile health and communications," *JAMA*, vol. 310, no. 11, pp. 1121–1122, 2013.

[14] R. Collier, "New tools to improve safety of electronic health records," 2014.

[15] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain technology overview," *arXiv preprint arXiv:1906.11078*, 2019.

[16] E. P. E. Deepika and E. R. Kaur, "Cryptocurrency: Trends, perspectives and challenges," *International Journal of Trend in Research and Development*, vol. 4, no. 4, pp. 4–6, 2017.

[17] D. Vujičić, D. Jagodić, and S. Randic, "Blockchain technology, bitcoin, and ethereum: A brief overview," in *2018 17th international symposium infoteh-jahorina (infoteh)*, pp. 1–6, IEEE, 2018.

[18] "Features of blockchain," May 2022.

[19] S. Tanwar, K. Parekh, and R. Evans, "Blockchain-based electronic healthcare record system for healthcare 4.0 applications," *Journal of Information Security and Applications*, vol. 50, p. 102407, 2020.

[20] Y. Xiao, B. Xu, W. Jiang, Y. Wu, *et al.*, "The healthchain blockchain for electronic health records: development study," *Journal of Medical Internet Research*, vol. 23, no. 1, p. e13556, 2021.

[21] K. Shuaib, J. Abdella, F. Sallabi, and M. A. Serhani, "Secure decentralized electronic health records sharing system based on blockchains," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 8, pp. 5045–5058, 2022.

[22] S. Niu, L. Chen, J. Wang, and F. Yu, "Electronic health record sharing scheme with searchable attribute-based encryption on blockchain," *IEEE Access*, vol. 8, pp. 7195–7204, 2019.

[23] F. Hashim, K. Shuaib, and F. Sallabi, "Medshard: Electronic health record sharing using blockchain sharding," *Sustainability*, vol. 13, no. 11, p. 5889, 2021.

[24] A. Dubovitskaya, F. Baig, Z. Xu, R. Shukla, P. S. Zambani, A. Swaminathan, M. M. Jahangir, K. Chowdhry, R. Lachhani, N. Idnani, *et al.*, "Action-ehr: patient-centric blockchain-based electronic health record data management for cancer care," *Journal of medical Internet research*, vol. 22, no. 8, p. e13598, 2020.

[25] J. Ktari, T. Frikha, N. Ben Amor, L. Louraidh, H. Elmannai, and M. Hamdi, "Iomt-based platform for e-health monitoring based on the blockchain," *Electronics*, vol. 11, no. 15, p. 2314, 2022.

[26] I. M. Pires, G. Marques, N. M. Garcia, F. Flórez-Revuelta, V. Ponciano, and S. Oniani, "A research on the classification and applicability of the mobile health applications," *Journal of personalized medicine*, vol. 10, no. 1, p. 11, 2020.

[27] A. K. Yetisen, J. L. Martinez-Hurtado, B. Ünal, A. Khademhosseini, and H. Butt, "Wearables in medicine," *Advanced Materials*, vol. 30, no. 33, p. 1706910, 2018.

[28] J. Dunn, R. Runge, and M. Snyder, "Wearables and the medical revolution," *Personalized medicine*, vol. 15, no. 5, pp. 429–448, 2018.

[29] "What is blockchain technology? - ibm blockchain."

[30] B. Edmondson, "How does blockchain work for small business," Aug 2022.

[31] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE international congress on big data (BigData congress)*, pp. 557–564, Ieee, 2017.

[32] K. Pandey and A. K. Pandey, "What are the different layers of blockchain technology?," Sep 2022.

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

[34] L. Ismail and H. Materwala, "A review of blockchain architecture and consensus protocols: Use cases, challenges, and solutions," *Symmetry*, vol. 11, no. 10, p. 1198, 2019.

[35] H. Sheth and J. Dattani, "Overview of blockchain technology," *Asian Journal For Convergence In Technology (AJCT) ISSN-2350-1146*, 2019.

[36] S. M. Kerner and B. Lutkevich, "What is ethereum?," Jan 2022.

[37] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[38] K. Wu, "An empirical study of blockchain-based decentralized applications," *arXiv preprint arXiv:1902.04969*, 2019.

[39] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. Leung, "Decentralized applications: The blockchain-empowered software system," *IEEE Access*, vol. 6, pp. 53019–53033, 2018.

X. Appendix

A. Smart Contract

Patient Data Smart Contract

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract PatientData {
    struct Data {
        string name;
        uint age;
        uint height;
        uint weight;
        uint systolic;
        uint diastolic;
        uint bloodsugar;
        string symptoms;
        string diet;
        uint timestamp;
    }

    struct AccessRequest {
        address doctor;
        address patient;
        bool granted;
        uint timeGranted;
        uint timeRevoked;
    }

    mapping(address => Data) private patientData;
    mapping(address => AccessRequest[]) private accessRequests;
    mapping(address => AccessRequest[]) private previousRequests;
    mapping(address => Data[]) private dataHistory;

    event DataUpdated(address patient, string name, uint age, uint
           height, uint weight, uint systolic, uint diastolic,
          uint bloodsugar, string symptoms, string diet, uint
          timestamp);
    event AccessRequested(address patient, address doctor);
    event AccessGranted(address patient, address doctor);
    event AccessRevoked(address patient, address doctor);

    function updateData(string memory _name, uint _age, uint
          _height, uint _weight, uint _systolic, uint _diastolic,
          uint _bloodsugar, string memory _symptoms, string memory
           _diet, uint _timestamp) public {
        address patient = msg.sender;
        Data memory currentData = patientData[patient];

        dataHistory[patient].push(currentData);
               patientData[msg.sender] = Data(_name, _age, _height,
                      _weight, _systolic, _diastolic, _bloodsugar,
                      _symptoms, _diet, _timestamp);
        emit DataUpdated(msg.sender, _name, _age, _height, _weight,
               _systolic, _diastolic, _bloodsugar, _symptoms,
              _diet, _timestamp);
    }

    function readData(address _patient) public view returns (
          string memory _name, uint _age, uint _height, uint
          _weight, uint _systolic, uint _diastolic, uint
          _bloodsugar, string memory _symptoms, string memory
          _diet, uint _timestamp) {
        require(msg.sender == _patient || hasAccess(_patient, msg.
              sender), "Access denied");
        Data memory data = patientData[_patient];
        return (data.name, data.age, data.height, data.weight, data
              .systolic, data.diastolic, data.bloodsugar, data.
              symptoms, data.diet, data.timestamp);
    }

    function requestAccess(address _patient) public {
        AccessRequest[] storage requests = accessRequests[_patient
              ];
        for (uint i = 0; i < requests.length; i++) {
            require(requests[i].doctor != msg.sender, "Access
                  request already submitted");
        }
        requests.push(AccessRequest(msg.sender, _patient, false, 0,
              0));
        emit AccessRequested(_patient, msg.sender);
    }

    function grantAccess(address _doctor, uint timeGranted) public
          {
        require(msg.sender != _doctor, "Patients cannot grant
              access to themselves");
        AccessRequest[] storage requests = accessRequests[msg.
              sender];
        for (uint i = 0; i < requests.length; i++) {
            if (requests[i].doctor == _doctor) {
                requests[i].granted = true;
                requests[i].timeGranted = timeGranted;
                emit AccessGranted(msg.sender, _doctor);
                break;
            }
        }
    }

    function revokeAccess(address _doctor, uint timeRevoked)
          public {
        require(msg.sender != _doctor, "Patients cannot revoke
              access from themselves");
        AccessRequest[] storage requests = accessRequests[msg.
              sender];
        for (uint i = 0; i < requests.length; i++) {
            if (requests[i].doctor == _doctor) {

                requests[i].granted = false;
                requests[i].timeRevoked = timeRevoked;
                previousRequests[msg.sender].push(requests[i]);
                emit AccessRevoked(msg.sender, _doctor);

                for (uint j = i; j < requests.length - 1; j++) {
                    requests[j] = requests[j + 1];
                }
                requests.pop();
                break;
            }
        }
    }

    function hasAccess(address _patient, address _doctor) public
          view returns (bool) {
        AccessRequest[] storage requests = accessRequests[_patient
              ];
        for (uint i = 0; i < requests.length; i++) {
            if (requests[i].doctor == _doctor && requests[i].
                  granted) {
                return true;
            }
        }
        return false;
    }

    function getDataHistory(address _patient) public view returns
          (Data[] memory) {
        require(msg.sender == _patient || hasAccess(_patient, msg.
              sender), "Access denied");
        return dataHistory[_patient];
    }

    function getAccessRequests(address _patient) public view
          returns (AccessRequest[] memory) {
        return accessRequests[_patient];
    }

    function getPreviousRequests(address _patient) public view
          returns (AccessRequest[] memory) {
        return previousRequests[_patient];
    }
}
```

B. Django Files

Settings

```
"""
Django settings for SP project.

Generated by 'django-admin startproject' using Django 4.2.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/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/4.2/howto/deployment/
      checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-g*ub9fa&8db9gedxg-1#8t)q&7w(*e_jiro
      *^#9zpj16#ibs6+'

# 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',
    'webapp',
    'crispy_forms',
    'crispy_bootstrap5',
]

CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap5"

CRISPY_TEMPLATE_PACK = "bootstrap5"

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 = 'SP.urls'

AUTH_USER_MODEL = "webapp.AuthUser"

AUTHENTICATION_BACKENDS = ['webapp.backends.EmailBackend']

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 = 'SP.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

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


# Password validation
# https://docs.djangoproject.com/en/4.2/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/4.2/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/4.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto
      -field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Views

```
from multiprocessing import context

from django.shortcuts import render, redirect
from django.contrib import messages
from django.contrib.auth import authenticate, login, logout,
      update_session_auth_hash
from django.contrib.auth.forms import UserCreationForm,
      PasswordChangeForm
from django.contrib.auth.models import Group
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

from web3 import Web3
import json
from datetime import datetime

from .models import *
from .forms import *
from .backends import *

w3 = Web3(Web3.HTTPProvider('HTTP://127.0.0.1:7545'))

with open("webapp/contract.json") as f:
    abi = json.load(f)


contract_address = '0x8aCBfD115fcd83325e8756806bBad01BfB6301AD'
contract_abi = abi
```

```
contract = w3.eth.contract(address=contract_address, abi=
      contract_abi)

# Create your views here.
@login_required(login_url="login_page")
def index_page(request):

    return redirect("login_page")

def register_page(request):
    form = CreateUserForm()
    if request.method == "POST":
        form = CreateUserForm(request.POST)
        if form.is_valid():
            user = form.save()

            if user.user_type == 1:
                 Doctor.objects.create(
                      user=user,
                 )
            elif user.user_type == 2:
                Patient.objects.create(
                    user=user,
                )

            email = form.cleaned_data.get('email')

            messages.success(request, 'Account was created for ' +
                  email)

            return redirect('login_page')

    context = {'form': form}
    return render(request, 'webapp/register.html', context)

def login_page(request):
    if request.user.is_authenticated:
        if request.user.user_type == 1:
            return redirect('doctor_dashboard')
        else:
            return redirect('patient_dashboard')
    else:
        if request.method == 'POST':
            email = request.POST.get('email')
            password = request.POST.get('password')

            user = authenticate(request, email = email, password =
                  password)

            if user is not None:
                login(request, user)
                if user.user_type == 1:
                    return redirect('doctor_dashboard')
                else:
                    return redirect('patient_dashboard')
            else:
                messages.info(request, "Email or password is
                      incorrect!")

        context = {}
        return render(request, 'webapp/login.html', context)

@login_required(login_url="login_page")
def logOut_page(request):
    logout(request)
    return redirect('login_page')

@login_required(login_url="login_page")
def patient_dashboard(request):
    if request.user.user_type == 1:
        return redirect("doctor_dashboard")

    puser = request.user
    patient_address = puser.address

    print(request.user.email)

    if request.method == "POST":
        dform = PatientDataForm(request.POST)

        if dform.is_valid():
            name = request.user.first_name
            age = dform.cleaned_data['age']
            height = dform.cleaned_data['height']
            weight = dform.cleaned_data['weight']
            systolic = dform.cleaned_data['systolic']
            diastolic = dform.cleaned_data['diastolic']
            bloodsugar = dform.cleaned_data['bloodsugar']
            symptoms = dform.cleaned_data['symptoms']
            diet = dform.cleaned_data['diet']
            timestamp = int(datetime.now().timestamp())

            contract.functions.updateData(name, age, height, weight
                  , systolic, diastolic, bloodsugar, symptoms, diet
                  , timestamp).transact({'from': patient_address})

            return redirect('patient_dashboard')

    else:
        form = EditUserForm(instance=request.user)
        dform = PatientDataForm()

        data = contract.functions.readData(patient_address).call({'
              from': patient_address})

        temp = list(data)
        temp[9] = datetime.fromtimestamp(temp[9])
        data = tuple(temp)

        print(data)

        prev_data = contract.functions.getDataHistory(
              patient_address).call({'from': patient_address})
        prev_data = prev_data[::-1]

        if prev_data:
            prev_data.pop()

        for k, (a, b, c, d, e, f, g, h, i, j) in enumerate(
              prev_data):
            prev_data[k] = (a, b, c, d, e, f, g, h, i, datetime.
                  fromtimestamp(j))

        access_requests = contract.functions.getAccessRequests(
              patient_address).call()
        prev_requests = contract.functions.getPreviousRequests(
              patient_address).call()

        for l, (m, n, o, p, q) in enumerate(prev_requests):
            prev_requests[l] = (m, n, o, datetime.fromtimestamp(p),
                  datetime.fromtimestamp(q))

        doctors = Doctor.objects.all()

        context = {'form': form, 'user': puser, 'dform': dform, '
              data': data, 'access_requests': access_requests, '
              prev_requests': prev_requests, 'prev_data':
              prev_data, 'doctors': doctors}
        return render(request, 'webapp/pdashboard.html', context)

@login_required(login_url="login_page")
def doctor_dashboard(request):
    if request.user.user_type == 2:
        return redirect("patient_dashboard")

    puser = request.user
    doctor_address = puser.address

    print(request.user.email)

    if request.method == "POST":
        rform = AccessRequestForm(request.POST)

        if rform.is_valid():
            patient_address = rform.cleaned_data['patient_address']

            patient = AuthUser.objects.get(address=patient_address,
                  user_type=2)

            if not patient:
                return redirect('doctor_dashboard')
            else:
                access_requests = contract.functions.
                      getAccessRequests(patient.address).call()

                for req in access_requests:
                    if req[0] == doctor_address:
                        return redirect('doctor_dashboard')

                contract.functions.requestAccess(patient_address).
                      transact({'from': doctor_address})

        return redirect('doctor_dashboard')

    else:
        form = EditUserForm(instance=request.user)
        rform = AccessRequestForm()

        patient_addresses = AuthUser.objects.filter(user_type = 2).
              values_list('address', flat = True)

        list_requests = []
        access_requests = []
        data = []
```

```
        for address in patient_addresses:
            list_requests += contract.functions.getAccessRequests(
                  address).call()

        for tuple in list_requests:
            if tuple[0] == doctor_address:
                if contract.functions.hasAccess(tuple[1],
                      doctor_address).call():
                    data.append(("Has Access!",))
                else:
                    data.append(("No Access!",))

                request_data = (tuple,) + (data,)
                data = []
                access_requests.append(request_data)

        patients = Patient.objects.all()

        context = {'access_requests': access_requests, 'form': form
              , 'user': puser, 'rform': rform, 'patients':
              patients,}
        return render(request, 'webapp/ddashboard.html', context)

@login_required(login_url="login_page")
def patient_data(request, patient_address):
    if request.user.user_type == 2:
        return redirect("patient_dashboard")

    doctor_address = request.user.address

    patient = AuthUser.objects.get(address=patient_address)

    if not contract.functions.hasAccess(patient_address,
         doctor_address).call():
        return redirect("doctor_dashboard")
    else:
        data = contract.functions.readData(patient_address).call({'
              from': doctor_address})
        prev_data = contract.functions.getDataHistory(
              patient_address).call({'from': doctor_address})
        prev_data = prev_data[::-1]

        if prev_data:
            prev_data.pop()

        temp = list(data)
        temp[9] = datetime.fromtimestamp(temp[9])
        data = tuple(temp)

        for k, (a, b, c, d, e, f, g, h, i ,j) in enumerate(
              prev_data):
            prev_data[k] = (a, b, c, d, e, f, g, h, i, datetime.
                  fromtimestamp(j))

        context = {'patient': patient, 'data': data, 'prev_data':
              prev_data}

    return render(request, 'webapp/patient.html', context)


@login_required(login_url="login_page")
def grant_access(request, doctor_address):
    if request.user.user_type == 1:
        return redirect("doctor_dashboard")

    patient_address = request.user.address
    time = int(datetime.now().timestamp())

    contract.functions.grantAccess(doctor_address, time).transact
          ({'from': patient_address})

    return redirect("patient_dashboard")

@login_required(login_url="login_page")
def revoke_access(request, doctor_address):
    if request.user.user_type == 1:
        return redirect("doctor_dashboard")

    patient_address = request.user.address
    time = int(datetime.now().timestamp())

    contract.functions.revokeAccess(doctor_address, time).transact
          ({'from': patient_address})

    return redirect("patient_dashboard")
```

Forms

```
from django import forms
from django.forms import ModelChoiceField, ModelForm
from django.contrib.auth.forms import UserCreationForm,
      UserChangeForm
from django.contrib.auth.models import User

from .models import *

class CreateUserForm(UserCreationForm):
    class Meta:
        model = AuthUser
        fields = ['first_name', 'last_name', 'username', 'email', '
              password1', 'password2', 'user_type', 'address']

class EditUserForm(UserChangeForm):
    class Meta:
        model = User
        fields = ['email']
        labels = {
            'email' : 'Email Address'
        }

class PatientDataForm(forms.Form):
    #name = forms.CharField(label = "Name", max_length=100)
    age = forms.IntegerField(label = "Age")
    height = forms.IntegerField(label = "Height (in cm)")
    weight = forms.IntegerField(label = "Weight (in kg)")
    systolic = forms.IntegerField(label = "Systolic Blood Pressure
          (in mmHg)")
    diastolic = forms.IntegerField(label = "Diastolic Blood
          Pressure (in mmHg)")
    bloodsugar = forms.IntegerField(label = "Blood Sugar Level in
          (mg/dL)")
    symptoms = forms.CharField(label = "Symptoms", max_length=100)
    diet = forms.CharField(label = "Diet", max_length=100)

class AccessRequestForm(forms.Form):
    patient_address = forms.CharField(label='Patient Address',
          max_length=42)
```

Models

```
from django.db import models
from django.contrib.auth.models import AbstractUser

# Create your models here.
class AuthUser(AbstractUser):
    USER_TYPE_CHOICES = {
        (1, 'Doctor'),
        (2, 'Patient'),
    }

    email = models.EmailField(unique = True)
    user_type = models.PositiveSmallIntegerField(choices =
          USER_TYPE_CHOICES, null = True)
    address = models.CharField(max_length=42, blank = True, null =
          True, unique= True)

class Patient(models.Model):
    user = models.ForeignKey(AuthUser, on_delete=models.CASCADE,
          related_name="p",null=True)

    def __str__(self):
        return self.user.first_name

class Doctor(models.Model):
    user = models.ForeignKey(AuthUser, on_delete=models.CASCADE,
          related_name='d', null=True)

    def __str__(self):
        return self.user.first_name
```

Backends

```
from django.contrib.auth.backends import ModelBackend
from django.contrib.auth import get_user_model

class EmailBackend(ModelBackend):
    def authenticate(self, request, email=None, password=None, **
          kwargs):
        User = get_user_model()
        try:
            user = User.objects.get(email=email)
        except User.DoesNotExist:
            return None
        if user.check_password(password):
            return user
        return None
```

C. Templates

Register

```
<!DOCTYPE html>
<html lang="en">
    {% load crispy_forms_tags %}
<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">
    <!-- bootstrap -->
    {% include 'webapp/head-modules.html' %}
    <title>Register</title>
</head>
<body>


    <!-- contenthere -->

    <div class="container-fluid">

        <div class="row align-items-center my-auto bg-light vh
              -100">

            <div class="col">

                <div class="container-fluid bg-dark p-3 w-50">

                    <h5 class="text-white px-3">Register</h5>
                    <hr class="text-white">

                    <form method = "POST" action = "">
                        {% csrf_token %}
                        <div class="row bg-dark mx-auto">

                            <div class="col text-light">

                                <div class="mb-3">
                                    {{form.first_name |
                                          as_crispy_field}}
                                </div>

                                <div class="mb-3">
                                    {{form.last_name | as_crispy_field
                                          }}
                                </div>

                                <div class="mb-3">
                                    {{form.username | as_crispy_field
                                          }}
                                </div>

                                <div class="mb-3">
                                    {{form.email | as_crispy_field}}
                                </div>

                                <div>
                                    {{form.address| as_crispy_field}}
                                </div>
                            </div>

                            <div class="col text-light">
                                <div>
                                    {{form.password1 | as_crispy_field
                                          }}
                                </div>

                                <div>
                                    {{form.password2 | as_crispy_field
                                          }}
                                </div>

                                <div>
                                    {{form.user_type| as_crispy_field
                                          }}
                                </div>



                                <div class="row">
                                    <div class="col">
                                        <br><a class="text-decoration-
                                              none text-muted" href
                                              ="{% url 'login_page'
                                              %}">Log in</a>
                                    </div>

                                    <div class="col text-end">
                                        <br>
                                        <input class="btn btn-success
                                              border-dark" type="
                                              submit" value="Register
                                              ">
                                    </div>

                                </div>

                            </div>
                        </div>

                    </form>

                </div>

            </div>

        </div>
    </div>

    <!-- bootstrap -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist
          /js/bootstrap.bundle.min.js" integrity="sha384-
          ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8
          nbTov4+1p" crossorigin="anonymous"></script>
</body>
</html>
```

Login

```
<!DOCTYPE html>
<html lang="en">
    {% load crispy_forms_tags %}
<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">
    <!-- bootstrap -->
   {% include 'webapp/head-modules.html' %}
   <title>Login</title>
</head>
<body>


    <!-- contenthere -->
    <div class="container-fluid">

        <div class="row align-items-center my-auto bg-light vh
              -100">

            <div class="col">

                <div class="container-fluid bg-dark p-3 w-50">

                    <h5 class="text-white px-3">Login</h5>
                    <hr class="text-white">

                    <form method="POST" action="">
                        <div class="row bg-dark align-items-center
                              mx-auto">
                            {% load static %}

                            {% csrf_token %}
                            <div class="col text-light">

                                <div class="mb-3">
                                    <label for="emailInput"><i class="
                                          fa-solid fa-at"></i> Email
                                          </label>
                                    <input type="text" name="email"
                                          class="form-control">
                                </div>

                                <div>
                                    <label for="passwordInput"><i
                                          class="fa-solid fa-key"></i
                                          > Password </label>
                                    <input type="password" name="
                                          password" class="form-
                                          control">
                                </div>

                                <div class>
                                    {% for message in messages %}
                                        <div class = "alert alert-info
                                              ">
                                         {{message}}
                                        </div>
                                    {% endfor %}
```

```
                        </div>


                        <div class="row">
                            <div class="col">
                                <br><a class="text-decoration-
                                      none text-muted" href
                                      ="{% url 'register_page'
                                       %}">Register</a>
                            </div>

                            <div class="col text-end">
                                <br>
                                <input class="btn btn-success
                                      border-dark" type="
                                      submit" value="Login">
                            </div>
                        </div>
                    </div>
                </div>
            </form>
            <br>

        </div>

    </div>


</div>
</div>


<!-- bootstrap -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist
      /js/bootstrap.bundle.min.js" integrity="sha384-
      ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8
      nbTov4+1p" crossorigin="anonymous"></script>
</body>
</html>
```

Patient Dashboard

```
<html>
    {% load crispy_forms_tags %}
    {% include 'webapp/head-modules.html' %}
    <body>

    {% block navbar %}
        {% include 'webapp/navbar.html' %}
    {% endblock navbar %}

        <div class = "container">
            <div class = "row">
                <div class = "col border border-3">
                    <h1> Personal Details </h1>
                        First name: {{user.first_name}} <br>
                        Last name: {{user.last_name}} <br>
                        Email Address: {{user.email}} <br>
                        Ethereum Address: {{user.address}}
                </div>

                <div class = "col border border-3">
                    <h1> Health Data </h1>

                    <div class = "row">
                        <div class = "col">
                            Age = {{data.1}}
                        </div>

                        <div class = "col">
                            Height = {{data.2}} cm
                        </div>

                        <div class = "col">
                            Systolic = {{data.4}} mmHg
                        </div>
                    </div>

                    <div class = "row">
                        <div class = "col">
                            Symptoms = {{data.7}}
                        </div>

                        <div class = "col">
                            Weight = {{data.3}} kg
                        </div>

                        <div class = "col">
                            Diastolic = {{data.5}} mmHg
                        </div>
                    </div>

<div class = "row">
    <div class = "col">
        Diet = {{data.8}}
    </div>

    <div class = "col">
        Blood sugar = {{data.6}} mg/dL
    </div>

    <div class = "col">

    </div>
</div>

<div class = "row">
    <div class = "col">
        <button type="button" class="btn btn-
              primary" data-bs-toggle="modal"
              data-bs-target="#edit">
            Update
        </button>
    </div>

    <div class="modal fade" id="edit" data-bs-
          backdrop="static" data-bs-keyboard="
          false" tabindex="-1" aria-labelledby="
          staticBackdropLabel" aria-hidden="true
          ">
        <div class="modal-dialog modal-dialog-
              centered">
            <div class="modal-content">
                <div class="modal-header">
                <h1 class="modal-title fs-5" id="
                      staticBackdropLabel">Health
                      Data</h1>
                <button type="button" class="btn-
                      close" data-bs-dismiss="modal"
                      aria-label="Close"></button>
                </div>
                <div class="modal-body">
                    <form method = "POST" action = "">
                        {% csrf_token %}

                        {{dform.age| as_crispy_field}}
                        {{dform.height|
                              as_crispy_field}}
                        {{dform.weight|
                              as_crispy_field}}
                        {{dform.systolic|
                              as_crispy_field}}
                        {{dform.diastolic|
                              as_crispy_field}}
                        {{dform.bloodsugar|
                              as_crispy_field}}
                        {{dform.symptoms|
                              as_crispy_field}}
                        {{dform.diet| as_crispy_field
                              }}


                        <input class="btn btn-success
                              border-dark" type="
                              submit" value="Submit">
                    </form>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn
                          btn-secondary" data-bs-
                          dismiss="modal">Close</
                          button>
                </div>
            </div>
            </div>
    </div>

    <div class = "col">
        <button type="button" class="btn btn-
              primary" data-bs-toggle="modal"
              data-bs-target="#dhistory">
            View Health History
        </button>
    </div>

    <div class="modal fade" id="dhistory" data-
          bs-backdrop="static" data-bs-keyboard
          ="false" tabindex="-1" aria-labelledby
          ="staticBackdropLabel" aria-hidden="
          true">
        <div class="modal-dialog modal-dialog-
              centered auto modal-xl">
        <div class="modal-content">
            <div class="modal-header">
```

```
                    <h1 class="modal-title fs-5" id="
                          staticBackdropLabel">Health
                          Data History</h1>
                    <button type="button" class="btn-
                          close" data-bs-dismiss="modal"
                          aria-label="Close"></button>
                    </div>
                    <div class="modal-body">

                        <table class = "table table-
                              striped table-bordered
                              table-hover">
                            <thead class = "thead-light">
                                <tr>
                                    <th scope = "col">Time
                                          </th>
                                    <th scope = "col">Age</
                                          th>
                                    <th scope = "col">
                                          Height (cm)</th>
                                    <th scope = "col">
                                          Weight (kg)</th>
                                    <th scope = "col">
                                          Systolic Blood
                                          Pressure (mmHg)
                                          </th>
                                    <th scope = "col">
                                          Diastolic Blood
                                          Pressure(mmHg)</
                                          th>
                                    <th scope = "col">Blood
                                           Sugar Level (mg
                                          /dL)</th>
                                    <th scope = "col">
                                          Symptoms</th>
                                    <th scope = "col">Diet
                                          </th>
                                </tr>
                            </thead>
                            <tbody>
                                <tr>
                                    <td>{{data.9|date:"Y-m-
                                          d G:i:s"}}</td>

                                    <td>{{data.1}}</td>
                                    <td>{{data.2}}</td>
                                    <td>{{data.3}}</td>
                                    <td>{{data.4}}</td>
                                    <td>{{data.5}}</td>
                                    <td>{{data.6}}</td>
                                    <td>{{data.7}}</td>
                                    <td>{{data.8}}</td>
                                </tr>

                                {% for data in prev_data
                                     %}
                                <tr>
                                    <td>{{data.9|date:"
                                          Y-m-d G:i:s
                                          "}}</td>

                                    <td>{{data.1}}</td>
                                    <td>{{data.2}}</td>
                                    <td>{{data.3}}</td>
                                    <td>{{data.4}}</td>
                                    <td>{{data.5}}</td>
                                    <td>{{data.6}}</td>
                                    <td>{{data.7}}</td>
                                    <td>{{data.8}}</td>
                                </tr>
                                {% endfor %}

                            </tbody>
                        </table>

                    </div>
                    <div class="modal-footer">
                        <button type="button" class="btn
                              btn-secondary" data-bs-
                              dismiss="modal">Close</
                              button>
                    </div>
                </div>
                </div>
            </div>

        </div>

    </div>

</div>
```

```
<div class = "row align-items-center">
    <div class = "col">
        <h1> Doctor Access </h1>
    </div>

    <div class = "col align">
        <button type="button" class="btn btn-primary"
              data-bs-toggle="modal" data-bs-target="#
              history">
            View Access History
        </button>
    </div>
</div>

<div class = "row">
    <div class = "col">

        <table class = "table table-striped table-
              bordered table-hover">
            <thead>
                <tr>
                    <th scope = "col">Doctor's Name</th>
                    <th scope = "col">Doctor's Address</
                          th>
                    <th scope = "col">Status</th>
                    <th scope = "col">Approve/Revoke</th>
                </tr>
            </thead>
            <tbody>
                {% for request in access_requests %}
                    <tr>
                        <td>
                            {% for doctor in doctors %}
                                {% if doctor.user.address
                                      == request.0 %}
                                    Dr. {{doctor.user.
                                          first_name}} {{
                                          doctor.user.
                                          last_name}}
                                {% endif %}
                            {% endfor %}
                        </td>
                        <td>{{request.0}}</td>
                        {% if request.2 %}
                        <td>Access Granted</td>
                        <td>
                            <a class="btn btn-danger btn-
                                  sm" href="{% url '
                                  revoke_access' request.0
                                   %}"> Revoke </a>
                        </td>
                        {% else %}
                        <td>Access Pending</td>
                        <td>
                            <a class="btn btn-success btn-
                                  sm" href="{% url '
                                  grant_access' request.0
                                  %}"> Approve </a>
                        </td>
                        {% endif %}
                    </tr>
                {% endfor %}
            </tbody>
        </table>

    </div>
</div>


    <div class="modal fade" id="history" data-bs-
          backdrop="static" data-bs-keyboard="false"
          tabindex="-1" aria-labelledby="
          staticBackdropLabel" aria-hidden="true">
        <div class="modal-dialog modal-dialog-centered
              modal-xl">
        <div class="modal-content">
            <div class="modal-header">
            <h1 class="modal-title fs-5" id="
                  staticBackdropLabel">Previous Requests
                  </h1>
            <button type="button" class="btn-close" data
                  -bs-dismiss="modal" aria-label="Close
                  "></button>
            </div>
            <div class="modal-body">
                <table class = "table table-striped table
                      -bordered table-hover">
                    <thead class = "thead-light">
                        <tr>
                            <th scope = "col">Doctor's
                                  Name</th>
                            <th scope = "col">Doctor's
```

```
                                    Address</th>
                                <th scope = "col">Time Granted
                                    </th>
                                <th scope = "col">Time Revoked
                                    </th>
                            </tr>
                        </thead>
                        <tbody>
                            {% for request in prev_requests %}
                                <tr>
                                    <td>
                                        {% for doctor in
                                            doctors %}
                                            {% if doctor.user.
                                                address ==
                                                request.0 %}
                                                Dr. {{doctor.
                                                    user.
                                                    first_name
                                                    }} {{
                                                    doctor.
                                                    user.
                                                    last_name
                                                    }}
                                            {% endif %}
                                        {% endfor %}
                                    </td>
                                    <td>{{request.0}}</td>
                                    <td>{{request.3|date:"Y-m-
                                        d G:i:s"}}</td>
                                    <td>{{request.4|date:"Y-m-
                                        d G:i:s"}}</td>
                                </tr>
                            {% endfor %}
                        </tbody>
                    </table>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-
                        secondary" data-bs-dismiss="modal
                        ">Close</button>
                </div>
            </div>
            </div>
        </div>

    </div>
</div>

</body>
</html>
```

Doctor Dashboard

```
<html>
    {% load crispy_forms_tags %}
    {% include 'webapp/head-modules.html' %}
    <body>

    {% block navbar %}
        {% include 'webapp/navbar.html' %}
    {% endblock navbar %}

        <div class = "container">
            <div class = "row">
                <div class = "col border border-3">
                    <h1> Personal Details </h1>
                        First name: {{user.first_name}} <br>
                        Last name: {{user.last_name}} <br>
                        Email Address: {{user.email}} <br>
                        Ethereum Address: {{user.address}}
                </div>

                <div class = "col border border-3">
                    <h1> Request Access </h1>

                    <form method = "POST" action = "">
                        {% csrf_token %}
                        {{rform.patient_address | as_crispy_field}}

                        <input class="btn btn-primary border-dark"
                               type="submit" value="Submit">
                    </form>
                </div>
            </div>

            <div class = "row">
                <h1> Access Requests </h1>
            </div>

            <div class = "row">
                <div class = "col">

<table class = "table table-striped table-
      bordered table-hover">
    <thead>
        <tr>
            <th scope = "col">Patient's Name</th>
            <th scope = "col">Patient's Address</
                th>
            <th scope = "col">Status</th>
            <th scope = "col">View</th>
        </tr>
    </thead>
    <tbody>
        {% for request in access_requests %}
            <tr>
                <td>
                    {% for patient in patients %}
                        {% if patient.user.address
                            == request.0.1 %}
                            {{patient.user.
                                first_name}} {{
                                patient.user.
                                last_name}}
                        {% endif %}
                    {% endfor %}
                </td>
                <td>{{request.0.1}}</td>
                {% if request.0.2 %}
                <td>Access Granted</td>
                <td>
                    <a class="btn btn-success btn-
                        sm" href="{% url '
                        patient_data' request
                        .0.1 %}">View Patient's
                        Data</a>
                </td>
                {% else %}
                <td>Access Pending</td>
                <td>
                    <button type="button" class="
                        btn btn-danger btn-sm"
                        data-bs-toggle="modal"
                        data-bs-target="#viewB">
                       View Patient's Data
                    </button>

                    <div class="modal fade" id="
                        viewB" data-bs-backdrop
                        ="static" data-bs-
                        keyboard="false"
                        tabindex="-1" aria-
                        labelledby="
                        staticBackdropLabel"
                        aria-hidden="true">
                       <div class="modal-dialog
                            modal-dialog-
                            centered">
                       <div class="modal-content
                            ">
                           <div class="modal-
                                header">
                           <h1 class="modal-title
                                fs-5" id="
                                staticBackdropLabel
                                ">Health Data</
                                h1>
                           <button type="button"
                                class="btn-close
                                " data-bs-
                                dismiss="modal"
                                aria-label="
                                Close"></button>
                           </div>
                           <div class="modal-body
                                ">
                              {{request.1.0.0}}
                           </div>
                           <div class="modal-
                                footer">
                           <button type="button"
                                class="btn btn-
                                secondary" data-
                                bs-dismiss="
                                modal">Close</
                                button>
                           </div>
                       </div>
                       </div>
                    </div>
                </td>
                {% endif %}
            </tr>
```

```
                    {% endfor %}
                </tbody>
            </table>
        </div>
    </div>

</body>
</html>
```

Patient's Data and History

```
<html>
    {% load crispy_forms_tags %}
    {% include 'webapp/head-modules.html' %}

    {% block navbar %}
        {% include 'webapp/navbar.html' %}
    {% endblock navbar %}

    <body>
        <div class = "container">
            <div class = "row">
                <center><h1>{{patient.first_name}} {{patient.
                    last_name}}'s Health Data History</h1></
                    center>
            </div>

            <div class = "row">
                <table class = "table table-striped table-bordered
                    table-hover">
                    <thead class = "thead-light">
                        <tr>
                            <th scope = "col">Time</th>
                            <th scope = "col">Age</th>
                            <th scope = "col">Height (cm)</th>
                            <th scope = "col">Weight (kg)</th>
                            <th scope = "col">Systolic Blood Pressure
                                (mmHg)</th>
                            <th scope = "col">Diastolic Blood
                                Pressure(mmHg)</th>
                            <th scope = "col">Blood Sugar Level (mg/
                                dL)</th>
                            <th scope = "col">Symptoms</th>
                            <th scope = "col">Diet</th>
                        </tr>
                    </thead>
                    <tbody>
                        <tr>
                            <td>{{data.9|date:"Y-m-d G:i:s"}}</td>
                            <td>{{data.1}}</td>
                            <td>{{data.2}}</td>
                            <td>{{data.3}}</td>
                            <td>{{data.4}}</td>
                            <td>{{data.5}}</td>
                            <td>{{data.6}}</td>
                            <td>{{data.7}}</td>
                            <td>{{data.8}}</td>
                        </tr>
                        {% for datum in prev_data %}
                            <tr>
                                <td>{{datum.9|date:"Y-m-d G:i:s"}}</
                                    td>
                                <td>{{datum.1}}</td>
                                <td>{{datum.2}}</td>
                                <td>{{datum.3}}</td>
                                <td>{{datum.4}}</td>
                                <td>{{datum.5}}</td>
                                <td>{{datum.6}}</td>
                                <td>{{datum.7}}</td>
                                <td>{{datum.8}}</td>
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </body>
</html>
```

Navbar

```
<nav class="navbar navbar-expand navbar-dark bg-dark">
    <div style="display: flex;" class="container-fluid">
        <button class="navbar-toggler" type="button" data-bs-toggle
            ="collapse" data-bs-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false" aria-
                label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link active" aria-current="page"
                        href="">HealthDataChain</a>
                </li>
                {% if request.user.user_type == 0 %}
                <li class="nav-item">
                    <a class="nav-link active" aria-current="page"
                        href="{% url 'patient_dashboard' %}">
                        Dashboard</a>
                </li>
                {% endif %}

                {% if request.user.user_type == 1 %}
                <li class="nav-item">
                    <a class="nav-link active" aria-current="page"
                        href="{% url 'doctor_dashboard' %}">
                        Dashboard</a>
                </li>
                {% endif %}

            </ul>
                <div class="navbar-nav ms-auto">
                <ul class="nav navbar-nav navbar-right ">
                    <a class="nav-link text-light active" href="{%
                        url 'logout_page' %}">
                    <i class="bi bi-box-arrow-right"></i>
                        Logout
                    </a>
                </ul>
        </div>
    </div>

</nav>
```

Modules

```
{% load static %}
<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 rel="stylesheet" href="{% static 'main.css' %}">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/
    bootstrap.min.css" rel="stylesheet">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
    bootstrap-icons@1.3.0/font/bootstrap-icons.css">
<link href="https://fonts.googleapis.com/css?family=Poppins
    :300,400,500,600,700,800,900" rel="stylesheet">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/
    bootstrap.bundle.min.js"></script>
<script src="https://kit.fontawesome.com/70e2a3091b.js"
    crossorigin="anonymous"></script>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
    @fortawesome/fontawesome-free@6.1.1/css/fontawesome.min.css
    ">
```

XI. Acknowledgment

I would like to thank the Lord, first of all, for His guidance, all throughout my stay in UP.

I would also like to thank my parents and siblings, for always being there for me, and for their neverending support and constant encouragement for me, especially when I needed it.

I would also like to thank Sir Marbert Marasigan and Sir Richard Bryann Chua, for giving me the guidance I need in order to finish this SP. Blockchain is a very new topic for me and without their help, I would not even come close to finishing it.

I would also like to thank my classmates, as well as my high school friends, for the endless study sessions, requirements cramming, and for just always being there for me in general. I will miss you all!

I may not be able to name everyone who helped me along this journey, but just know that I will forever be grateful for all the support and guidance that I have received in my stay in UP.