dBio: Healthcare Data on Web 3.0

CSCI E-599, Spring 2022
Harvard University

David Crowe, Evans Patel, Narek Asadorian,
Stephen Sheldon, Zuzana Matevossian

Customer: Bach Adylbekov
Course Staff: Prof. Peter Henstock, Roman Burdakov

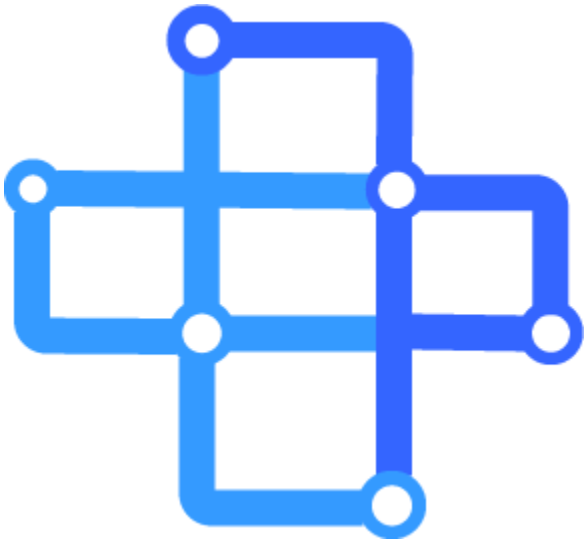

Table of Contents

Abstract

Recent decades have seen the introduction of systems and standards designed to improve Electronic Medical Record (EMR) accessibility and interoperability. However, in practice, EMRs are often siloed in databases across multiple healthcare providers, each with their own access mechanisms. Furthermore, patients lack a simple way to claim ownership of and control access to their sensitive medical records. To address these challenges, the dBio decentralized application stores patient data in a peer-to-peer public storage network, encrypts the data and manages secure access using proxy re-encryption, and optionally grants ownership on a blockchain. dBio supports four primary workflows: *read, write, access control*, and *claim ownership* of EMRs. The system architecture includes four primary components: a Fast Healthcare Interoperability Resources (FHIR) proxy server, backend protocol server, user interface, and smart contract. dBio gives patients secure control and ownership over an exhaustive set of their EMRs.

Keywords: *Electronic Medical Records, FHIR, Interplanetary File System, Ironcore SDK, proxy re-encryption, Solidity, Ethereum, blockchain*

1. Introduction

Electronic Medical Record (EMR) interoperability, patient control over data access, and patient ownership of their EMRs are long standing challenges in healthcare. These challenges persist despite multiple standards published by organizations like Health Level 7 (HL7) intended to improve patient access and organizational interoperability. HL7 introduced their v1 standard in 1987 as a proof of concept implementation to define the standard's content and structure. HL7 v2 was introduced in 1989 to address integration between administrative and clinical hospital systems, but lacked a unifying ontology to make it effective at scale. HL7 v3 introduced a central information model in 1995, but was widely considered a failure for being "too complex and expensive to implement in real world systems" [1]. In response to the byzantine HL7 v3, an organization of open source contributors began developing the HL7 Fast Healthcare Interoperability Resources (FHIR) standard in 2011 with a focus on usability and adoption. In 2020, the U.S. Centers for Medicare and Medicaid Services (CMS) released the *Interoperability and Patient Access Final Rule* (*CMS-9115-F*), which required all payers regulated by CMS to implement the HL7 FHIR standards by July 1, 2021. In the *Final Rule*, the CMS emphasized

patient privacy and medical record interoperability in their stated goals of "improving patient access and care, alleviating provider burden, and reducing overall health care costs, all while taking steps to protect the privacy and security of patients' personal health information" [2]. As HL7 and FHIR standards evolve, interoperability and patient control over their data continue to be underpinning principles.

Despite improvements in standardization efforts, healthcare record systems are still more provider-centric than patient-centric with low interoperability [3]. Healthcare records are collected by providers, and stored in provider-selected EMR and database systems. This provider-centric approach to data management results in centralized data silos each with their own process to retrieve patient data, making it difficult to compile comprehensive sets of medical records. Furthermore, data silos are vulnerable to cyber attacks and data theft [4]. For example, in 2017 a cyber-attack utilizing WannaCry ransomware affected 80 hospitals in Britain's National Health Service network preventing access to patient EMRs [5].

In recent years, advances in decentralized system and blockchain technologies have unlocked new opportunities to address long standing challenges in healthcare data security, as this paper will explore in detail. For example, smart contracts on the blockchain allow immutable, trustless tracking of EMR ownership; peer-to-peer distributed storage systems eliminate data silos; and proxy re-encryption enables secure sharing of EMRs. The integration between decentralized web3 technologies with the FHIR standard will bring the healthcare data ecosystem into the 21st century, giving patients more control and ownership.

2. Background / Existing Work

Existing healthcare data solutions on the blockchain avoid writing patient data to the public ledger due to privacy and security concerns. Specifically, storing personal health information (PHI) on a public blockchain poses risks to compliance with the Health Insurance Portability and Accountability Act (HIPAA). Thus, existing approaches store data in "off chain" storage while storing pointers to the data on the blockchain [6]. For example, FHIRChain [7] aims to enable clinical data exchange via decentralized app (dApp) to address the lack of trust and interoperability in existing solutions. FHIRChain employs public key cryptography to identify

practitioners and patients by their public key. Patient data is stored off-chain in a centralized data store, with a "sign then encrypt" scheme applied to data pointers on the blockchain so recipients can verify the signature of the sending party. The FHIR standard is enforced by FHIRChain's model-view-controller upon insertion and update of resources, but while FHIRChain authenticates users using public key encryption it does not allow users to track ownership or control data access. Furthermore, given that patient EMRs are stored in FHIRChain's centralized data store, the system simply creates a new data silo.

Several commercial blockchain projects aim to improve interoperability among healthcare providers and improve access control over medical data. Permissioned blockchains[1] are a natural fit for this problem as they allow administrators to define who can access the blockchain and what data they have access to. Therefore, Hyperledger Fabric created by the Linux Foundation stands out as a strong fit for blockchain products dealing with EMRs [8]. Hyperledger has been adopted by projects such as Holocare [9] and B4Health [10] that create networks of hospitals, each with its own Certificate Authority (CA). The projects centrally manage CAs to define data access permissions for who can access the network and add or read transactions. Notably, Holocare and B4Health both aim to improve EMR interoperability and data access control without a particular focus on patient access. Healthchain stores encrypted EMRs on the Interplanetary File System (IPFS), records a hash of the EMR on chain in Hyperledger Fabric, and grants users sole responsibility for managing permissions to their EMRs [11]. While Healthchain provides patient's control over their EMRs, there is an opportunity to further enhance the public key cryptography mechanism by implementing proxy re-encryption. Proxy re-encryption involves re-encrypting data from one public key to another without accessing private keys or the plaintext data [12], and offers a path to secure data access control, as this paper will explore in detail.

Blockchains other than Hyperledger can also serve as the backbone of systems that prioritize EMR access control and interoperability. For example, MedRec is a private Ethereum platform

[1] Investopedia defines "a permissioned blockchain" as "a distributed ledger that is not publicly accessible. It can only be accessed by users with permissions. The users can only perform specific actions granted to them by the ledger administrators and are required to identify themselves through certificates or other digital means" (Investopedia, 2022).

enabling patients to audit and take ownership of their medical records in the context of clinical trials [13]. MedRec uses smart contracts to record EMR metadata to the blockchain including owner, view permissions, and retrieval location. Medical researchers are incentivized to contribute computational resources to sustain the network via a token mining mechanism. In return, researchers earn census level, anonymized EMR metadata. MedRec only stores hashes and pointers on the blockchain; healthcare providers are required to store the actual EMRs on centralized databases in FHIR format. 1UpHealth maintains a private blockchain called 1Up on which it stores provenance information for FHIR resources. Whenever medical records are inserted, exchanged, modified, or deleted, a new block is minted containing a hash of the record and relevant metadata [14]. While 1UpHealth exposes an API endpoint to serve the provenance of resources, it also incentivizes healthcare providers to participate in its network by maintaining nodes, earning and spending 1Up coins, and verifying transactions themselves. However, neither MedRec nor 1UpHealth maintains the patients' data on a peer-to-peer storage medium and thus misses on decentralization.

3. dBio Contribution

dBio is a healthcare data privacy platform that empowers patients to take full ownership of their medical records. Today, it is not easy for patients and medical providers to get access to comprehensive medical records, as the records are siloed in databases across multiple healthcare providers. Further, patients do not have a simple way to claim ownership of and control access to their sensitive medical records. To address these challenges, dBio stores patient data in a peer-to-peer distributed network, secures the data and manages access using proxy re-encryption, and records ownership to blockchain.

dBio's core workflows are *upload*, *view*, *control access*, and *claim ownership* of Electronic Medical Records (EMRs). Using the *upload* workflow, medical providers upload EMRs to dBio on behalf of patients via a Fast Healthcare Interoperability Resources (FHIR) server. The *view* workflow allows patients to view resources that have been uploaded for them in the dBio user interface. Users can request access to 3rd party EMRs (e.g., a medical provider requesting access to a patient's records), and grant 3rd parties access to their EMRs (e.g., a patient granting a

medical provider access to their EMRs). Patients can claim ownership of their EMRs and register ownership on the Ethereum blockchain.

The dBio system architecture includes four primary components: a Java Enterprise Edition HAPI FHIR proxy server, user interface, protocol layer, and Solidity smart contract. The HAPI FHIR server is an implementation of the open-source HAPI standard to ensure EMR interoperability. dBio improves FHIR server functionality by encrypting EMRs before they are sent across the internet to the dBio protocol layer. Specifically, before uploading a new EMR, the FHIR proxy server calls the Ironcore SDK to create an access control group including the EMR's uploader (e.g., a medical provider) and owner (e.g., a patient). The EMR is then encrypted using the group transfer re-encryption key and sent to the dBio protocol layer. This is the first step in dBios secure, end to end EMR encryption.

Patients can view, claim ownership, and control access to EMRs via the React-native user interface. Patients login with their existing gmail address via Auth0 single sign-on (SSO). The authentication workflow is integrated with Tor.us CustomAuth SDK, which either retrieves the users' previously known Ethereum credentials or creates an Ethereum address for the user based on Auth0 authentication. Once logged in, the Ironcore SDK is initiated and used to manage EMR access control throughout the application. Ironcore allows patients to create access control groups, then uses proxy re-encryption to enable members of the group to decrypt the original users EMRs. Proxy re-encryption uses three related keys in the encryption workflow [15]. First, EMRs are encrypted using the access control group's public key. When a group member attempts to decrypt the EMR, a group transform key converts it into ciphertext readable by individual members of the group. Then each group member uses their private key to decrypt the ciphertext and render it as plaintext. A key feature of proxy re-encryption is that the party doing the transformation cannot decrypt the ciphertext while transforming it. The transformer never gains access to the plaintext EMR and does not have access to the user's private keys. If a patient wants to remove a third party's access to their health records, dBio removes them from the Ironcore access control group, and the third party can no longer decrypt the user's EMRs.

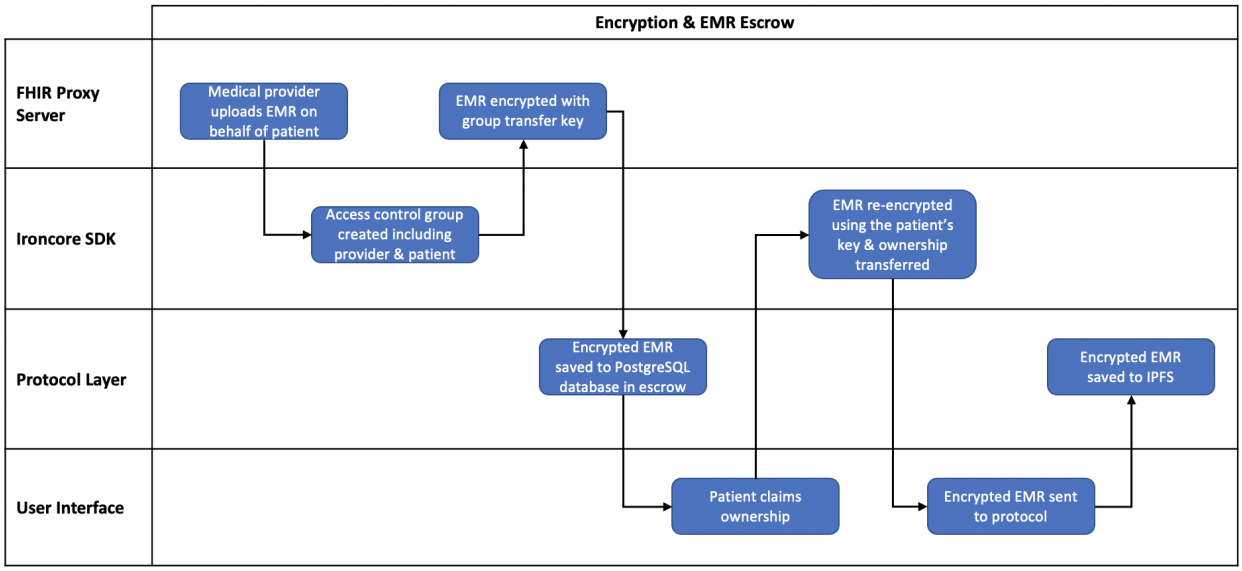

Figure 3.1 - Encryption & EMR escrow workflow

dBio's protocol layer handles requests from the FHIR server and the user interface by implementing a Rust based API, which communicates with the open-source PostgreSQL database and the Interplanetary File System (IPFS). The API covers dBio's core functionality including *upload*, *view*, *control access*, and *claim ownership*. For uploads, the protocol layer initially stores EMRs in the PostgreSQL database in escrow. Once a patient claims ownership of the EMRs, the protocol layer saves the encrypted EMR to IPFS and removes it from escrow in PostgreSQL. IPFS is a distributed, peer-to-peer file system for storing and sharing data. Storing EMRs on IPFS allows dBio users to break data silos inherent in existing systems and to access a comprehensive set of their medical records from all medical providers. Files can be stored on and accessed from multiple IPFS nodes, eliminating single points of failure. Each file stored to IPFS has a unique content address and is publicly accessible. Notably, dBio only stores all EMRs as ciphertext, encrypted with AES256-GCM using IroncoreSDK, and not the original plaintext EMRs. Therefore, even if a bad actor knew an EMR's IPFS web address, they would not be able to decrypt the patient's data without the patient's Ethereum Ironcore identity, generated based on the Ethereum private key.

dBio deploys a Solidity smart contract to the Ethereum blockchain to record EMR ownership. When a user claims ownership of their EMRs, the protocol layer creates a voucher signed by the

dBio private key. Vouchers are created as part of a “lazy minting” process and can be redeemed for NFTs [16]. NFTs are cryptographic assets that are recorded to blockchain and possess unique identification codes and metadata to distinguish them from one another. Vouchers contain all the information that will go into the actual NFT including the user's public key address. When a patient wants to record EMR ownership, they redeem the signed voucher. Once redeemed, the NFT is transferred to the buyer. In this way the dBio user is registered as the owner of the EMR on the blockchain. In order to record ownership of their EMRs on the blockchain, users must agree to pay processing (also known as “gas”) fees to mint the NFT and record ownership to the blockchain.

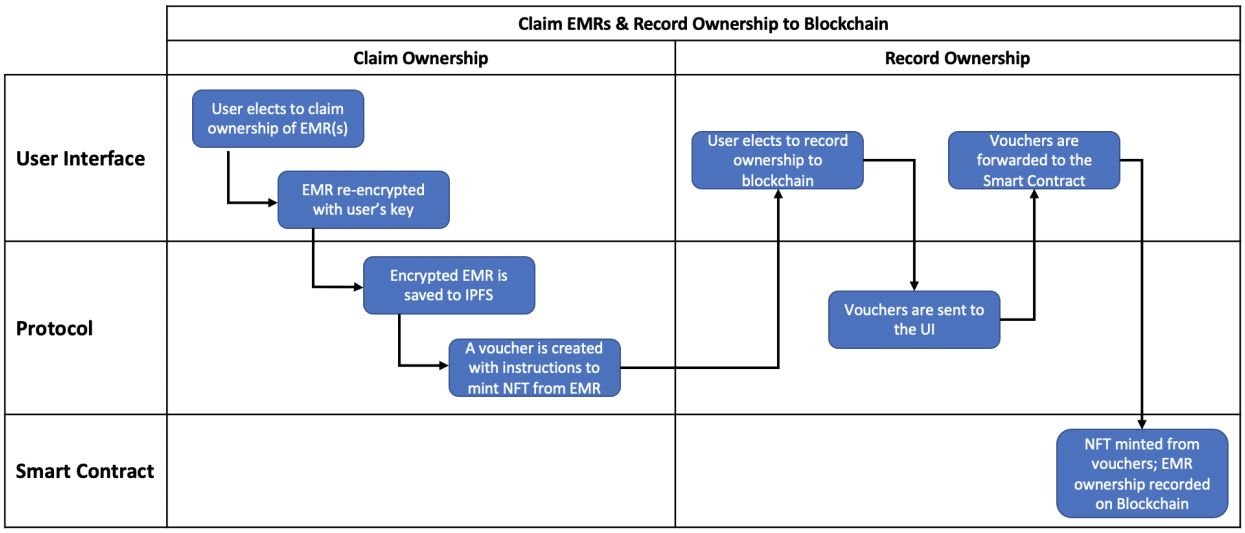

Figure 3.2 - Claim & Record Ownership of EMRs to Blockchain workflow

4. Future Work

The current architecture of dBio blends traditional Web 2.0 components such as OAuth, REST endpoints, and a centralized protocol server with decentralized web3 technologies like IPFS, proxy re-encryption, and the Ethereum blockchain. In a future iteration of the platform, we would like to advance the application architecture to become more decentralized and trustless. For example, the central protocol server and its metadata state can be transitioned to a peer to peer protocol where independent protocol nodes work collaboratively to reach consensus on system state. Furthermore the proxy re-encryption scheme which is currently handled by trusted

party IronCore Labs can be evolved into a threshold re-encryption scheme like NuCypher's Umbral.

dBio's long term vision involves a blockchain-based monetization model for patient healthcare data on top of a decentralized architecture. The model will establish a cryptocurrency based on Ethereum, to be utilized in all transactions involving patient data and protocol operations. Researchers and clinicians will be able to perform paid encrypted searches on de-identified fields of patient records to find clinical study participants, or to generate data sets of interest for research analysis. When patients approve requests for use by such third parties, they will be paid in cryptocurrency to release their data. The model will provide additional economic incentives for healthcare providers to own and maintain FHIR proxies and protocol nodes for public use, minting more currency for their wallets when facilitating transactions and releasing records to patients.

5. Conclusion

Patients and healthcare providers both benefit when EMRs are interoperable, secure, and patient controlled and owned. Systems that prioritize these characteristics can allow users to protect their sensitive medical data and enable better healthcare outcomes by aggregating a more complete set of EMRs for each patient. dBio integrates a FHIR proxy server for interoperability. dBio integrates with Ironcore SDK to give patients control over access to their data using proxy re-encryption. EMR ownership is claimed by users, who can register ownership to the Ethereum blockchain using a Solidity smart contract. Once claimed, dBio stores encrypted EMRs in the distributed, peer to peer IPFS network, breaking data silos and avoiding single point of failure for EMRs.

6. Resources

Source code is available on GitHub under the GNU General Purpose License v3 (GPLv3). GPL follows four guiding principles, including freedom to use the software for any purpose; change the software to suit individual needs; freely share the software; and freely share the changes you make [17]. While GNU grants use rights, the software is copyrighted and all future

versions of the software must have at least as permissive a license as GPL. This prevents future versions of the software from being made proprietary.

Name	Description
dBio Demo	A full system representation in docker-compose format. Start here to get a feeling for what this dApp does!
Web Client	Frontend UI application comprising the dBio patient portal.
Protocol	Protocol control server encompassing numerous API endpoints for interacting with encrypted patient data and facilitating proxy re-encryption of stored resources.
FHIR Proxy	Custom FHIR server implementation for use in healthcare IT infrastructure. Interacts with dbio-protocol for access control and data read/write for patients.
Solidity Smart Contract	Ethereum smart contract supporting Lazy Minting of vouchers for NFTs corresponding to patient data.

Figure 6.1 - dBio Github Repo Names, Links, & Descriptions

7. References

[1] Bender, D., & Sartipi, K. (2013). HL7 FHIR: An Agile and RESTful approach to healthcare information exchange. *Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems*, 326–331.

[2] Department of Health and Human Services. (n.d.). *CMS-9115-F*. Retrieved February 15, 2022, from https://www.cms.gov/files/document/cms-9115-f-interoperability-and-patient-access-final-rule-compiled-faqs.pdf

[3] Usman, M., & Qamar, U. (2020). *Secure Electronic Medical Records Storage and Sharing Using Blockchain Technology*. Procedia Computer Science, 174, 321–327. https://doi.org/10.1016/j.procs.2020.06.093

[4] N. A. Asad, M. T. Elahi, A. A. Hasan and M. A. Yousuf, " *Permission-Based Blockchain with Proof of Authority for Secured Healthcare Data Sharing*," 2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT), 2020, pp. 35-40, doi: 10.1109/ICAICT51780.2020.9333488.

[5] Landi, H., 2019. *Report: 40% of healthcare organizations hit by WannaCry in past 6 months*. [online] Fierce Healthcare. Available at: https://www.fiercehealthcare.com/tech/lingering-impacts-from-wannacry-40-healthcare-organizations-suffered-from-attack-past-6-months

[6] R. Li, T. Song, B. Mei, H. Li, X. Cheng and L. Sun, "Blockchain for Large-Scale Internet of Things Data Storage and Protection," in IEEE Transactions on Services Computing, vol. 12, no. 5, pp. 762-771, 1 Sept.-Oct. 2019, doi: 10.1109/TSC.2018.2853167.

[7] Zhang, P., White, J., Schmidt, D. C., Lenz, G., & Rosenbloom, S. T. (2018). FHIRChain: Applying Blockchain to Securely and Scalably Share Clinical Data. *Computational and Structural Biotechnology Journal*, *16*, 267–278. https://doi.org/10.1016/j.csbj.2018.07.004

[8] Yu H, Sun H, Wu D, Kuo TT. Comparison of Smart Contract Blockchains for Healthcare Applications. AMIA Annu Symp Proc. 2020;2019:1266-1275. Published 2020 Mar 4.

[9] Hoang, H. D., Thi Thu Hien, D., Nhut, T. C., Dang Truc Quyen, P., Duy, P. T., & Pham, V.-H. (2021, December 1). *A Blockchain-based Secured and Privacy-Preserved Personal Healthcare Record Exchange System*. IEEE Xplore. https://doi.org/10.1109/ICMLANT53170.2021.9690542

[10] Roehrs, A., André da Costa, C., Da Rosa Righi, R., Henrique Mayer, A., & Stoffel Antunes, R. (n.d.). *B4HEALTH - AN ARCHITECTURE MODEL FOR PERSONAL HEALTH RECORDS WITH HL7 FHIR AND HYPERLEDGER FABRIC*. International Conference on WWW/Internet 2019.

[11] Chenthara, S., Ahmed, K., Wang, H., Whittaker, F., & Chen, Z. (2020). *Healthchain: A novel framework on privacy preservation of electronic health records using blockchain technology*. *PLOS ONE*, *15*(12), e0243043. https://doi.org/10.1371/journal.pone.0243043

[12] Umbral PRE Threshold Proxy Re-encryption. (n.d.). NuCypher. Retrieved May 8, 2022, from https://www.nucypher.com/proxy-re-encryption

[13] Ekblaw, A., & Azaria, A. (2016). MedRec: Medical Data Management on the Blockchain. Viral Communications. https://viral.media.mit.edu/pub/medrec/release/1a

[14] 1upHealth - connect with electronic health data. (n.d.). 1Up.Health. Retrieved February 18, 2022, from https://1up.health/

[15] *Transform Encryption*. (n.d.). IronCore Labs. Retrieved May 3, 2022, from https://ironcorelabs.com/docs/data-control-platform/concepts/transform-encryption/

[16] *Lazy minting*. (n.d.). NFT School. Retrieved May 3, 2022, from https://nftschool.dev/tutorial/lazy-minting/

[17] *A Quick Guide to GPLv3 - GNU Project - Free Software Foundation*. (n.d.). Https://Www.Gnu.Org/. Retrieved May 3, 2022, from https://www.gnu.org/licenses/quick-guide-gplv3.html

8. Requirements

8.1 Functional Requirements

We worked backwards from our customer to define twelve in-scope functional requirements for dBio. The requirements include a total of 35 acceptance criteria. Overall, we completed 27/35 (77%) acceptance criteria. See detailed acceptance criteria by requirement in Appendix I.

	Acceptance Criteria		
Requirement	**Completed**	**Total**	**% Completed**
Users must be able to register	1	2	50.0%
Once registered, users must automatically connect to the app	5	5	100.0%
Users must be able to upload EMRs	4	5	80.0%
Users must be notified when a new EMR is uploaded on their behalf	0	2	0.0%
Users must be able to claim ownership of their EMRs	2	3	66.7%
Users must be able to view their EMRs	4	4	100.0%
Users must be able to request access to other user's EMRs	2	2	100.0%
Users must be able to view, authorize, and reject EMR access requests	3	3	100.0%
Users must be able to see which 3rd parties have access to their EMRs	2	3	66.7%
Users must be able to modify access to their EMRs	2	2	100.0%
Once access is granted, users must gain access to the relevant EMRs	2	3	66.7%
Users should be able to delete their EMRs	0	1	0.0%
Total	27	35	77.1%

Figure 8.1 - % of Acceptance Criteria Completed by Requirement

8.1.1 Users must be able to register

We met one of two acceptance criteria for user registration. Given that dBio uses OAuth for login, there is no need for a separate registration workflow. Therefore, we consciously chose not to force users to register by providing their public key, phone number, and email address via the dBio UI. However, we do establish an association between the user's email address and their public cryptographic key using OAuth and Tor.us. OAuth allows dBio users to log in to the app using their existing gmail address. Tor.us associates the gmail address with the user's cryptographic key.

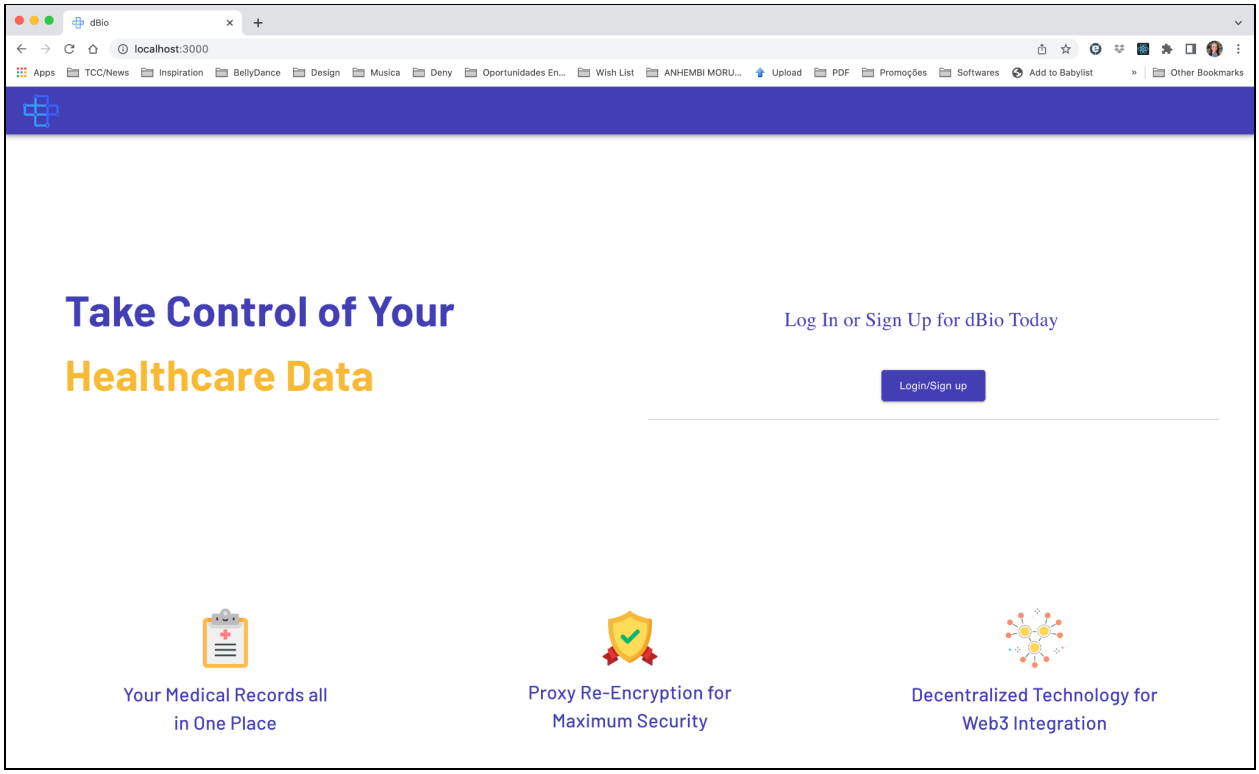

Figure 8.1.1 - Screenshot of dBio landing page before user logs in

8.1.2 Once registered, users must automatically connect to the app

We met all five acceptance criteria for automatically connecting to the app. First, users can login via the dBio UI with OAuth single sign-in. Next, the dBio app associates users with their public key via Tor.us. If a user has previously logged in via OAuth/ Torus with their associated public key, they are automatically logged into the dBio app. If a user has not previously logged into dBio, they are prompted to login. Finally, users can login to their dBio account via the FHIR proxy e.g., to enable medical record uploads.

8.1.3 Users must be able to upload Electronic Medical Records (EMRs)

We met four of five acceptance criteria for uploading EMRs. First, users can upload EMRs to dBio via the FHIR proxy server. Next, the FHIR proxy server calls Ironcore API to create an access control group and encrypt the EMR with the group transfer key before sending it over the internet to the dBio protocol layer. Finally, users can upload updated versions of previously submitted EMRs, and the updated version is saved as a new file.

We deprioritized one acceptance criteria due to development time limitations. Specifically, while updated EMRs are assigned a unique identifier upon upload, they do not include a version number. This makes it more difficult to associate versions of the same EMR and track changes over time. We would prioritize adding this feature in future versions of the dBio system.

8.1.4 Users must be notified when a new EMR is created for them

We deprioritized both acceptance criteria related to notifying users about newly created EMRs

due to development time limitations. Specifically, users do not receive notifications when a new medical record is created for them. We would prioritize adding this feature in future versions of the dBio system.

8.1.5 Users must be able to claim ownership of their EMRs

We met two of three acceptance criteria for claiming ownership of EMRs. Users are able to claim their medical records which are then transferred from escrow and re-encrypted with the user's IronCore private key. Once a user claims a record it is reflected in the user interface by being shown under a "claimed records" table.

Specifically, users can mint a non-fungible token representing each EMR. Users can register ownership of the NFTs on the Ethereum blockchain. However, EMR ownership status is not displayed as an icon in the dBio app. We would prioritize adding this feature in future versions of the dBio system.

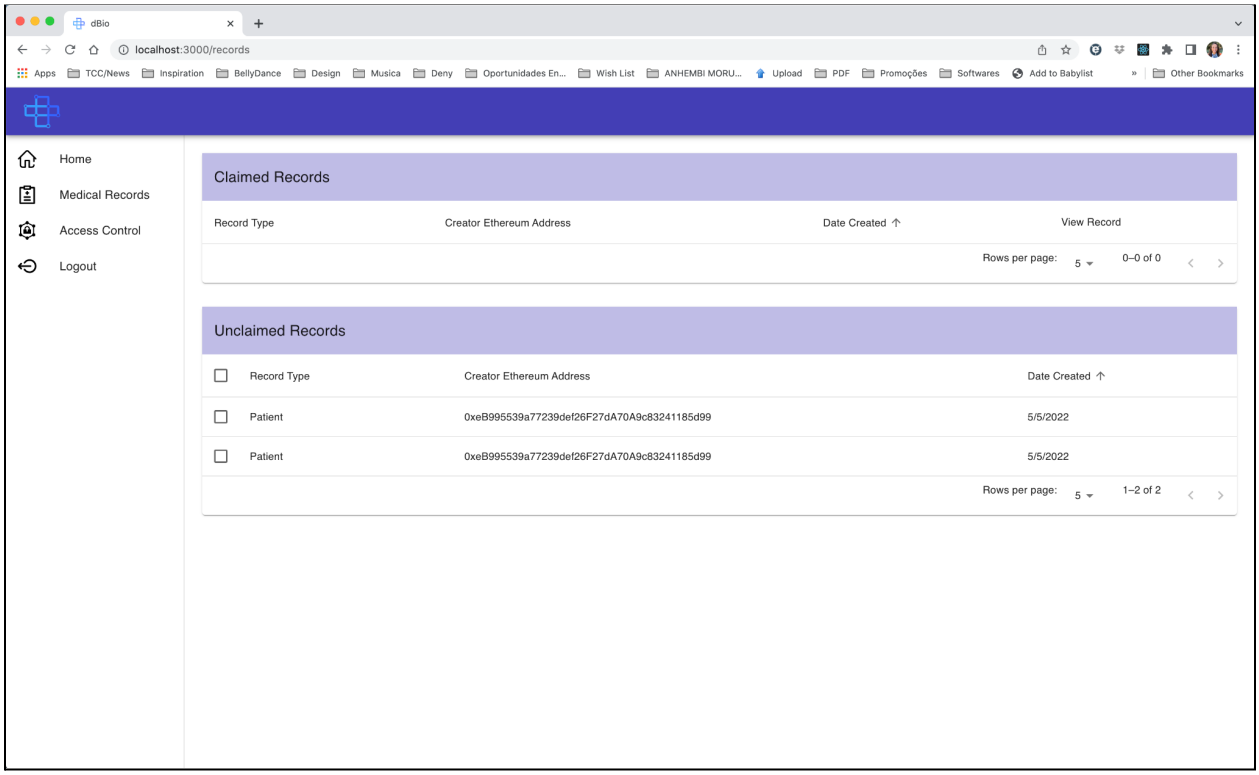

Figure 8.1.5 - Claim Ownership of EMRs

8.1.6 Users must be able to view their EMRs

We met all four acceptance criteria for users viewing their EMRs. First, users can see a table of available medical records in the dBio user interface. Available EMRs include the user's own records and any other EMRs the user has been granted access to. Each row of the table represents a single EMR and includes 1/ Record Type; 2/ Record ID; 3/ a link to view the medical record. Upon clicking the link, the user sees the record rendered in human readable format.

8.1.7 Users must be able to request access to other users' EMRs

We met both acceptance criteria for requesting access to EMRs. First, users can request access to other users' medical records via the dBio UI. Second, access requests include requestor name, requestor ethereum address, and the date the request was created.

8.1.8 Users must be able to view, authorize, and reject access requests for their EMRs

We met three of three acceptance criteria for viewing, authorizing, and rejecting access requests for EMRs. Specifically, users can view open access requests for their medical records in the dBio UI. Open access requests are displayed including requestor name, requestor ethereum address, and the date the request was created. The user can accept or reject access requests.

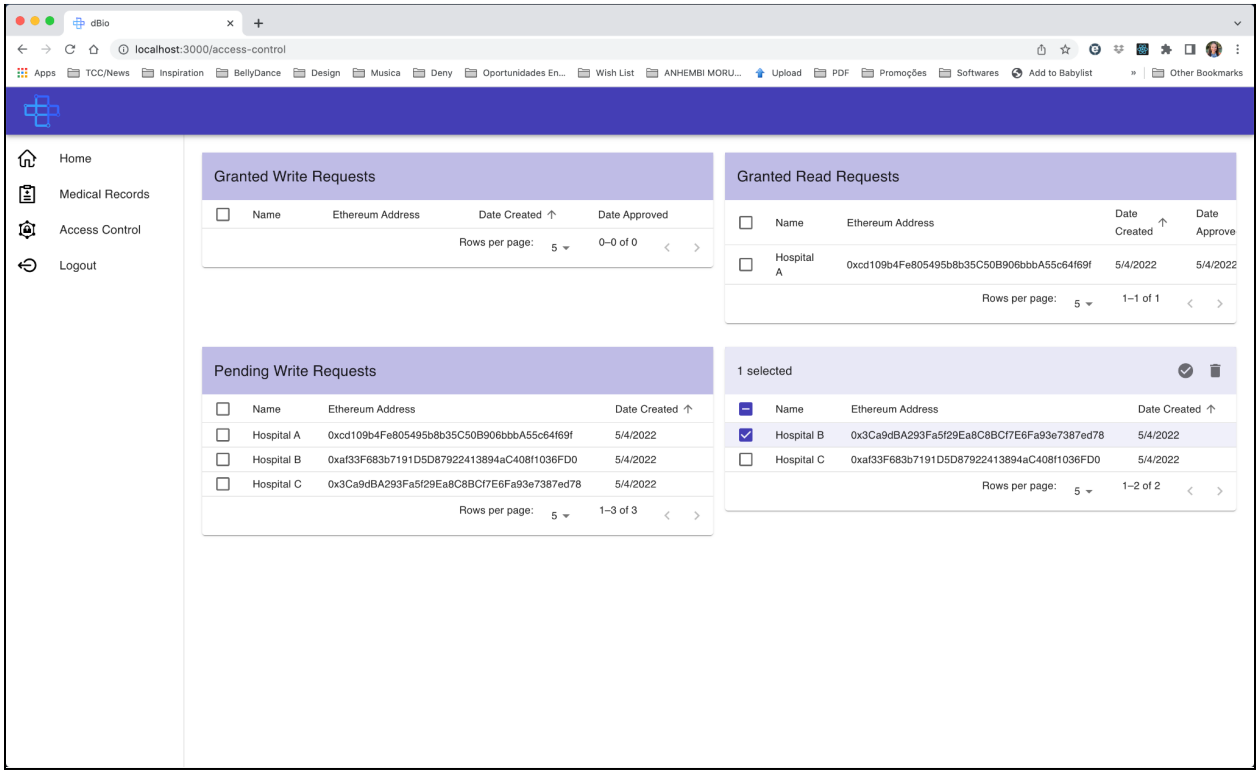

Figure 8.1.8 - View EMR Access Requests

8.1.9 User must be able to see who has been granted access to their EMRs

We met two of three acceptance criteria related to seeing who has access to EMRs. Specifically, users can view a history of access requests for their medical records. The history distinguishes between approved and pending requests, and clearly displays all 3rd parties who currently have access to the users EMRs. However, users do not see rejected requests as they are deleted by dBio.

8.1.10 Users must be able to modify access to their EMRs

We met both acceptance criteria for modifying access to EMRs. Users can reject access for someone who previously was granted access. They can also grant access to someone who

previously was rejected access. However, previously rejected access requests are deleted, so the requestor needs to submit a new access request.

8.1.11 Once access has been granted, Users must gain access to the relevant EMRs

We met two of three acceptance criteria for gaining access to decrypt EMRs. First, if access is granted, users are added to the appropriate Ironcore access control group. They are then able to decrypt and view the (3rd party) medical records.

We deprioritized notifying users via email when they are granted access to EMRs due to development time limitations. We would prioritize adding this feature in future versions of the dBio system.

8.1.12 Users should be able to delete their records

We deprioritized the acceptance criteria that users can "delete" their own medical records by deleting the associated encryption key. We would prioritize adding this feature in future versions of the dBio system.

8.2 Estimates

Before beginning development, we forecast development hours for ten of dBio's twelve in-scope functional requirements. The other two functional requirements were initially deemed out of scope and added at a later stage of the project. Figure 5.2 summarizes forecast vs. actual development hours by requirement. Overall, we underestimated development time by 17.6% (75 hours).

	Acceptance Criteria			
Requirement	**Forecast**	**Actual**	**Delta (#)**	**Delta (%)**
Users must be able to register	25	0	(25)	-100.0%
Once registered, users must automatically connect to the app	25	100	75	300.0%
Users must be able to upload EMRs	75	75	-	0.0%
Users must be notified when a new EMR is uploaded on their behalf	25	0	(25)	-100.0%
Users must be able to claim ownership of their EMRs	25	75	50	200.0%
Users must be able to view their EMRs	50	50	-	0.0%
Users must be able to request access to other user's EMRs	25	75	50	200.0%
Users must be able to view, authorize, and reject EMR access requests	75	75	-	0.0%
Users must be able to see which 3rd parties have access to their EMRs	50	10	(40)	-80.0%
Users must be able to modify access to their EMRs	0	10	10	N/A
Once access is granted, users must gain access to the relevant EMRs	50	20	(30)	-60.0%
Users should be able to delete their EMRs	0	10	10	N/A
Total	425	500	75	17.6%

Figure 8.2 - Forecast & Actual Development Hours by Functional Requirement

We identified three lessons learned that we would implement in future projects to improve forecast accuracy. First, we learned that it is difficult to forecast development time at the functional requirement level. Rather, it is better to break functional requirements down into more granular development tasks, and forecast time to complete each of those. Going forward, we will forecast development time at the more granular task level to improve estimate accuracy.

We learned that it is difficult to estimate development hours for complex tech stacks, particularly

when the tech stack is in flux. When we made our initial development estimates, we were still defining the dBio tech stack and how we would implement the features required to complete the non-functional requirements. System design choices made after estimates were established had a direct impact on actual development time. Going forward, we will forecast development time only after the system architecture is more defined, or focus forecasts on parts of the system that are clearly defined and leave forecasts for other parts of the system for later.

Finally, we learned that it is difficult to estimate development hours for new teams. Established teams have the benefit of using past experience to advise new development hour estimates. For example, they can compare a new task to how long a previous, similar task took to generate an estimate. Further, development time is relative and not all developers work at the same pace. Established teams have a better sense of their pace of development. Going forward, we would be able to establish more accurate estimates given our accumulated experience as a team.

As we advanced in development, we identified that development hours were running higher than expected. Therefore, we consulted with our customer to prioritize the most important requirements. This led to deprioritizing nice-to-have features e.g., managing updates to EMRs and notifying users when a record has been uploaded on their behalf. Of the six acceptance criteria that we did not complete, five were deprioritized and one is unnecessary due to a change in system design.

9. Risks

We grouped risk into three categories: Threat Model, Intentional Risks, and Unavoidable Risks.

9.1 Threat Model

1. Storing users' encrypted protected health information (PHI) on a public medium like IPFS could lead to a malicious party accessing the users' data.
    a. Even the best encryptions could theoretically be broken, and perhaps the likelihood of someone trying to decrypt the data is higher if the encrypted data is public.
    b. More likely than cracking the encryption, users' private key could be stolen and used to access and decrypt the PHI as similar attacks have been attempted to take over users' blockchain wallets.
2. Users can be phished to grant access to their PHI to malicious entities.
3. If any actor transmits PHI in plain text, a malicious party could intercept the message and the patients' PHI could be stolen.

9.2 Intentional Risks

9.2.1 Tradeoffs Between Security and Complexity

The nonfunctional requirements of the system make heavy demands on the overall security of the system. This may lead to more complexity in the design of the system to satisfy both functional requirements as well as security demands. As an example, if the entirety of the storage layer is encrypted, then the relationships between data are not preserved through the encryption. In order to access and perform actions based on these relationships, the entirety of the storage layer would have to be decrypted, which would be an expensive operation. This limits the complexity of

potential search and analytics on data in the storage layer, and creates new problems related to allowing third parties to ask to access data that matches particular criteria without compromising on patients' data privacy.

Mitigation Strategy: We made an intentional choice to prioritize the security and privacy guarantees of the system as opposed to adding on more functionality. For a proof of concept, we prioritized showing that the base functionality of the system can work with the planned decentralized infrastructure while holding true to security and privacy promises.

9.2.2 Nascency of Technology Stack and Integration Concerns

Many components of the selected technology stack are very nascent, due to the recency of decentralized systems' rise to popularity. This poses risks in implementation, as documentation for these technologies can be sparse, and since many of these technologies are still works in progress, some bugs that may arise during implementation may be inherent to the technologies themselves. In order to leverage the benefits of a decentralized system for putting patients in control of their health data, these technologies need to be used, but their nascency exposes platforms built upon them to their associated risks and pitfalls. As a consequence, the integration of these components to work as part of a system may be complicated, with very few points of reference for aid if an error is encountered.

Mitigation Strategy: We worked with our customer and directly with teams that develop SDKs integrated into dBio (e.g., Ironcore) to ensure the feasibility of the integration of components. We continued communicating with these teams during development to ensure that the components of our system work well together reliably.

9.2.3 Blockchain Dilemma: Scalability, Security, & Decentralization

All blockchain based systems face a fundamental tradeoff between scalability, security, and decentralization. For example, the Ethereum blockchain uses a proof of work based consensus algorithm that is highly secure but less scalable in terms of the number of transactions that can be processed. Given this transaction processing bottleneck, gas prices to process Ethereum transactions risk being cost prohibitive. Alternative blockchain technologies such as Polygon prioritize scalability over security, meaning more transactions can be processed at a lower gas price.

Mitigation Strategy: We decided to prioritize minimizing tech stack complexity vs. optimizing for blockchain transaction gas costs. We chose to use Ethereum as our blockchain and Solidity for smart contracts as 1/ our team has basic experience building decentralized apps on Ethereum and Solidity and 2/ Ethereum and Solidity are among the most established Web3 technologies with a strong community to support developers.. For a future production implementation of dBio (out of scope for this project), we may consider moving from Ethereum to another blockchain like Polygon. This would minimize any impact Ethereum gas prices may have on real world adoption of dBio.

9.2.4 Adoption Headwind: Patients must register before 3rd parties can upload their medical docs

Due to patient privacy considerations and the encryption workflow, we will not allow third parties to upload records for patients until the patient registers with dBio. This means that patients will register to use dBio, but initially will not have any records to review. Only after they register for the account can 3rd parties begin to upload their records.

Mitigation Strategy: We believe that more patients would register for dBio if they knew they already had medical documents stored on the dBiodecentralized database and ready to view. However, due to patient privacy and technical details of encryption implementation, we decided that it is not appropriate or possible to store medical records before the patient registers.

9.3 Unavoidable Risks

9.3.1 Adoption by Patients and Third Parties

Medical personnel are already overburdened with many software tools that place a large administrative burden on them; the addition of yet another workflow may not be well received. Additionally, patient understanding and adoption of dBio is required, which poses another unique challenge. This may require patients to understand which parts of their medical data are important to share with a third party and which parts are not necessary to share. The burden of control falls on the patient, so undersharing of data may be cumbersome for interactions between patients and third parties, while oversharing of data would undermine the objective of giving patients more control over their data in an effort to preserve their privacy.

Mitigation Strategy: dBio includes a way for third parties to request data, and in actual clinical interactions, patients can ask clinicians to request data from dBio. This allows for adoption to spread from either the side of healthcare providers and third parties or from the side of patients. Mass adoption of dBio by either group of potential users can heavily pressure the other group to also adopt the dBio system. Additionally, by its nature, dBio is meant to be interoperable with all systems that adhere to FHIR standards.

9.3.2 Security of Messages Between FHIR Endpoints

While the dBio system itself is secure through its use of public key cryptography, FHIR endpoints typically send and receive their messages over the wire as JSON or XML. At the network level, the security of these messages depends on Transport Layer Security, making messages between FHIR endpoints subject to the security considerations of TLS. The dBio system opts to send end-to-end encrypted messages to FHIR endpoints so that even if TLS is circumvented, there is another layer of encryption protecting the data. The FHIR standard does not support ciphertext in request and response bodies.

Mitigation Strategy: We implemented a FHIR proxy endpoint for use in the healthcare provider's network. The endpoint calls Ironcore to securely encrypt EMRs before sending them over the internet. The intent of this is to have third parties run this adapter node on their local networks, minimizing the risk of malicious actors being able to intercept raw FHIR messages. However, this would require third parties to implement the appropriate infrastructure and run this

adapter node on their systems, tying back into the risks associated with the inertia to change of healthcare systems' information technology architectures.

10. System Design

10.1 Workflows

10.1.1 Login

Functional Requirement: Users must connect to the app automatically via a third party login tool
Description: The Login workflow includes three high level steps. First, users login using their existing gmail account. This involves opening the dBio UI, clicking the login button, and choosing the appropriate email address via OAuth. Second, dBio verifies the user email address via OAuth, and retrieves their Ethereum key via Tor.us. Third, the user logs into dBio using their public key and is initialized in the Ironcore SDK.

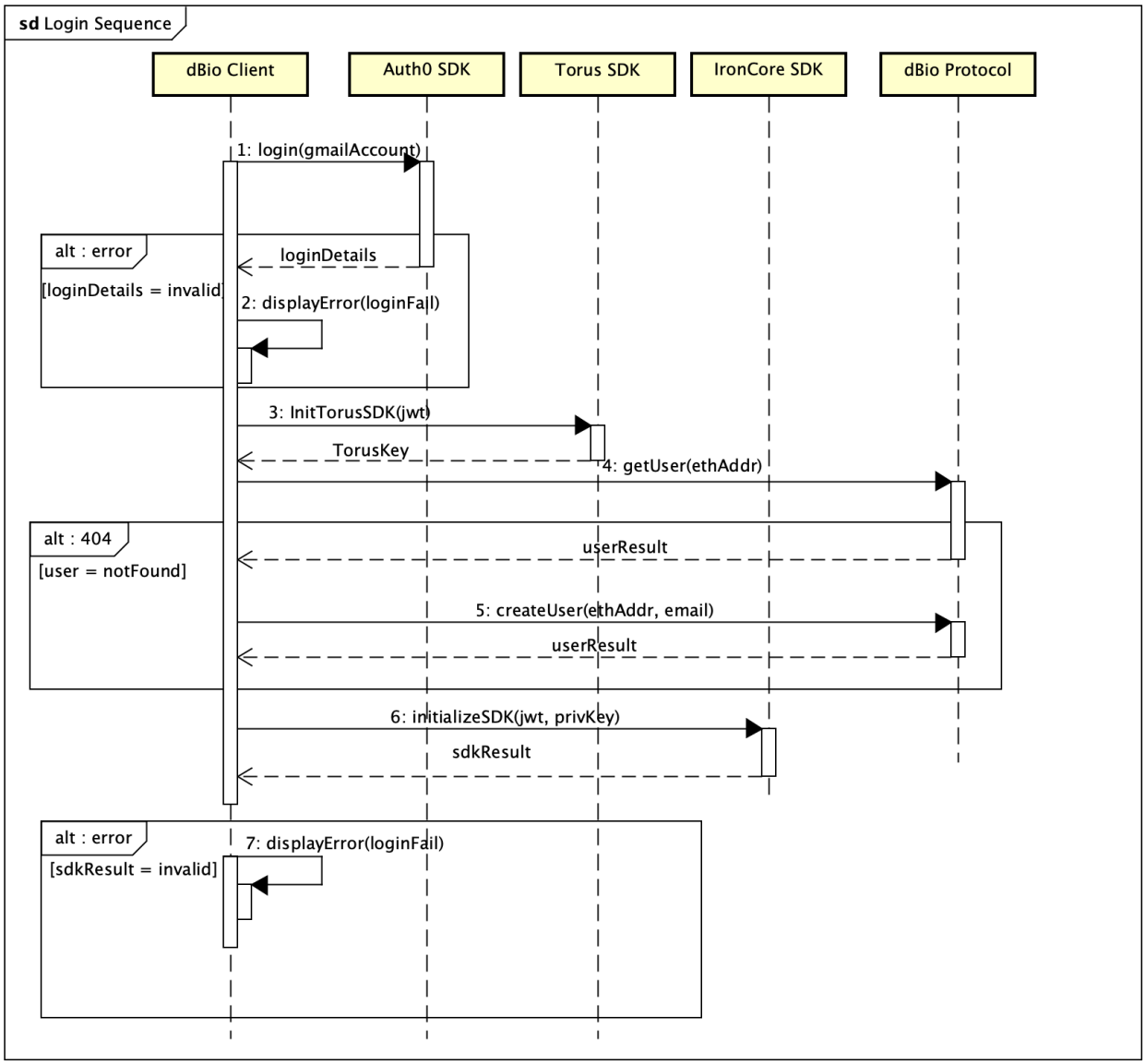

Figure 10.1.1 - Login Sequence Diagram

10.1.2 Upload Record

Functional Requirement: Users must be able to upload medical records
Description: The Upload Resource workflow has three high level steps. First, medical providers create or update a new record for a patient within their Electronic Medical Record system. The patient information includes an email address that is used to link the resource to the patient's dBio user account. Second, the EMR is posted to the FHIR proxy server, which responds by creating an Ironcore access group including the medical provider and patient, using the group transfer key to encrypt the EMR, and sending it to the dBio protocol server. The protocol server stores the encrypted EMR in escrow in the dBio postgreSQL database. The patient can then review and claim ownership of the EMR in the dBio UI.

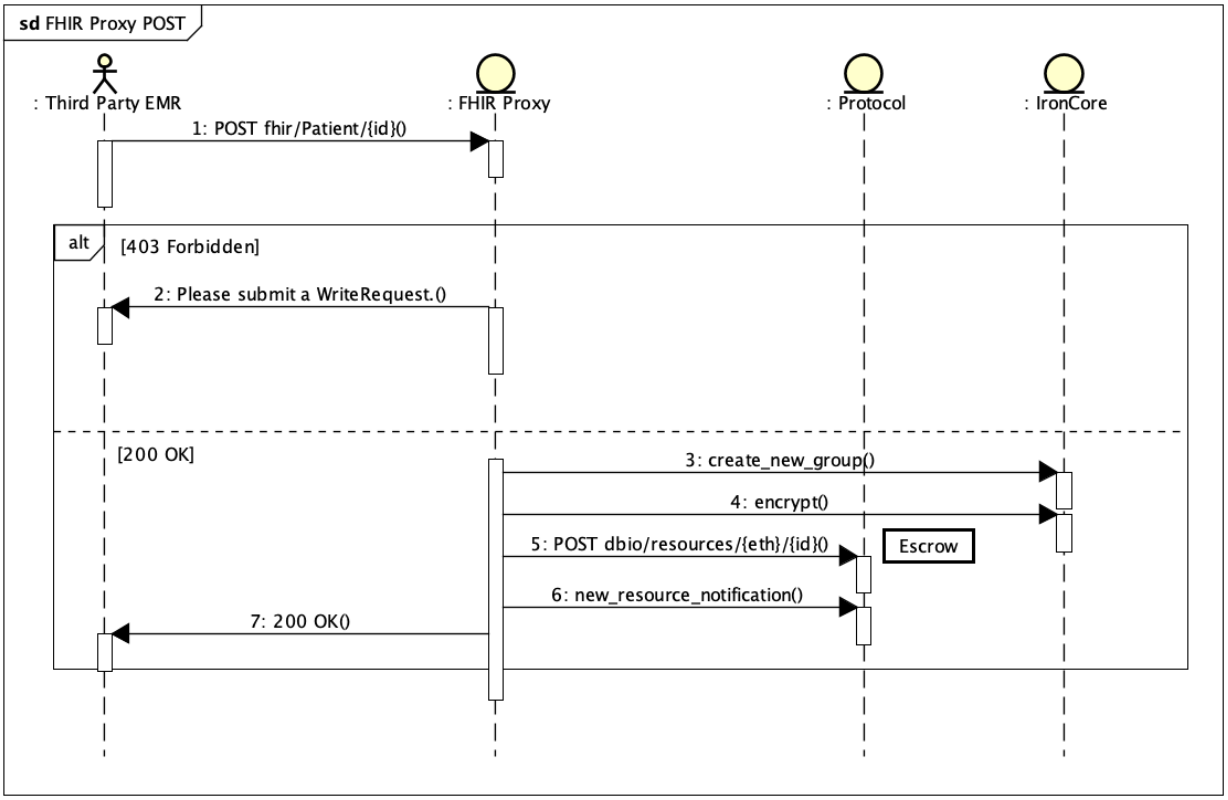

Figure 10.1.2 - Upload Record Sequence Diagram

10.1.3 Access Record

Functional Requirement: Users must be able to view their medical records
Description: Users can access EMRs through the FHIR proxy server and dBio UI, and the workflow includes three high level steps. First, a get request is sent to the protocol layer, which reads the encrypted EMR from IPFS (for claimed EMRs) or PostgreSQL database (for unclaimed EMRs). The protocol layer returns the EMR to the proxy server or UI, which calls the Ironcore API to decrypt it. If the user has permission to view the EMR, it is then decrypted by the proxy server or web client and available for the user to view as plaintext.

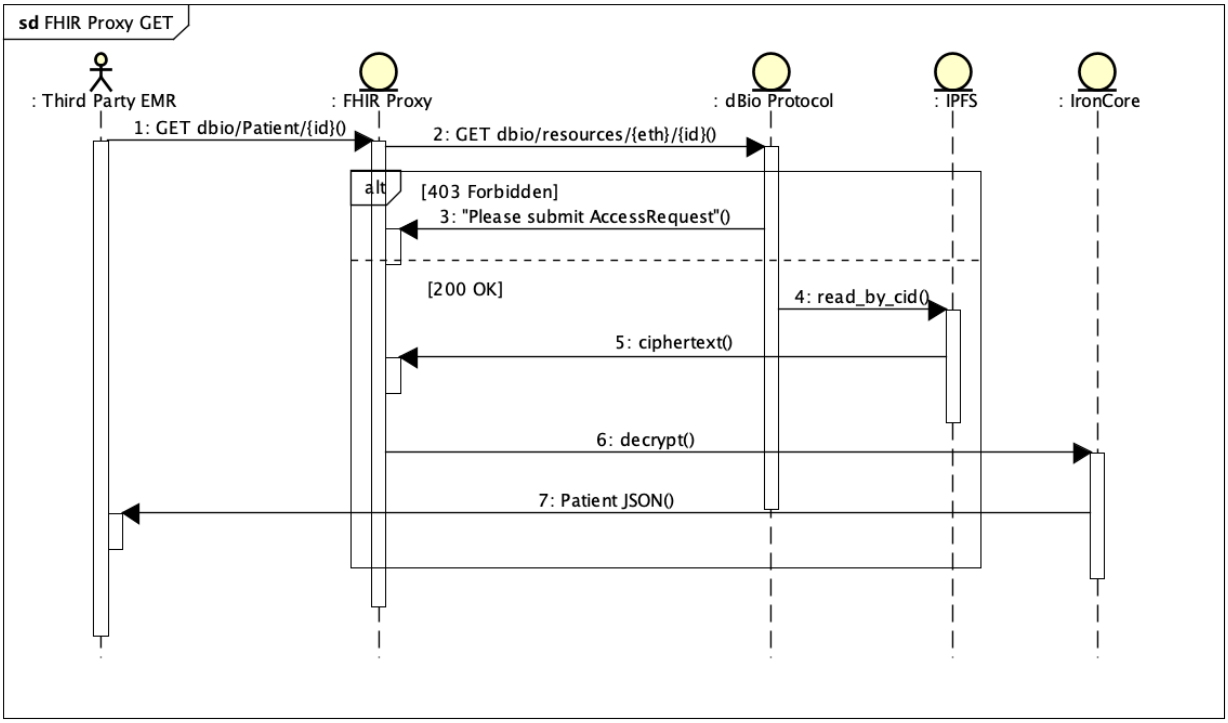

Figure 10.1.3 - Access Record Workflow

10.1.4 Request Access to EMR

Functional Requirement: Users must be able to request access to other user's electronic medical records
Description: Requesting access to EMRs has two high level steps. First, the requestor logs into the dBio app and fills in an access request form identifying whose records they want to access. Second, the request is sent to the owner of the records for their consideration.

10.1.5 Grant Access to EMR

Functional Requirement: Users must be able to view, authorize, and reject third party access requests for their medical records
Description: Granting access to EMRs has two high level steps. First, the user receiving the request must log into the dBio UI and accept the access request. If accepted, the requestor is added to the Ironcore access control group for that EMR, and is able to decrypt and view the record using proxy re-encryption.

10.1.6 Register EMR Ownership

Functional Requirement: Users must be able to claim ownership of their medical records
Description: Registering EMR ownership begins with the user logging into dBio using OAuth/ Torus and selecting the EMRs they would like to claim ownership for. Next, the protocol layer retrieves the selected EMRs from escrow in the PostgreSQL database and sends them to the UI, which calls Ironcore SDK to decrypt them using the access control group transfer key. Ironcore is called again to re-encrypt the EMRs with the user's personal Ethereum key. The re-encrypted EMR is sent to the protocol layer, stored in IPFS, and a voucher is created with instructions to

mint an NFT from the EMR. The protocol also removes the original ciphertext from escrow in the dBio postgres database. Finally, the user can choose to record EMR ownership to the blockchain by paying processing (“gas”) fees to mint the NFT.

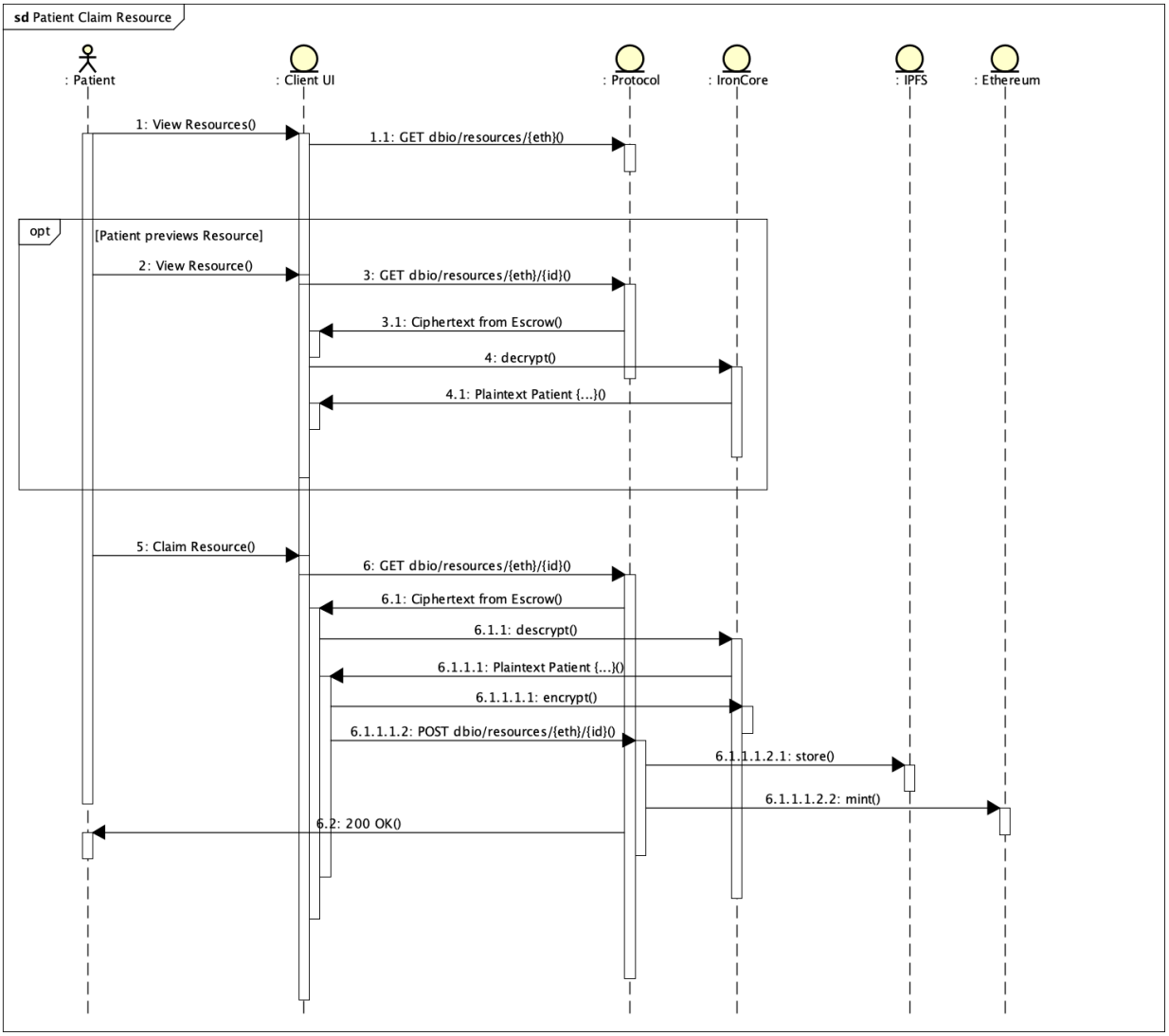

Figure 10.1.6 - Register Ownership of Record Workflow

10.2 Technology Stack

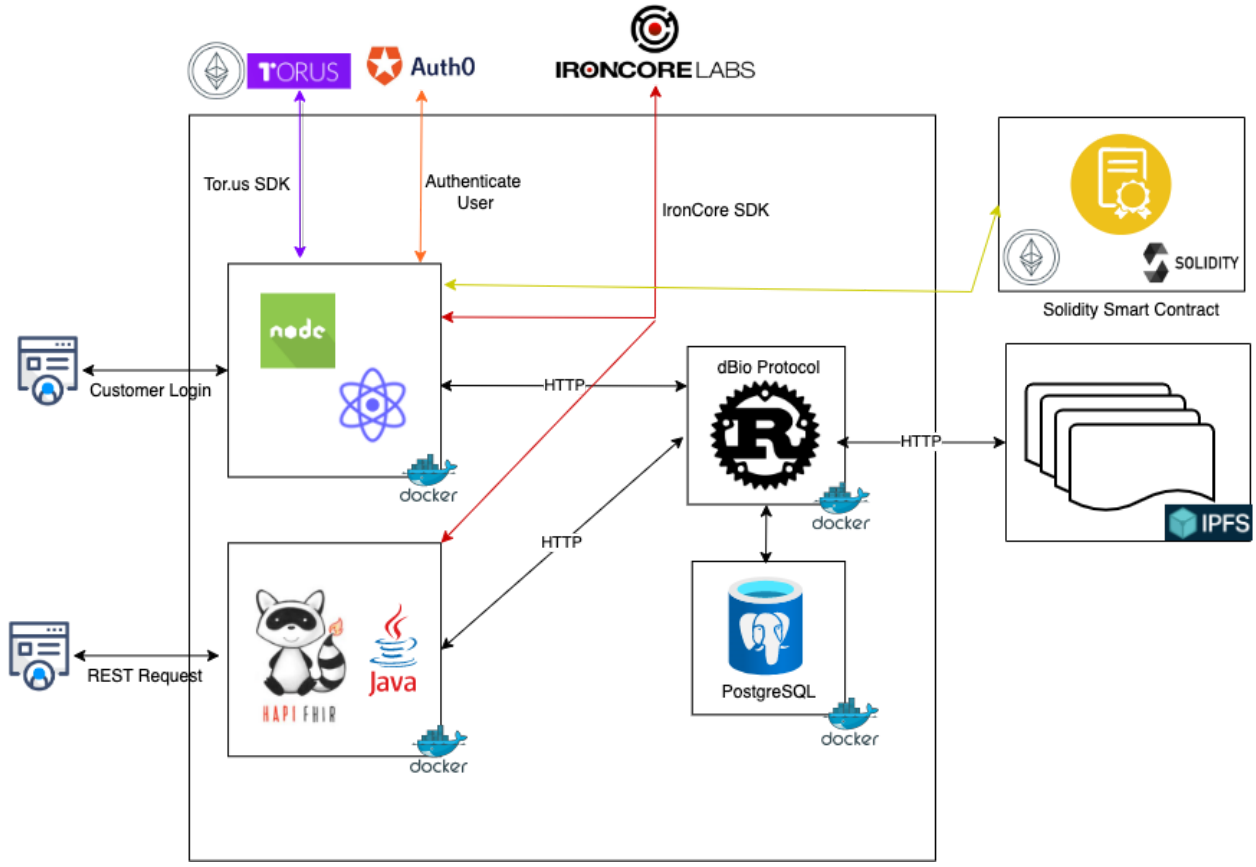

Figure 10.2.1 - dBio Technology Stack

10.2.1 Client Layer

The dBio application has a centralized user interface programmed in React. React was chosen due to its ease to learn for individuals with prior JavaScript experience. dBio leverages OAuth and Torus SDK to integrate passwordless authentication for web, mobile, and decentralized applications. OAuth allows dBio users to log into the app using their existing gmail accounts. Initially, we planned to use Magic SDK for passwordless authentication, but switched to Torus due to integration with OAuth.

dBio leverages the Ironcore Labs Data Control Platform to encrypt and control access to patient EMRs. Specifically, patients can grant and reject access requests for their medical records. For example, if a patient accepts a doctor's access request, the doctor is added to the patients Ironcore access control group. Members of the group are able to decrypt the patient records using proxy re-encryption. The patient can revoke access at any time. Upon revoking access, the doctor is removed from the Ironcore access control group, and no longer can decrypt the patient's records.

The client layer also interacts with Ethereum, an open-source decentralized blockchain which uses a proof-of-work consensus mechanism[2]. The Ethereum blockchain supports smart contracts programmed in the Solidity programming language. An ether.js script will construct transactions to be executed on the Ethereum blockchain when patients claim ownership of a submitted record in the dBio system. Ownership will be staked with a non-fungible token (NFT) that correlates to the user's public key and the record being stored in IPFS.

10.2.2 Protocol Layer

The middle layer of dBio consists of a Rust based API that connects the FHIR proxy server and UI with the data layer. The API covers all of dBio's primary workflows. For uploads, the FHIR server encrypts EMRs using the Ironcore SDK and sends them as ciphertext to the protocol layer. Initially, the protocol layer stores the EMRs in the PostgreSQL database in escrow. Once the patient claims ownership of the EMRs, the protocol layer saves the encrypted EMR to the Interplanetary File System (IPFS) and removes it from escrow in PostgreSQL. Before saving the EMR to IPFS, the EMR is re-encrypted using the patient's public key. At this point the EMR is no longer accessible to the uploader from within dBio unless the patient explicitly grants them access.

10.2.3 Data Layer

The dBio data layer consists of IPFS and a PostgreSQL database. IPFS is a decentralized system for content-addressed data storage hosted on a global decentralized network of nodes. When a file is stored in IPFS an immutable content identifier (CID) is generated to reference it by the hash of its contents dBio will store ciphertext versions of FHIR resources at rest in IPFS. The file can then be accessed from the IPFS network using the CID. The PostgreSQL database manages associations between users, EMRs, and access requests. For example, a Users table will include the Ethereum public key and email address for each user. An EMR table includes information on FHIR resource type, resource ID, creator and owner ID, timestamp, and IPFS CID (if ownership of the EMR has been accepted by the user). An Access Request table includes requestor and requestee ID, request details, and request status.

10.2.4 FHIR Proxy

An adapter proxy will be implemented with the HAPI FHIR toolkit for third parties (healthcare providers, research clinics) to deploy in their local infrastructure. The proxy will "speak" the FHIR standard on the frontend for interoperability with existing systems, but ultimately handle end-to-end encryption and follow the dBio protocol on the backend. End to end encryption begins and ends at the FHIR proxy itself. For example, medical records being sent from a user to the dBio protocol are only encrypted upon reaching the FHIR proxy. Similarly, medical records requested from the dBio protocol server are decrypted at the FHIR proxy. Therefore, the FHIR proxy must sit within the third party's local network as an extra layer of protection in which a ciphertext is sent over TLS rather than a raw JSON object.

[2] https://ethereum.org/en/developers/docs/intro-to-ethereum/

10.3 Discussion

We started with an ambitious tech stack integrating various emerging technologies. Some were ultimately deprioritized e.g., hosting the protocol layer on a decentralized compute platform. The team spent significant time diving deep into related technologies like Fluence and Aqua. In retrospect, this time would have been better served getting a head start on blockchain integration or ensuring FHIR integration for more resource types. Further, certain design decisions were impacted by tech stack ambiguity early in development. For example, we wrote the protocol layer in Rust rather than a higher level language because it compiles to web assembly language, making it a good fit with Fluence. Once development began, we quickly identified a need to simplify our tech stack. Therefore, we removed Fluence and Aqua, and decided to integrate the Ironcore SDK rather than build our own proxy re-encryption scheme.

11. Testing & Tool Suite

11.1 Solidity Smart Contract Testing

We use the Chai assertion library, ethers.js library to interact with the Ethereum blockchain, and Hardhat Ethereum development environment to create unit tests for the Solidity smart contract that mints NFTs and records ownership of them to the blockchain. The unit tests confirm that the smart contract deploys, redeems a single NFT from a voucher, redeems multiple NFTs from a list of multiple vouchers, and fails to mint NFTs when the voucher is not signed by dBio.

11.2 Protocol Testing

We use the Postman API testing platform to test the functionality of our FHIR REST API and protocol server. We test the functionality of all dBio routes using Postman tests.

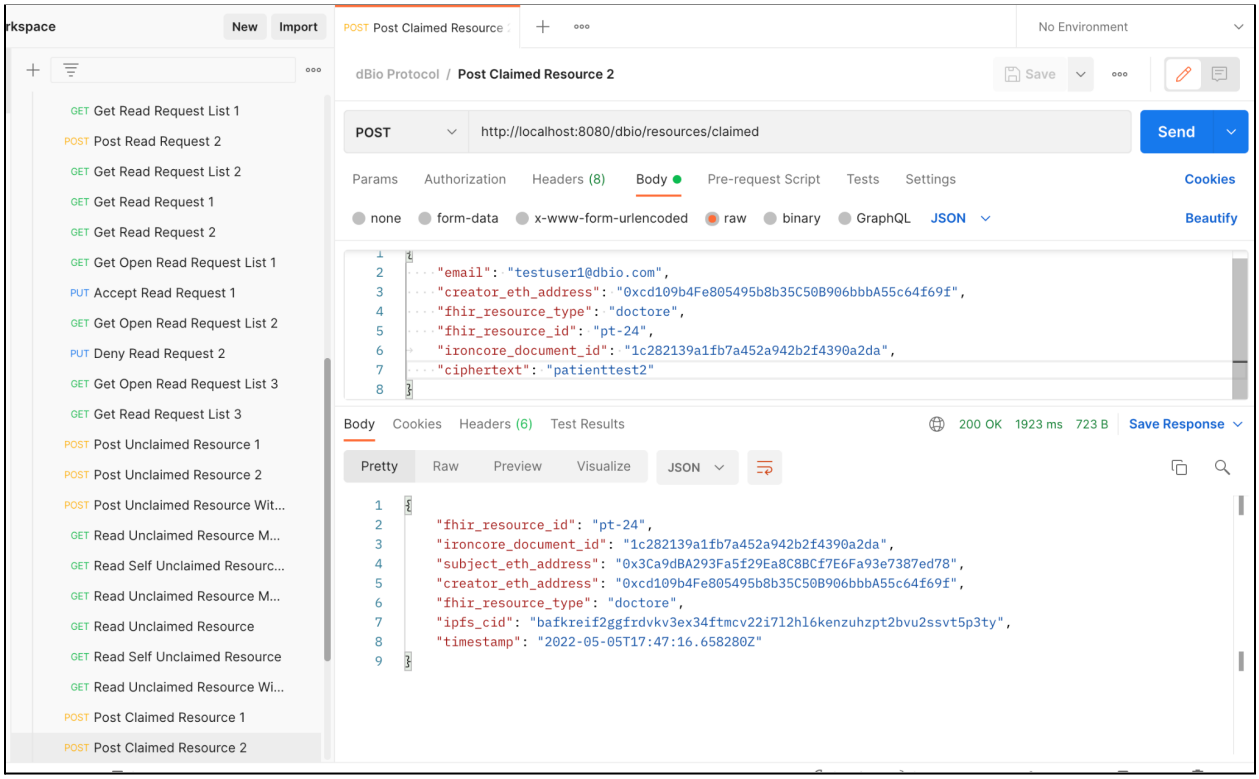

Figure 11.2 - Postman Tests for the dBio Protocol

11.3 Tool Suite

We leverage GitHub as our primary project management tool for issue tracking, code repository and version control, and continuous integration. We use Docker to containerize the components of the dBio system. We leverage Slack as our primary tool for asynchronous communication.

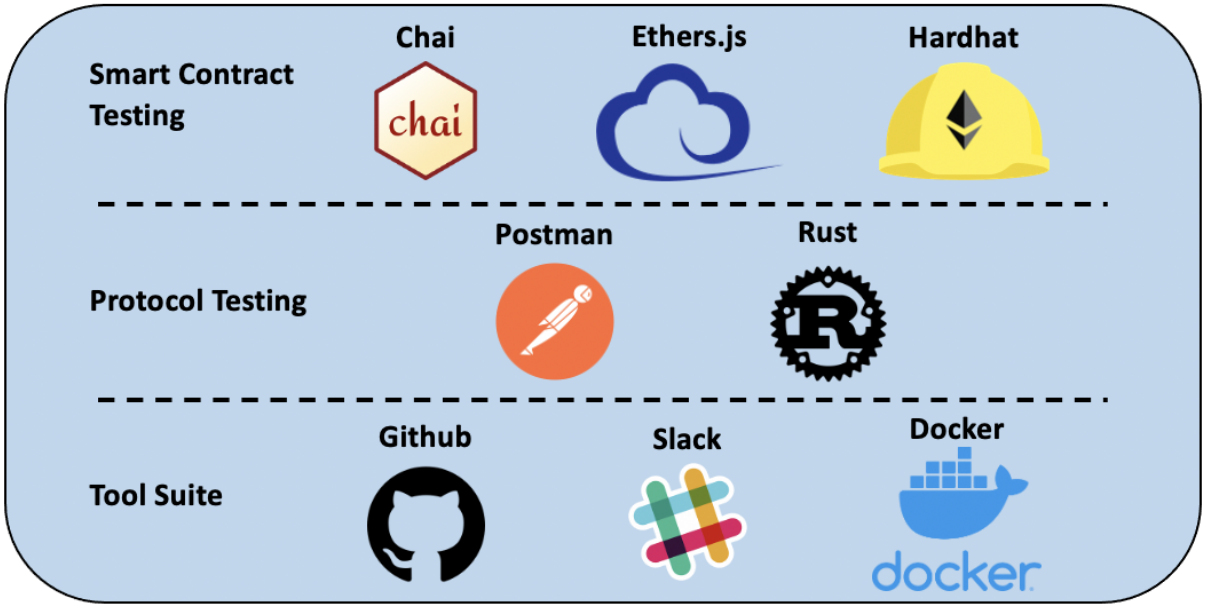

Figure 11.3 - dBio Testing & Tool Suite

12. Development Process & Lessons Learned

12.1 Development Process

We used an agile approach to development and project management. We used slack for daily, asynchronous stand-up updates to inform the team what each team member was working on. During the most busy weeks, we met daily for in person stand ups. We used weekly sprint planning meetings each Monday to define that week's tasks and assign them to team members. We used Github to create, assign, track, and close tasks during sprint planning. Closed tasks were tied to pull requests where applicable. On Wednesdays we met to identify, discuss, and remove blockers. On Sundays we came together with our customer for demo day where team members dove deep into their work from the previous week. The scope and agenda for each meeting became increasingly clear as development progressed, and meeting productivity increased significantly.

12.2 Lessons Learned

We experienced many challenges while building dBio. The challenges can be grouped into two categories: Contribution Gaps and Project Complexity. The range of contribution levels amongst teammates varied. Causes include differences in experience level, knowledge gaps, responsibilities outside of the project, and expectations on time commitment. The contribution gap challenges were exacerbated by project complexity. Project complexity started high due

dependence on nascent technologies with confusing documentation. Complexity increased further due to tech stack changes late into development.

We took several steps to overcome these challenges. First, we reduced the complexity of the tech stack e.g., using a proxy re-encryption service rather than building our own. Second, we appointed a Program Manager and realigned teammates according to skill levels and abilities e.g., teammates with less technical experience focused more on documentation and reports. Finally, we experimented with pair programming so more senior developers could coach more junior developers to transfer knowledge and improve productivity.

Appendix

I. In Scope Functional Requirements

Use Case [*person hours*]	**User Stories/ Acceptance Criteria**
Users must be able to register [*25 person hours*]	● Users can register by providing their public key, phone number, and email address via the dBio UI. [Incomplete - Design Change] ● Upon successful registration, dBio establishes an association between the user's email address and their public cryptographic key. [Complete]
Once registered, Users must be able to connect to the app automatically [*25 person hours*]	● Users can login via the dBio UI with OAuth single sign on. [Complete] ● The dBio app associates Users with their public key via Torus. [Complete] ● If a user is registered with dBio and has logged in via OAuth/ Torus with their associated public key, they are automatically logged into the dBio app. [Complete] ● If a user is not registered with dBio, or is registered with dBio but not logged into OAuth/ Torus with their associated public key, they are prompted to register for dBio or switch accounts. [Complete] ● Users can login to their dBio account via the FHIR proxy e.g., to enable medical record uploads. [Complete]
Users must be able to upload medical records [*75 person hours*]	● New EMRs can be uploaded to dBio via the FHIR proxy server [Complete] ● Updated EMRs can be uploaded to dBio via the FHIR proxy server [Complete] ● When a User uploads a medical record to dBio, the FHIR proxy server encrypts the record and sends it to the dBio FHIR endpoint to be uploaded to IPFS [Complete] ● When an updated record is uploaded to dBio, it is saved as a net new file [Complete]

	• When an updated record is uploaded to dBio, it is assigned a unique, ordinal version number [Incomplete]
Users must be notified when a new medical record is created for them. [*25 person hours*]	• Users receive an email notification when a new medical record is created for them. [Incomplete] • The email includes the Record Type, Record ID, and email address of the User that uploaded the record. [Incomplete]
Users must be able to claim ownership of their medical records [*25 person hours*]	• Users can mint a non-fungible token representing each medical record [Complete] • Users can track ownership of each non-fungible token on the Ethereum blockchain [Complete] • The dBio app UI displays an icon reflecting the ownership status of each medical record [Incomplete]
Users must be able to view their medical records [*50 person hours*]	• Users can see a table of available medical records in the dBio UI [Complete] • Available medical records include 1/ the Users own records and 2/ any other medical records the User has been granted access to [Complete] • Each row of the table represents a single record and includes 1/ Record Type; 2/ Record ID; 3/ a link to view the medical record [Complete] • Upon clicking the link, the User sees the record rendered in human readable format. [Complete]
Users must be able to request access to other users' medical records [*25 person hours*]	• Users can request access to other users' medical records via the dBio UI [Complete] • Access requests include requestor name, requestor ethereum address, and the date the request was created [Complete]
Users must be able to view, authorize, and reject third party access requests for their medical records	• Users can view open access requests for their medical records in the dBio UI. [Complete] • Open access requests are listed in a table including requestor name, requestor ethereum address, and the date

[*75 person hours*]	the request was created [Complete] ● The user can accept or reject access requests. [Complete]
Users must be able to see which third parties have been granted access to their medical records [*50 person hours*]	● Users can view access requests for their EMRs [Complete] ● The UI clearly distinguishes between granted, pending, and rejected requests [Incomplete for rejected requests; complete for granted & pending requests] ● The UI clearly displays all 3rd parties who have access to the Users medical records [Complete]
Users must be able to modify access to their medical records	● Users can reject access for someone who previously was granted access [Complete] ● Users can grant access to someone who previously was rejected access [Complete]
Once access has been granted, Users must gain access to the relevant Resources [*50 person hours*]	● If access is granted, the User is added to the appropriate access control group within Ironcore [Complete] ● If access is granted, the User can decrypt and view the requested (3rd party) medical records [Complete] ● If access is granted, the User receives an email notifying them that the requested medical records are available. [Incomplete]
Users should be able to delete their records	● Users should be able to "delete" their own medical records by deleting the associated encryption key [???]

II. Out of Scope Functional Requirements

Use Case [story points]	**User Stories/ Acceptance Criteria**
Users must be able to view data provenance for their medical records	● Users can view a table of medical providence data ● Medical providence data includes 1/ Resource Type; 2/ Resource ID; & 3/ a log of transactions involving the Resource.

	• The log of transactions details who accessed the resource and when
User identities must be verified by a third party service	• Upon account registration (first login) users are authenticated by a third party service to confirm their identity
Users must be able to associate more than one email address to their account	• Users can associate multiple email addresses with their dBio account • The new email account is verified by sending a code to the new email address and requiring the user to confirm the code.
Users must be able to elect to make their medical records searchable by third party apps	• Users can control whether metadata related to their resources is searchable by third parties • The available metadata includes Resource Type • Users can control searchability settings for individual documents • Users are able to modify their searchability settings e.g., removing searchability for a resource that was previously searchable
Third parties must be able to search health records that are marked as searchable by Users [*25 person hours*]	• Third party Users can search records marked as searchable by their owners • Third party Users can select the desired Resource Type, Patient Age Range, & Patient Sex for the search. • Upon executing the search, third party Users will see a summary of search results including 1/ how many Resources the search returned and 2/ distribution of Resources by Sex & Age.
Users must be able to control access to individual medical records	• Users can control access for individual medical records • If a User with multiple records grants access of a single record to a 3rd party user, the 3rd party User is able to

	decrypt and view the shared resource, but none of the Users other records
Users must be able to review, accept, and reject bids for their medical records	● Users can view open bids for their medical records in the dBio UI. ● Open bids are listed in a table including Requestor ID (i.e., public key or email) and a description of how the Resource will be used ● The user can accept or reject bids. ● The bid will include an access expiration date. ● If the patient decides to revoke access to the Resource(s) before the expiration date, they return the entire value of the bid to the bidder in cryptocurrency. ● Once the expiration date passes, the payment cannot be returned to the bidder for any reason. ● If a User accepts a bid, the Resource(s) will only be made available to the bidder after the crypto payment in the amount of the bid (if any) is confirmed. ● Users can see past bids for their medical records e.g., those that have already been accepted or declined.
Users must be able to bid for access to 3rd party Users' medical records	● Users can bid for access to 3rd party Users' medical records via the dBio UI. ● Users can specify seven parameters for their bids:: 1/ Resource Type; 2/ Age Range; 3/ Sex; 4/ Bid Price; 5/ Expiration Date; 8/ Description of how the data will be used. ● Once a bid is accepted, the bidder is able to decrypt and view the Resources included in the bid until the expiration date or the 3rd party User revokes access

III. Non-functional Requirements

HIPAA Regulations

The system should be compliant with HIPAA regulations, particularly the HIPAA Privacy Rule and HIPAA Security Rule. For all data pertaining to patients that is not de-identified, the system should seek patient approval before sharing that data. For all data stored and transmitted electronically, the system must ensure the integrity, security, confidentiality, and availability of that data.

Security
Security within the platform needs to satisfy the current HIPAA regulations and HL7 Security Services Framework. These include provisions for encrypted data-in-transit and encrypted data-at-rest and authentication mechanisms to ensure that only privileged users can partake in the data transactions between parties.

Encryption at Rest
Patients' PHI data should be signed and encrypted using Decentralized Identifiers (DID) when stored in IPFS. The DID associated with a user should be able to be rotated for re-encryption and also to ensure that a user can delete encryption keys.

PHI should be encrypted using proxy re-encryption, which would allow PHI ciphertext to be transformed without decryption so that it can be decrypted using another user's (or, as in this case, another organization's) private key. The implementation should ensure that the nodes transforming the encrypted data do not have access to the data themselves, which is a key component of proxy re-encryption.

Encryption in Transit
Ensure that data is encrypted using DID as it is being transmitted between different parties and the different layers of the dBio system. The encrypted communication between IPFS nodes satisfies a portion of this requirement but the encryption between the frontend, dApp, the database layer and IPFS should also be encrypted. Data sent between dBio's backend and the FHIR proxy should be encrypted as well using a transfer encryption public key, an improvement over the existing standard.

Authentication & Authorization
User authentication will be a key component of the user workflow since only identified users will be able to use the application. Logging in should be as painless as possible for the user, ideally handled by a third party solution such as MagicLink for browser applications.

Systems like Magic provide a good example of how the initial authentication flow can be handled. Ethereum native APIs can be leveraged but third party projects (Clerk, Moralis) might be leveraged to handle the communication between Metamask and Ethereum to verify the user's identity.

As for authorization, the users' permissions should be limited to the user's own PHI. When a member of an organization (such as a researchers or a doctor) requests access to a user's PHI, that access should be granted via proxy re-encryption where the requester's private key is used to decrypt the PHI and the access can be revoked by simply removing that transformation of data that allows for the requester to access the data.

Performance Requirements

The latency of a write transaction is defined as the amount of time between a user submitting a write request and the write request transaction being written to a block in the blockchain. A write transaction can include a user creating a new medical record or a user updating an existing medical record. The latency of a read transaction is defined as the amount of time between a user submitting a query for EMR data and the query result being returned to the user that made the request. The phrase "most of the time" is defined as at least 95% of the time. The phrase "the rest of the time" is defined as the time when a user is not initiating a transaction with the dBio system, such as when they are navigating to different web pages. The scalability of the system measures how performance and cost are affected in response to an increase or decrease in system processing demands.

Response Time
○ The response time of a read transaction must be a maximum of 2000 milliseconds most of the time. ○ The rest of the time web pages in the dApp must have a maximum response time of 1000 milliseconds most of the time. ○ It must take a maximum time of 1000 milliseconds for a patient to receive a notification of a request to access their electronic medical records. ○ It must take a maximum time of 2000 milliseconds for a user's access permissions to be changed.
Scalability
○ The system must be able to support up to 2000 simultaneous transactions with a maximum response time of 2000 milliseconds. ○ The system must be able to support up to 4000 simultaneous transactions with a maximum response time of 3000 milliseconds. ○ The system must be able to support up to 4000 simultaneous transactions during the time of 9 AM Eastern Standard Time and 8 PM Eastern Standard Time.
Platform
○ Performance testing for blockchain related user transactions must be tested in a non-production environment that doesn't incur expenses (such as Ganache). ○ Performance testing for data related transactions must be tested in a non-production environment to avoid modifying any production data or environment.
Error Rate
○ Write transactions must not fail at least 99.9% of the time. ○ Read transactions must not fail at least 99.9% of the time. ○ Authorization requests must not fail at least 99.9% of the time.

Figure III - Performance Requirements

IV. Team Contract

Meetings

- Team meetings should be recorded so unavailable teammates can catch up on discussions and decisions.
- Meetings should have at least one agenda item, and teammates are encouraged to suggest agenda items on Slack before the meeting occurs.
- Calendar invites with a Zoom link should be sent for all formal team meetings, to all internal and external joinees.
- Teammates should fill out their unavailability in the dbio Team Unavailability calendar.

Documentation

- Documentation must be composed in Google Docs, and teammates should utilize the "suggest edit" and commenting features liberally while collaboratively writing.
- All written segments should be reviewed by at least another team member for grammar, clarity, and formatting.
- Diagrams and media assets should be stored in the team Google Drive folder.
- Formatting standard:
    - **Margin:** 1"
    - **Font:** Times New Roman
    - **Sizes:**
        - Title: 26pt, Bold
        - Header 1: 18pt, Bold
        - Header 2: 14pt, Bold
        - Header 3: 12pt, Italic
        - Normal text: 12pt

Engineering Practices

- The team should aim for >75% unit test coverage to reduce the occurrence of bugs.
- Pair early, pair often. Collaboration is good.
- Teammates should have overlapping knowledge of different systems in the project.
- The *trunk* branch is protected in GitHub. Use pull requests! Pull requests should have at least one approving reviewer who is not the author before merge.
- CI/CD should be enabled for all repositories, running tests and integration tests after building code.
- Practice good code hygiene:
    - Keep functions short and write clean, testable, idiomatic code.
    - Write docstrings to annotate functions and comments to explain what critical pieces of code do.
    - Use a code linter where applicable to keep formatting uniform.
- Try to work in *repeatable environments* by using tools that statically define how software is installed, set up, and run. It helps others get up to speed quickly when switching to collaborate with you. e.g.
    - running a server in a Docker container > running locally on dev machine
    - using a declarative package manager > installing packages manually
    - scripting things > memorizing a chain of commands

V. Team Strategy

Project Management and Software Development Process
Our team is utilizing GitHub's Organization and Project features for project management. We have a dBio organization on GitHub which serves as a central location for code repositories, CI/CD, and task management. We are using the Kanban agile development methodology to manage tasks. We organize our Kanban board into five columns - To Do, In Progress, Review in Progress, Reviewer Approved, and Done. Team members are responsible for creating issues on the Kanban board and moving them to the correct column as they work on them.

We have created a Kanban board under GitHub's Projects feature. GitHub allows us to generate issues to be placed on the Kanban board and associate those issues with GitHub repositories. This level of integration and code management makes it easier for our team to correlate issues with our code repositories.

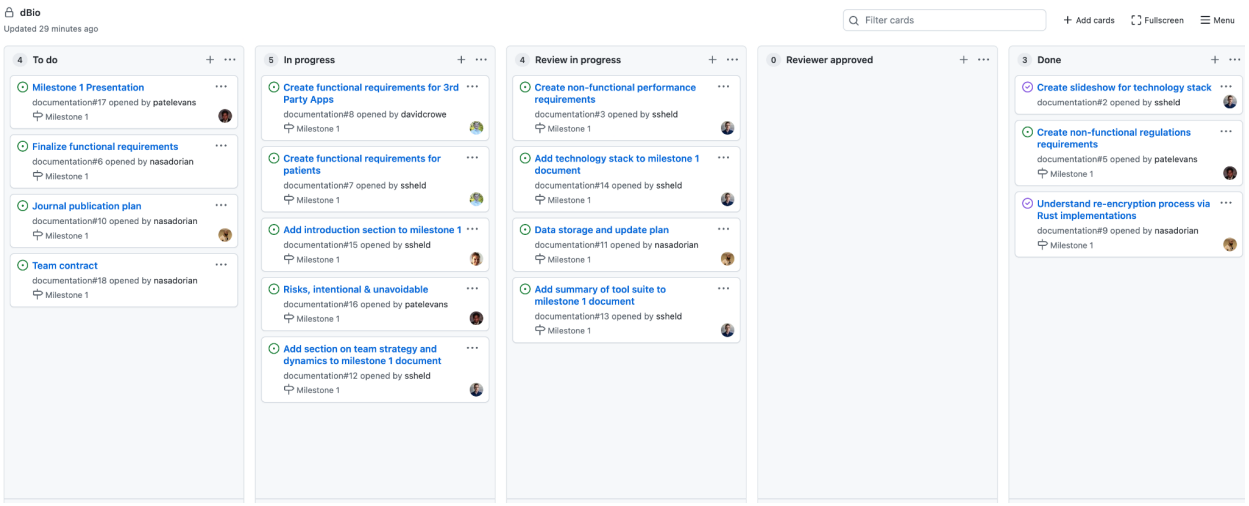

Figure V - dBio Github Kanban Board

We use Google Docs to create any documentation related to the project. Once the documentation has been finished we push it to our Documentation repository.

Meetings and Communication
Currently, our team has three set meetings a week. These meetings occur on Monday, Wednesday and Friday. Recently we have been meeting more frequently to work out the details of our project scope and architecture. Meeting frequency will most likely fluctuate throughout the semester depending on the workload and deadlines. Our customer, Bach Adylbekov, has made himself extremely available and attends all of our meetings if they do not conflict with his schedule.

Our meetings are conducted using Zoom. Meetings are recorded as it is not always possible for everyone to attend due to the high frequency of meetings. After meetings are finished they are uploaded toYouTube as unlisted videos and a link is pushed to a GitHub repository. Meeting videos are typically posted within three hours of a meeting ending.

Decision Making
Decision making occurs during meetings. Our team discusses issues and tasks during these meetings. We try to come to a consensus on decisions. For task assignment we present all tasks that need to be done and then allow members to choose and negotiate which ones they want to pursue. If a member is absent for a meeting in which task assigning occurs then we assign them a task. That member can either take that task or come back to the team and renegotiate to see if another team member would like to either trade tasks or take their task.

VI. Publication Plan

Publication	**Rationale**	**Cost**
IEEE HL7 FHIR DevDays	Largest FHIR-focused conference in the world, aimed at developers and healthcare administrators.	Free
Health Information Management Journal	HIMJ is "a forum for the dissemination of original research and opinions related to the management and communication of health information."	Free
Frontiers in Medical Technology	"*Frontiers in Medical Technology* sets out to provide an open approach where the new advances in [the medical] field can address [patient] challenges in an online community."	$875
International Journal of Information Management	This journal is focused on data management to facilitate "...changes in patterns of behavior of customers, people, and organizations, and information that leads to changes in the way people use information to engage in knowledge focussed activities."	$2310

VII. dBio Developer Installation Manuals
Developer installation instructions are outlined in the README for each dBio components Github repo:

- dBio Demo
- Web Client
- Protocol
- FHIR Proxy
- Solidity Smart Contract

VIII. Glossary of Domain Specific Terms

Aqua
A language for distributed systems where programs are executed on many peers, sequentially or in parallel, forming a single-use coordination network.

Blockchain
Provides immediate, shared and completely transparent information stored on an immutable ledger. A blockchain network can track orders, payments, accounts, production and much more. Blockchain provides a single view of the truth and all transaction details are fully transparent.

Certificate Authority
Provides certificate services to blockchain users e.g., related to user enrollment, transactions invoked on the blockchain, and TLS-secure connections between users or components of the blockchain.

Decentralized app (dApp)
Digital applications or programs that exist and run on a blockchain or peer-to-peer (P2P) network of computers instead of a single computer, leaving them outside the purview and control of a single authority.

Decentralized Identifiers (DID)
Globally unique, resolvable identifiers that have high availability, and are cryptographically verifiable. DIDs are typically associated with cryptographic material, such as public keys, and service endpoints, for establishing secure communication channels. They are useful for any application that benefits from self-administered, cryptographically verifiable identifiers such as personal identifiers, organizational identifiers, and identifiers for Internet of Things scenarios.

Electronic Medical Records
Digital equivalent of paper medical records or charts at a clinician's office. EMRs typically contain general information about a patient such as treatment and medical history as it is collected by the individual medical practice.

Ethereum
A programmable blockchain that replicates and runs programs called smart contracts on all nodes of the network

Fast Healthcare Interoperability Resources (FHIR)
A standard for exchanging healthcare information electronically. The basic building block in FHIR is a Resource (analogous to a medical record).

Fluence
A peer-to-peer application platform which allows the creation of applications free of proprietary cloud providers or centralized APIs. At the core of Fluence is the open-source language Aqua.

Gas

Refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network. Since each Ethereum transaction requires computational resources to execute, each transaction requires a fee. Gas refers to the fee required to conduct a transaction on Ethereum successfully.

HAPI FHIR toolkit
A complete implementation of the HL7 FHIR standard for healthcare interoperability in Java.

Health Insurance Portability and Accountability Act of 1996 (HIPAA)
A federal law that required the creation of national standards to protect sensitive patient health information from being disclosed without the patient's consent or knowledge.

Health Level 7 (HL7) v2 Standard
A framework (and related standards) for the exchange, integration, sharing, and retrieval of electronic health information. These standards define how information is packaged and communicated from one party to another, setting the language, structure and data types required for seamless integration between systems. HL7 standards support clinical practice and the management, delivery, and evaluation of health services, and are recognized as the most commonly used in the world.

Hyperledger Fabric
A distributed ledger platform offering modularity and versatility for a broad set of use cases. The modular architecture accommodates plug and play components, such as consensus, privacy, and membership services.

Interplanetary File System
The InterPlanetary File System (IPFS) is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

Ironcore Data Control Platform
Open-source encryption tools that let developers control access to sensitive data and build minimal trust, zero-trust, end-to-end encrypted applications.

Marine
A general purpose WebAssembly runtime. Fluence nodes use Marine to host the Aqua Virtual Machine and execute hosted Wasm services.

Non-fungible Tokens (NFTs)
Cryptographic assets on a blockchain with unique identification codes and metadata that distinguish them from each other. Unlike cryptocurrencies, they cannot be traded or exchanged at equivalency. This differs from fungible tokens like cryptocurrencies, which are identical to each other and, therefore, can serve as a medium for commercial transactions.

Permissioned blockchain
A distributed ledger that is not publicly accessible. It can only be accessed by users with

permissions. The users can only perform specific actions granted to them by the ledger administrators and are required to identify themselves through certificates or other digital means.

Polygon
A decentralized Ethereum scaling platform that enables developers to build scalable user-friendly dApps with low transaction fees.

Postman
An API platform for building and using APIs.

Protected Health Information (PHI)
Any information in a medical record that can be used to identify an individual, and that was created, used, or disclosed in the course of providing a health care service, such as a diagnosis or treatment.

Proxy re-encryption
A type of public-key encryption that allows a proxy entity to transform or re-encrypt data from one public key to another, without having access to the underlying plaintext or private keys.

Public key cryptography
Involves a pair of keys known as a public key and a private key (a public key pair), which are associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published and the corresponding private key is kept secret. Data that is encrypted with the public key can be decrypted only with the corresponding private key.

React
A JavaScript library for building user interfaces.

Selenium
Automates web browsers e.g., to test the user interface.

Solidity
An object-oriented, high-level language for implementing smart contracts to govern the behavior of accounts within the Ethereum state.

ThreadDB
A multi-party database built on IPFS that combines 1/ a novel use of event sourcing, 2/ Interplanetary Linked Data (IPLD), and 3/ access control to provide a distributed, scalable, and flexible database solution for decentralized applications.

Torus
Non-custodial key management network that splits a user's key between a device, an input, and an existing login. A user can reconstruct their key by combining any of the two.