PEER-TO-PEER PERSONAL HEALTH RECORD

A Thesis

Submitted to the Faculty

of

Purdue University

by

William Connor Horne

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science Electrical and Computer Engineering

August 2019

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Zina Ben Miled, Chair

Department of Electrical and Computer Engineering

Dr. Lauren Christopher

Department of Electrical and Computer Engineering

Dr. Maher Rizkalla

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of the Graduate Program

To Kate and Cullane.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Zina Ben Miled. Her advice and support have been invaluable. I would also like to thank my committee members Dr. Lauren Christopher and Dr. Maher Rizkalla. The previous work by Zachary King was valuable to the success of this thesis. The IUPUI ECE and CS departments are appreciated for helping me develop my knowledge. Special thanks to the Naval Research Laboratory and Dr. Judson Hervey for supporting my graduate education. I would also like to thank all the students who have been my friends for these past two years.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

ABBREVIATIONS xii

GLOSSARY xv

ABSTRACT xvi

1 INTRODUCTION 1

1.1 Health Records 2

1.2 Proposed Framework 3

2 RELATED WORK 5

2.1 Personal Health Record 11

2.2 Peer-to-Peer Networks 14

2.2.1 Distributed Transactions 19

2.2.2 Blockchain 21

2.2.3 Applications of P2P Networks to Health Records 22

2.3 Application Layer Protocol 24

2.4 EHR Datamining 25

2.4.1 Predictive Models and Bayesian Statistics 27

2.4.2 Bayesian Applications to EHR Datamining 28

2.4.3 EHR Datasets 31

3 ARCHITECTURE 33

3.1 Network Actions 36

3.2 Transactions 42

3.2.1 Authorization and Operation 43

3.2.2 Protocol 45

Page

3.3 Hypertension Prediction Service 49
3.3.1 Data Set 50
3.3.2 Preprocessing 54
3.3.3 Algorithm 56
3.3.4 Training and Testing 56
4 IMPLEMENTATION 58
4.1 Index Server 58
4.1.1 Database 58
4.1.2 API and Handlers 61
4.2 Peer Client 62
4.2.1 FHIR Client 71
4.2.2 P2P Subsystem 71
4.2.3 TCP Library 76
4.3 Service Client 79
4.4 Network 80
4.4.1 Registration 80
4.4.2 Login 82
4.4.3 Resource Lookup 84
4.4.4 Posting Documents 85
4.4.5 Requesting and Approving Transactions 86
4.5 Transaction 89
4.5.1 Get 92
4.5.2 Push 93
4.5.3 Service 94
4.6 Hypertension Prediction Service 96
4.6.1 Naive Bayes Model 97
4.6.2 Performance 99
5 CONCLUSION 100

Page

REFERENCES 103
APPENDIX A Hypertension Survey Info 119
APPENDIX B Example Feature Vector 121
APPENDIX C Example Client Usage of System 124
C.1 Provider Requesting a Patient Health Record 130
C.2 Patient Requesting the Hypertension Service 134

LIST OF TABLES

Table Page

3.1 Selected Condition Features 50

3.2 Selected Consolidated Features 53

4.1 Classifier Performance with Panels 17, 18, and 19 99

B.1 Feature Vector 121

LIST OF FIGURES

Figure Page

2.1 Types of Network Architectures 16
2.1 Types of Network Architectures (continued.) 17
2.2 MEPS Panel Design 32
3.1 System Architecture 34
3.2 Client Document 35
3.3 Registration 36
3.4 Heartbeat Process 37
3.5 Search for Network Resources 38
3.6 Fetch FHIR Documents from Portal 39
3.7 Record Registration 40
3.8 Registration and Approval of Service 41
3.9 Network Transaction 42
3.10 Transaction Request Data 43
3.11 Transaction Request and Approval 43
3.12 Transaction Data 44
3.13 Update Data 44
3.14 Get Protocol 46
3.15 Push Protocol 47
3.16 Service Protocol 48
3.17 MEPS Data Preprocessing Pipeline 54
3.18 Feature Vector 56
4.1 Index Server Classes 59
4.2 Database Collection 60
4.3 REST Access 61

Figure Page

4.4 Peer Client Subsystems 62
4.5 Peer Client Collections 63
4.6 Network and WUI Classes 65
4.7 WUI Paths 66
4.8 Registration 66
4.9 Login 66
4.10 Record 67
4.11 Account Page 67
4.12 Network Documents and Transaction Request 69
4.13 Transaction Page 70
4.14 Peer Records Table 70
4.15 FHIR Query 71
4.16 Peer-to-Peer Classes 72
4.17 Job Classes 74
4.18 TCP library 76
4.19 Service Client Subsystems 79
4.20 Transaction Process 89
4.21 Hypertension Prediction Classes 96
C.1 Welcome Page 124
C.2 Patient Registration 125
C.3 Patient Login 125
C.4 Network Page 126
C.5 Portal Selection 126
C.6 Downloading Records from a Registered Portal 127
C.7 Viewing Records 128
C.8 FHIR Data for a Single Record and Posting to Network 129
C.9 Provider: Network Records 130
C.10 Provider: Selecting Patient Record for a Transaction 130

Figure Page

C.11 Provider: Sending Request for Get Transaction 131
C.12 Patient/Provider: Transaction Waiting for Approval 131
C.13 Patient: Approving Transaction 132
C.14 Provider/Patient: Transaction Complete 132
C.15 Provider: Viewing Peer Records 133
C.16 Provider: Transferred Peer Record 133
C.17 Patient Filling out Survey for Service 134
C.18 Survey as a Record 135
C.19 Survey Posted to Network 135
C.20 Selecting Service Transaction 136
C.21 Requesting Service Transaction 136
C.22 Service Transaction Finished 137
C.23 Hypertension Service Result 137

ABBREVIATIONS

ALS	Amyotrophic Lateral Sclerosis
ARHQ	Agency for Healthcare Research and Quality
ASIC	Application-specific Integrated Circuit
BFT	Byzantine Fault Tolerance
BMI	Body Mass Index
BNP	Bayesian Non-Parametric
BSON	Binary JSON
CAD	Computer-aided Diagnosis
CEHRT	Certified Electronic Health Record Technology
CMS	Centers for Medicare and Medicaid Services
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
DHT	Distributed Hash Table
EM	Expectation Maximization
EVM	Ethereum Virtual Machine
FHIR	Fast Healthcare Interopability Resources
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GPI	Generic Product Identifier
GUI	Graphical User Interface
HHS	Health and Human Services
HIE	Health Information Exchange
HIPPA	Health Insurance Portability and Accountability Act

HITECH	Health Information Technology for Economic and Clinical Health Act
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICD-10	International Classification of Diseases Revision 10
ICD-10-CM	International Classification of Diseases Revision 10 Clinical Modification
ICD-9	International Classification of Diseases Revision 9
ICD-9-CM	International Classification of Diseases Revision 9 Clinical Modification
ICU	Intensive Care Unit
IHIE	Indiana Health Information Exchange
IP	Internet Protocol
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
MAP	Maximum a Posteriori
MEPS	Medical Expenditure Panel Survey
MR	Medical Record
NAT	Network Address Transaction
ONC	Office of National Coordinator for Health Information Technology
P2P	Peer-to-Peer
PHR	Personal Health Record
POW	Proof-of-Work
PPV	Positive Predictive Value
REST	Representational State Transfer
RFI	Request for Information
RMSE	Root Mean Squared Error
RTT	Round-Trip-Time
RV	Random Variable

SDA	Stack Denoising Autoencoder
SEER	Surveillance, Epidemiology, and End Results Program
SHA2	Secure Hash Algorithm 2
SHA3	Secure Hash Algorithm 3
SMOTE	Synthetic Minority Over-sampling Technique
SMR	State Machine Replication
SNAT	Symmetric Network Address Translation
SSP	SAS Transport File
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TURN	Traversal Using Relay around NAT
UDP	User Datagram Protocol
ULMS	Unified Medical Language System
VA	Department of Veteran Affairs
WEKA	Waikato Environment for Knowledge Analysis
WUI	Web User Interface
XML	eXtensible Markup Language

GLOSSARY

Begnin fault	a distributed fault such as message loss or a peer crash
Byzantine fault	a distributed fault such as a malicious or faulty peer
posterior	probability of an event given evidence
prior	probability of an event using previous information
consensus	agreement on a single value among distributed peers
features	input data dimensions for a machine learning model
index server	the central server used to progress and ensure transactions
job	a work unit to carry out a transaction
liveness	the distributed system does not halt in progress
panel	a survey period in MEPS
patient	person that is the recipient of healthcare services
peers	a program that is participating in a peer to peer network, acts as both a client and server
provider	person or organization that provides healthcare services
quorum	the number of peers necessary to agree for consensus
record	a health record in the network
requester	the peer requesting a transaction
safety	only correct actions are taken in a distributed network
target	the peer that is the target of the transaction request
third-parties	agent that needs access to the healthcare information

ABSTRACT

Horne, William Connor. M.S.E.C.E., Purdue University, August 2019. Peer-to-Peer Personal Health Record. Major Professor: Zina Ben Miled.

Patients and providers need to exchange medical records. Electronic Health Records and Health Information Exchanges leave a patient's health record fragmented and controlled by the provider. This thesis proposes a Peer-to-Peer Personal Health Record network that can be extended with third-party services. This design enables patient control of health records and the tracing of exchanges. Additionally as a demonstration of the functionality of a potential third-party, a Hypertension Predictor is developed using MEPS data and deployed as a service in the proposed framework.

1. INTRODUCTION

Patients and healthcare providers need to exchange electronic health records. This exchange is necessary for patients to inform providers of their health and for multiple health care providers to collaboratively provide the care needed by each patient. Traditionally, health record exchanges have primarily focused on provider-to-provider exchanges and with a common strategy being the use of a data warehouse called a Heath Information Exchange (HIE). However, these strategies and the associated HIE often do not involve the patient. Exchanges are limited to a given region or target only large institutions. Therefore, patients that seek health services from small health providers or move from one region to another will have a fragmented health record.

To overcome the above issues, this thesis proposes a Peer-to-Peer (P2P) Personal Health Record (PHR) that enables the exchange of patient data between patients and health providers. The system consists of a transaction-based model with patients authorizing access to their records. The network logs each transaction, enabling the trace of accesses to each record. Providers can post new records thereby updating the patient's PHR. Moreover, both patients and providers can access third party network services to perform operations on these records. The advantages of the proposed PHR include:

- patient control of his/her health records
- a common platform for the exchange of health data that can offer a continuum of medical history for each patient
- ease of access to historical data for new health providers added to the system
- ability to keep the health record for each patient up-to-date and accessible in real time to all concerned
- a distributed risk of compromised data

In addition, an efficient representation of the health record will help the patient make informed decisions about their health. In order to act upon the information in the PHR, the proposed system is augmented with a prediction service. The prediction service uses a Bayesian Network to predict a patient's risk of developing Hypertension based on a submitted health record. This predictor serves as an example of the types of services that can enhance the utility of the proposed PHR.

1.1 Health Records

A patient's electronic health record documents the health of a patient and their interactions with healthcare providers. Though implementations vary, the FHIR ontology [11] contains a generalization of the information captured in health records. It details roles, workflows, and financial information that documents the patient's interaction with a provider. The provider records the observations, symptoms, and conditions for each patient. Diagnoses in FHIR are standardized according to various ontologies including ICD-10 [165], the International Classification of Diseases used for medical diagnosis. In summary, the FHIR ontology includes definitions for:

1. Individuals and their roles in the system (e.g., patient, provider, related, etc.)
2. Entities such as organizations, locations, or devices
3. Workflows such as tasks or appointments
4. Management of the encounter between a patient and a health provider
5. Clinical information such as observations, conditions, and medications
6. Financial information including billing

Several initiatives have attempted to overcome the fragmentation of patient health records across multiple institutions. One such initiative is the provider's use of a Health Information Exchange (HIE) [59]. HIE are data warehouses that aggregate health records from different providers. Providers can then pull the aggregated records from the HIE to update their internal information. However, while HIE collect information from large providers, they are often unable to reach out to smaller providers.

Moreover, HIE are generally limited to a certain region and unable to aggregate health records from outside their hub of providers.

In 1996, the United States government passed HIPPA, which mandated a patient's right to access their own health record [156]. To fulfill this requirement, some providers have created online portals for patients to view their records. Patients can access their records but the service is ineffective. Providers control and define the representation of the records limiting the service's usefulness to the patient [92, 170]. Records from multiple portals are often incompatible with each other, making it hard to build a comprehensive view of a patient's health. An alternative to this approach of provider controlled medical records is a Personal Health Record.

A Personal Health Record (PHR) [109] is an inversion of the provider-centric approach to health records. The patient stores and maintains the health record and the provider contacts the patient for record access. This patient-centric approach engages the patient and allows him or her to choose services from providers. Moreover, the PHR provides a platform for customized services such as appointment notifications, research related to a patient's specific conditions, treatment options, and a connection to social networks for information sharing [50].

Researchers have estimated that a move to PHRs in the United States will result in a benefit of $11-19 billion annually [74, 144]. However, despite this economic benefit, PHR's have experienced low uptake. This is attributed to a lack of patient awareness about PHRs, provider misgivings, and issues in interoperability [88, 147].

1.2 Proposed Framework

This thesis proposes a Peer-to-Peer (P2P) network for the aggregation and exchange of patient health records. The network consists of:

- A P2P network of users that is maintained by an index server
- A client that the patient or provider can use to interact with the network
- A protocol for exchanging records between the members of the health network

- A callable network service that implements a Bayesian model to predict a patient's risk for Hypertension, exemplifying how services can be added to the proposed platform

The proposed PHR system avoids the central governance and the overhead associated with maintaining an HIE. Indeed, in the proposed system, the provider can directly contact the patient. Moreover, the patient has control over his or her data and can selectively authorize access to this data. Finally, by leveraging the underlying P2P communication network, third party providers can offer additional health services to the patient such as the prediction of a patient's risk to develop a condition from their health record.

The remainder of this thesis is organized as follows. The second chapter summarizes the current state of health record systems, peer-to-peer architectures, distributed transactions, EHR datamining, and Bayesian approaches. The third chapter describes the design of the proposed system, including system architecture, client features, and the dataset and algorithm used for predicting a patient's risk of developing Hypertension. Chapter four presents the system implementation. Chapter five concludes with an analysis of the proposed system, its limitations, and potential extensions.

2. RELATED WORK

Patients, providers, and third parties are roles in the healthcare system. A patient is an individual who is seeking, receiving, or received medical care from a provider. A healthcare provider is the direct giver of care, usually a hospital, doctor's practice, or medical institution. This exchange of service to improve the patient's health is healthcare. A third party is an agent that needs access to information on the exchange of care between the patient and the provider but is not a direct beneficiary. Common third parties include, but are not limited to: referred specialist, researchers, government agencies, insurers, and family of the patient. The interactions between the patient and the provider are documented in the patient's health record.

The stakeholders in healthcare, the patients, providers, and third-parties, have differing requirements that a health record needs to fulfill. A patient needs unrestricted access to their health record. Indeed, a patient may need to transfer this information to a new provider. A provider has similar requirements with a greater emphasis on managing the patient's EHR, interacting with various payment outlets, providing a legal record, and tracking resources. Third party requirements are more diverse. For instance, a diagnostic service might need partial access to lab results and payment information in order to perform the service on behalf of the provider. A research group or government organization may want to analyze and track statistics for a population of patients. A patient's descendants may want to preserve a family member's record to document medical history. Thus, an EHR must balance and flexibly handle each party's needs.

According to [120], a health record contains the following information:

- Administrative and billing data
- Patient demographics

- Progress notes
- Vital signs
- Medical history
- Diagnosis
- Medications
- Immunizations
- Allergies
- Radiology images
- Lab and test results

The patient medical information in EHRs conforms to various standards. The current diagnosis standard is the World Health Organization's ICD-10 medical classification list [165]. This specification standardizes the naming of diseases, symptoms, and medical terms. Older records use the earlier ICD-9 standard [164]. Gaps in the standard have caused countries, medical groups, and EHR software developers to create extensions such ICD-10-CM [27]. Other organizations have developed ontology's such as SNOMED-CT [154] and the Unified Medical Language System (ULMS) [86, 157], to encode the semantic meaning of the medical terminology. Some encodings are proprietary such as the Generic Product Identifier (GPI) [162], which codes prescription drugs and groups.

Hospitals and care providers have been digitizing health records for the past three decades [16]. The resulting electronic health records (EHR) consist of the patient's history, encounters with providers, and the patient medical record (MR) (e.g., procedure, lab reports, prescriptions, etc.). Several countries require providers to use EHRs [16] or heavily incentivize their adoption [16, 45]. In the United States, EHRs are certified by the Centers for Medicare and Medicaid Services (CMS) and Office of the National Coordinator for Health Information Technology (ONC) through the Certified Electronic Health Record Technology (CEHRT) certification [58,119]. Moreover, CMS and ONC promote the adoption of a certified EHR through Medicare and Medicaid funding programs. EHR developers have created various electronic health-

care solutions, often tailored to individual providers. Example commercial EHR systems include: EPIC [43], Cerner [29], and Meditech [98], which together comprise 66% of the market [149]. These systems adopt different representations of electronic health records. Therefore, transferring records between these EHRs requires using protocols such as HL7 [63] or the more recent FHIR [11]. To support these protocols, vendors must implement a mapping of the EHR's internal representation to the protocol's representation. Developing this mapping may be costly and tedious [52].

The expense and vendor lock-in associated with commercial EHR's has created interest in open EHR standards. The openEHR foundation proposed an open specification of templates and archetypes to implement an EHR, rather than purchase a proprietary solution [124]. Several implementations have been deployed [123], but none in the United States. Another standard is the ISO-13606, Health informatics - Electronic health record communication [69]. A benefit of these open specifications is the ease of development of novel solutions that are customized to providers or specific fields [37]. However, these standards leave implementation details unanswered, often leading to wildly varying solutions [49]. Moreover, the implementation of these standards requires software expertise which is tangential to the core business of healthcare providers.

To facilitate the exchange of records between providers, States and providers have created Health Information Exchanges (HIE) [59]. HIE are third party brokers that aggregate health records for various regional providers, handle the conversion of the various records to a common data representation, and transfer records to requesting third parties. HIE fulfill three types of transfer [161]:

- Direct Exchange: Securely send and receive patient information between providers.
- Query Exchange: Find information related to an unknown patient.
- Consumer Mediated Exchange: Intake records directly from the patient.

The ONC has incentivized the development of HIEs with grants from the Health Information Technology for Economic and Clinical Health act (HITECH) [2,59]. As of 2017, 69% of hospitals had agreements with an HIE [71] but this number is misleading. The coverage is patchwork, with access to only members of the HIE network resulting in providers needing multiple HIE agreements to service patients. Furthermore, it is difficult to transfer records across multiple HIEs and the capabilities of the exchanges are often inadequate, making it difficult for providers to integrate HIEs with their local EHR [117]. In Indiana, the Indiana Health Information Exchange (IHIE) [68] encompasses most of the major providers in the state but has yet to include rural health providers. In Illinois, established regional exchanges prevented the success of a statewide exchange [139]. At the national level, the ONC has sponsored the creation of eHealth Exchange [42] to facilitate interstate transfers. As of 2019, 75% of US hospitals and 59 HIEs had agreed to participate in eHealth Exchange. When these mechanisms are absent, the patient needs to facilitate the transfer of records between providers.

The ONC has expressed interest in making EHRs interoperable [118]. CMS has created the Promoting Interoperability Program to incentivize providers [28]. Several EHR vendors and HIE have agreed to increase interoperability under Carequality [24], an effort and standard to ease the difficulty of transfer and conversion. HL7 proposed the Fast Healthcare Interoperability Resources (FHIR) [11,18] as a standard to transfer records between providers. FHIR has two major components, a generalized ontology to represent the various aspects of an EHR and a Representational State Transfer (REST) protocol to exchange the information. Application developers can target the standard to create services based on the ontology and interact with servers that adhere to the REST representation. HL7 has continued development of services using FHIR under the Argonaut project [64]. One service is SMART [95], a patient approved access framework for third parties using OAuth2 [36]. Carequality has recently supported FHIR [150] and it is likely that FHIR will become the industry standard for EHR transfer.

Public sector development has focused on enhancing the Blue Button program. Blue Button is a CMS and US Department of Veterans Affairs (VA) initiative that started in 2010 [104] to enable patients to download their records. Blue Button supports access to Tricare, the VA, and Medicare but has also garnered support from the private sector. Over the years, CMS has updated Blue Button several times to reflect new EHR developments. Originally, downloaded records were text files. An update called Blue Button Plus, includes an option to export the record as HL7 CDA XML and supports online push/pull requests [20, 104]. A supporting Department of Health and Human Services (HHS) project called Blue Button Connector [1] provided an online website to help patients search for accessible records. The connector later grew to include a service lookup component for developers. The connector was discontinued in May 2017. The development of FHIR generated a 2015 HHS pilot project to support the standard [61, 142]. CMS continued this work and released Blue Button 2.0 in 2018 [26]. Blue Button 2.0 supports FHIR v3.0.0 and allows developers to register clients for the service. Approved clients can access Tricare, VA, and Medicare Part A, B, and D records.

All the above initiatives are institution-centric and in support of provider record ownership. The ownership and control of Health Records is controversial. A 2014 study [22, 140, 152] of primary care patients who had been given granular control of their records found that 43% of patients enabled some restriction on their record. Primary care providers could override patient wishes, with 10 overrides of restricted records out of 126 overrides. In surveying the patients, 93% agreed with restricting part of their record and 95% with restricting persons from accessing the record. The corresponding responses for providers were lower at 54% and 42%, respectively. Follow-up responses supported this patient/provider disparity.

In the United States, patients are guaranteed access to their records under HIPAA [156]. To fulfill this requirement, hospitals have begun offering online portals, sometimes called tethered PHR, for patients to extract and view their records [77]. In a 2017 review of patient portals [170], the United States Government Accountabil-

ity Office (GAO) interviewed providers who reported that 91-92% of their patients were offered electronic access to their health information. However, these numbers are not supported by other studies. For example, Lyes [92] reported that 33% to 45% of patients recalled being directly offered this access. Other studies in 2016 and 2018 [13, 127] reported offer access rates of 34 to 60.3% [1]. Peacock and Anthony [13, 127] found that uptake was heavily influenced by the provider's explicit offer and that both men and minorities were offered access at lower rates despite both groups expressing the same level of interest as their peers. The GAO also interviewed patients about their portal experiences and found that patients were frustrated by incomplete or incorrect records and poor user interfaces. Patients expressed interest in aggregating records from multiple portals and viewing their longitudinal records.

A portal has limited usefullness to a patient and is oriented towards fulfilling regulatory requirements. Indeed, the provider still controls and defines the representation of the health records. Portal records downloaded from multiple providers often have incompatible formats, preventing the aggregation of these records. If a portal is not in place, it can take months to access this health data. Access can also be costly to the patient, and is often non-compliant with the patient's wishes [92].

[1]The 60.3% access offers includes access offers from insurers.

2.1 Personal Health Record

A Personal Health Record (PHR) makes the patient, rather than the provider, the custodian of his or her health information. A PHR stores and manages the health records from various providers, creating a single access point for the patient. Though the definition of a PHR has been historically ill-defined [116], a CMS Request for Information (RFI) suggested that a PHR has the following attributes [109]:

- owned and controlled by the patient
- contains lifetime health information
- stores health information from all healthcare providers for each patient
- accessible from any place at any time
- private and secure
- transparent, where the patient can track access to his or her health data by others
- facilitate the exchange of health information between health providers

A PHR resolves several issues inherent to provider-centric approaches. A single access point to a patient creates a hub for providers. This resolves the conformance issues between various EHRs and provides a communication bus between providers. The provider solutions of HIE and EHR systems consist of distributed architectures making it difficult to achieve a single consistent view of the patient's records. Moreover, EHR and HIE have difficulties in of correctly aggregating records from external sources as patient records lack a primary key [44]. To correctly identify patient records, providers must resort to heuristic and probabilistic methods with error rates around 15-38% [44]. A PHR eliminates this issue by making the PHR the master view of the patient's health records.

For patients, a PHR can provide access to curated health information, enable the easy tracking of chronic conditions, and facilitate communicate with providers [147]. Services and home instruments can connect and update the health record. Moreover, the records, transfers, and communications between various providers is transparent

to the patient. The patient can control access to their records, dictate the terms of access, and trace accesses by providers. This can provide a patient evidence of HIPAA violations. For caregivers, a PHR provides a framework to help manage the health of the patient [147].

Providers and third parties can benefit from PHRs. PHRs can help providers engage with patients to improve outcomes, transfer health records between different EHR systems, and get real time feedback from patients [147]. It also provides a common access point, eliminating the need to rely on HIE [147]. Moreover, PHR are theorized to provide economic benefit. A 2008 economic projection study [74, 144] of PHR adoption in the United States found the economic benefits depend on the PHR used. Portal solutions with each provider implementing their own and with no interoperability were estimated to be an annual net negative of $29 billion with an initial cost of $130 billion. Third party and interoperable PHRs were projected to provide $11-19 billion in benefit with initial cost between $3.7-21 billion. The study's authors noted that third party PHRs were highly scalable and that interopablilty was key to reducing cost and providing benefit.

There are several challenges to PHR adoption: PHRs are difficult to implement and deploy [88,145,147]; Patients are often unaware of existing PHR systems and may be concerned that their records would be insecure [88, 145, 147]; Not all patients want or need a proactive role in their health management [115, 145]; and Providers have concerns about workflow integration, patient interpretation, PHR validity, liability, and incentive [88, 115, 147]. In addition, independent PHRs limit institutional control of health data which is counter to the business model of these institutions [147]. The bearer of cost is also unresolved [147]. Recently, several high-profile PHR systems failed including Google Health [89], Microsoft Health Vault [101], and Dossia [40]. All three of these programs were targeted at the sponsoring company's employees in hopes of decreasing the company's healthcare cost. In light of these concerns and issues, it is important that both patients and providers see immediate benefits to

using a PHR. PHR and PHR-like systems are beneficial to patients with long-term conditions but have had issues with cost and accessibility [145].

CMS conducted a Personal Health Record Pilot called MYPHRSC for Medicare enrollees in South Carolina from 2009-2011 [115]. The PHR consisted of a web-based application from HealthTrio that provided users summaries of health information, medications, claims, calendars, and more. An outreach program contacted 100,000 Medicare beneficiaries in South Carolina and attempted to generate interest with demonstrations and adverts. The study conducted follow up surveys and interviews with beneficiaries. Overall, 4,114 beneficiaries registered and 3,041 used the PHR more than once. Pilot users expressed interest in the improved communication with providers, health summary information, and the printable wallet card with information on health, medications, and emergency contact. Even though the study focused on patients with chronic conditions, these users were not more likely to use the PHR. The lack of uptake among this group might have been caused by the short time period of the pilot and the lack of results tracking. Other cited issues were inaccurate claims data, usability issues, and lack of integration with beneficiaries' workflows.

MyHealtheVet (MVH) [153] is a tethered PHR system operated by the VA which was launched in 2003 and is still in operation. Veterans can refill prescriptions, look at health records, search for health information, message the VA, and more. A 2011 usability survey of MVH [56] reported that patients were interested in the prescription refill feature and in printing their health information. However, there was negative feedback in regards to the search function and usability. MVH added the Blue Button feature in 2010. MyHealtheVet is not a true PHR. Patients must manually share their information downloaded through Blue Button with a third party. The third party is then responsible for correctly extracting the information.

Third-party applications have started to integrate with the updated Blue Button service. One application is iBlueButton, a PHR by Humetrix [65]. It is a mobile-based PHR that allows a patient to access and locally store their health records from government Blue Button providers. The application allows the patient to make

notes and check for drug conflicts. The application is limited as it strictly targets government Blue Button projects. Additionally, the application can only pull records and can't post updates back to the provider.

Patient's Like Me is an online social network for patients with chronic conditions [126]. Originally the program targeted patients with Amyotrophic lateral sclerosis, ALS, but has since grown to encompass more conditions. Users can share information on conditions and treatments [50,160]. However, users do not control the shared information and the information is subject to research datamining to fund the program. Several services can be added to a PHR in order to enhance its benefits to the patients. For example, a service that enables the prediction of risk factors for chronic disease. An example of such a service for Hypertension is demonstrated in this thesis. Another service that might be integrated into a PHR is facilitated provider-patient communication. For example, providers can monitor medication adherence and disease conditions. A 2011 study found that testing and screening reminders helped increase patient medication adherence [166]. A commercial application based on this idea is care.coach [23]. In care.coach, a provider can communicate with patients through an online 3D avatar thereby limiting the need for doctor's visits.

2.2 Peer-to-Peer Networks

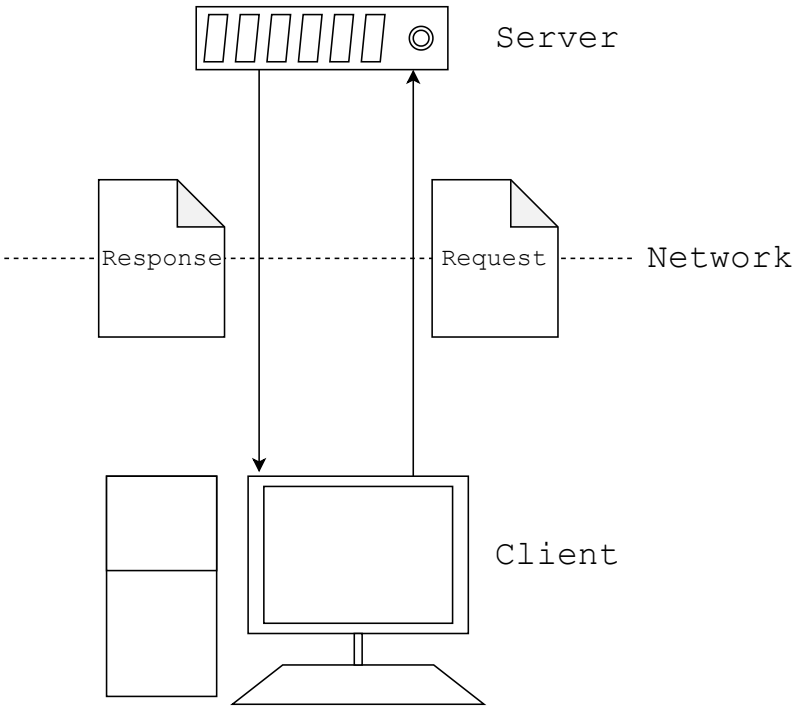
There are two different common types of network communication, Client-Server and Peer-to-Peer (P2P) (Fig. 2.1). In a Client-Server architecture (Fig. 2.1(a)), clients connect to a central server [78]. The central server controls access to information and clients exchange information by way of the central server. A Peer-to-Peer network architecture has clients called *peers*, who communicate directly with each other [78]. These peers must operate as both client and server in the network.

Network design and topology play key roles in the design of a P2P architecture. There are several models: *centralized*, *pure*, and *hybrid*. These designs have advantages and disadvantages.

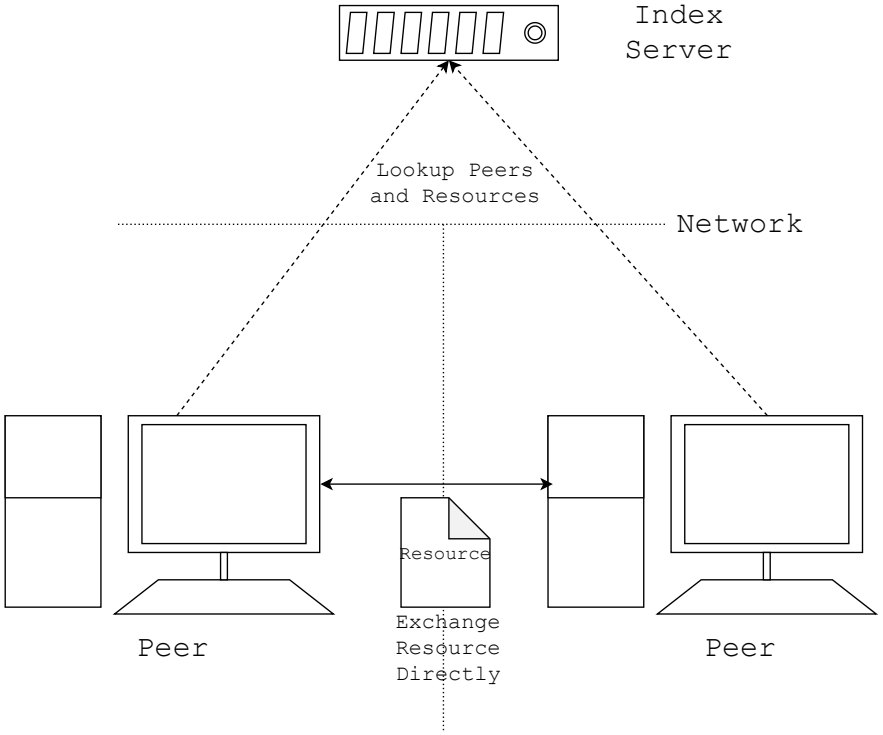
A centralized P2P model (Fig. 2.1(b)) uses an index server (also called a central server) to provide resource lookup for the network [67]. Clients search the index server to find peer resources and then contact the target peer in order to exchange data directly. Benefits of this system are ease of deployment and fast resource lookup. Drawbacks include a reliance on the index sever and limited scalability.

A pure P2P network (Fig. 2.1(c)) has no index server and clients must query other peers to discover resources [67]. To search for a resource, the client either floods the network with a query [67] or accesses a distributed hash table (DHT) [67, 78] maintained by the peers which lists the resources and owner. The primary benefit of a pure P2P system is an equal relationship among the peers, with peers sharing costs and resources fairly. However, drawbacks include slow lookup and high network complexity. Pure P2P networks such as Guntella [134] had issues with network traffic until the development of DHTs.

The hybrid P2P model (Fig. 2.1(d)) selects some peers in the network to act as *supernodes* [67]. Other peers connect to the network through these supernodes. Supernodes route peer messages to other supernodes who then deliver the messages. This design has a faster lookup than a pure model and is more resilient than the centralized model. However, the hybrid P2P model is difficult to implement. Skype prior to 2016 [151] used a hybrid P2P model [67].

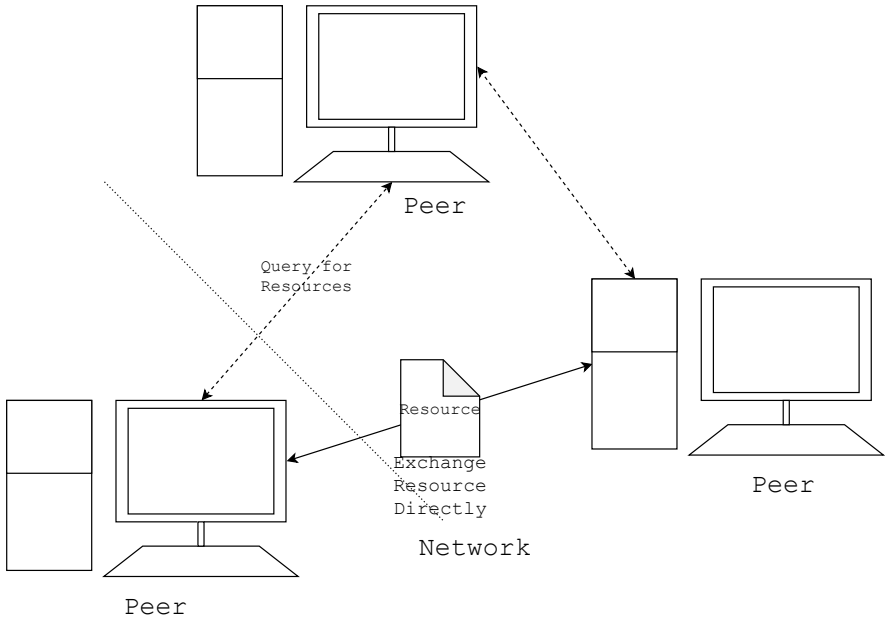


(a) Client-Server

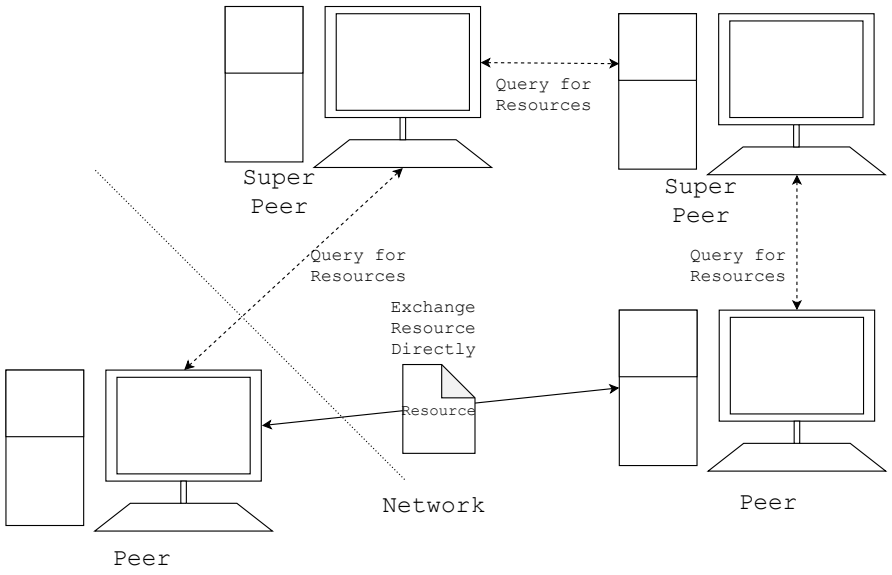


(b) Centralized Peer-to-Peer

Fig. 2.1.: Types of Network Architectures



(c) Pure Peer-to-Peer



(d) Hybrid Peer-to-Peer

Fig. 2.1.: Types of Network Architectures (continued.)

There are three stages to P2P communication: accessing the network, connecting to peers, and searching for a resource. Each stage has challenges that the network design must overcome.

During the first stage when accessing the network, the peer has no initial knowledge of the network topology. A standard method to locate other peers is a signaling server. A signaling server has a predetermined location and holds a list of networked peers and their IPs. The client application queries this server for networked peers and attempts to connect to one of the peers using the peer's IP. With IPV4, port forwarding, and Symmetric Network Address Translation (SNAT), it is not always possible to connect directly to another peer. TCP Hole punching [146] and STUN/TURN protocols [94] can overcome these issues. The communication between the peers is usually a custom application protocol built on top of TCP or UDP. Once connected to the network, the peer can search for resources in the network. The lifecycle of a search varies according to the network topology. In pure and hybrid P2P models, the query can flood the network or access the DHT maintained by peers. In a centralized model, the client sends the query to the index server. The index server has a list of resources in the network and can direct the client to the owner. The client can then connect to the target peer and exchange the resource directly or by routing the data through the network.

P2P networks have several advantages over Client-Server architectures. P2P networks are scalable [78]. Compute, network, and storage cost are shared by the peers and not concentrated in a single server. Lastly, P2P networks are resilient against network failure [67]. Despite interest in P2P networks, the architecture has several issues that limit its adoption. SNAT requires a third-party server to facilitate routing of data between peers [94]. Moreover, *peer-churn* [78], where peers enter and leave the network, can lead to network degradation when too few peers are available. To counteract this, peers must be properly incentivized to remain in the network. Lastly, P2P file-sharing networks such as BitTorrent are heavy used to illegally share copyrighted works, which limits business support of P2P models [67].

2.2.1 Distributed Transactions

In a distributed network, transactions among peers takes the form of *consensus*. A distributed environment is unreliable, creating the need for a protocol that solves consensus in a bounded time [2]. The performance of a distributed system is measured by its *safety* and *liveness* [7]. Safety means that an incorrect action does not occur, and liveness refers to the system's ability to progress in state. Fischer, Lynch, and Paterson [48] proved that there is no protocol that satisfies both. The focus changes to providing a best effort protocol. Another issue with distributed systems is the reliability of the individual parts. Processes can lie or fail, an error called a *Byzantine* fault [3]. A Byzantine fault can be malicious or as a result of a failure to follow the protocol. In contrast, a *Benign* fault is a unintentional crash or message loss [25]. Ignoring the unreliability of the connection, Byzantine Fault Tolerance is possible but only if more than $\frac{2}{3}$ of the peers are working correctly.

The simplest solution to consensus is to have a central server manage the state of the system [93]. The server is the global view of the system and acts as the synchronization between clients. Clients replicate the state of the server upon connecting and post updates to the server. The server checks the updates against its own internal copy and broadcast state updates periodically to the clients. This architecture is used in distributed simulations [93, 113]. This design provides some benefits. Incorrect actions are not possible, and it does not assume client trust. Moreover, client failure does not block progress of the server's state and thus the network. However, there are limitations with this centralized architecture. For instance, clients may need to repeat actions if they are out of date. Clients can keep up with the state of the system at the cost of a Round Trip Time (RTT) [78]. The central authority is a point of failure in the network and the design suffers from limited scalability. Modified P2P versions can remove this issue at the cost of safety [113].

[3]Consensus is commonly illustrated as the allegory of the two-general problem [55].
[3]This is in reference to the allegory of the Byzantine General Problem [84].

The Two-Phase-Commit [55] protocol is a commonly used protocol for consensus in which clients post to a central authority.

- The authority asks if the clients want to commit
- Clients post their messages and wait
- Authority verifies the transaction
- Authority broadcasts the commit to the Clients
- Clients acknowledge the broadcast

This protocol is safe in most scenarios but is reliant on the central authority. If the authority fails the clients will never release their resources. The Three-Phase-Commit protocol builds on this protocol by adding a timeout to make the protocol nonblocking but with a loss of safety.

The Paxos algorithm [82] is a complex algorithm that guarantees safety and good liveness. The protocol uses periods called rounds, each with a ballot ID and 3 phases. On each round, a process or multiple processes can act as the leader. Each leader queries the other processes for their ballot ID. If the proposed leader has the highest ID then they are the leader for the round. The next phase has the new leader push a new value to the group. The group responds to the leader. If the leader receives a majority of the responses, called a *quorum*, it broadcasts to the group that the round is complete. Otherwise, the leader moves onto the next round. Paxos is noted for its cost and complexity to implement and variations have been proposed [83, 122]. Further it can only protect against *Benign* faults. A program that implements a variation of Paxos is Apache Zookeeper [66].

State Machine Replication (SMR) [81,138] can provide protection against Byzantine Faults. A state machine is a computation model in which a machine executes actions to transform its current state. In SMR, each client runs a replica of the same state machine executing the same actions. When a byzantine failure occurs, as long as more than 3 healthy replicas of the faulted process exists [84], the system is fault tolerant. However, a naive implementation of SMR can fail to protect against Byzantine faults [25]. A Byzantine Fault Tolerant system needs a replication algorithm with

proper creation, coordination, and verification. The first implementation of such a system was the Practical Byzantine Fault Tolerant (BFT) algorithm [25].

2.2.2 Blockchain

Blockchain is another architecture designed to solve Byzantine Faults. Unlike BFT replication protocols, it achieves its resiliency though hashing. Blockchain uses strong cryptography and Proof-of-Work (POW) [15] to provide an append only ledger [99] consisting of *blocks*. The proof-of-work prevents Byzantine faults by making network peers find cryptography hashes before committing transactions. These blocks are linked together by these hashes, making rewrites of the transaction tree increasingly difficult. Blockchains differ from BFT replication protocols. Though they both provide protection against byzantine faults, Blockchains are open to new users while BFT systems need a prior verification. Blockchains are also more resistant to Sybil attacks, where an attacker directs a large number of hidden nodes to disrupt the system. High-profile Blockchain applications include the cryptocurrencies Bitcoin [111] and Ethereum [163].

Bitcoin [111] is a digital currency that uses a Blockchain to maintain a distributed record of transactions and owners of *bitcoins*. The goal of the application is to provide a decentralized, secure, and anonymous method for exchanging assets. Each owner is identified by a public key. The coin's value is tied to the market's demand. Units of bitcoins can be transferred from one client to another by issuing a transaction to the network. The owner signs the transaction request with their private key. The system verifies the owner using the public key. Transaction requests are pooled together into blocks to be verified and appended to the ledger. Nodes in the network called *miners* verify a block by finding a hash of variable strength, a proof-of-work. After a hash is found, the block is added to the chain and the miner is rewarded a bitcoin for the proof-of-work. Bitcoin implements a stack-based language to support dynamic

transactions [85]. To prevent transactions from not completing, the stack language does not implement loops and is not Turing Complete [4].

Ethereum [163] is another cryptocurrency aimed at providing verified electronic contracts called smart contracts. It is similar in design to Bitcoin but has several key differences. One of Bitcoin's issues is the hashing race between miners. Over time miners have upgraded from CPUs to GPUs to ASICs in order to increase their hashing output. This has contracted the market of miners, putting the network at-risk. To combat this issue, Ethereum's block hashing algorithm was changed from SHA2 to an ASIC-resistant version of SHA3 [10]. To implement electronic contracts, Ethereum implements the Ethereum Virtual Machine (EVM) in place of Bitcoin's stack language. EVM is Turing Complete, allowing arbitrary programs to execute on the network. To prevent programs from running indefinitely, each program is charged the cost of its execution steps. The verification of valid EVM programs has been hard to implement in practice [19].

Blockchain systems are difficult to evaluate. Cryptocurrencies such as Bitcoin and Ethereum have noted issues with high-transaction cost [79], illegal activities [31], and market-volatility [79, 129]. Furthermore, the word Blockchain is used to market alternative products. Hyperledger [148] is a series of industry-sponsored projects that are marketed as private blockchains. However, most of these projects are append-only BFT systems. An append-only BFT does not use POW and users are verified.

2.2.3 Applications of P2P Networks to Health Records

Both peer-to-peer and blockchain applications have been previously proposed.

King [76] proposed a peer-to-peer PHR utilizing Blue Button Plus and FHIR. The system uses a client-server architecture and a series of subnetworks. Peers can

[4] A Turing complete language can approximate any Turing machine (an idealized computer), or equivalently, create any algorithm that can run on a Turing machine [87]. A Turing complete language does not comment on the difficulty of implementation or the computational cost of a target program. A non-Turing complete language has limitations in what programs can be represented and in the case of bitcoin, is purposefully chosen to prevent unsafe programs.

request access to a patient's subnetwork. The architecture proposed in this thesis extends this previous work by providing full PHR functionality as well as integration with third-party service providers.

Roehrs [135] proposed a distributed PHR called OmniPHR. It uses the Chord algorithm to manage a series of encrypted blocks of health records over a set of peers. Each block is digitally signed. Each peer translates the records into open standards, distributes the blocks to peers, validates a peer's blocks, and manages communication between peers. Roehrs tested the system's network dynamics by increasing the number of nodes and "backbone" routers. He found that the Chord algorithm scaled with the number of peers and did not have increased latency. Roehr's system is similar to the PHR proposed in this thesis. However, Roehrs is focused on the performance analysis of the Chord algorithm. The focus of this thesis is on the functionalities needed to facilitate interactions between patients and health providers.

Xia [167] proposes a cloud-based health record sharing system that uses a Blockchain style ledger to track transfers of health records. Peers upload health record data to a shared cloud repository. Peers request and grant access to records through a transaction. A node selects transactions and computes a proof-of-work hash. After the proof-of-work is completed, the request is granted. An exchange protocol is then used to access the shared repository. This system benefits little from the Blockchain solution. Another issue of this solution is that data is stored off chain which removes the benefit of a Merkle tree [100]. Indeed, the transactions and hashes of the data are increasingly secure, but the data itself is not. The participants of the network need to trust the hash of the data and cannot verify it until reading, at which point it could be corrupted.

Urovi [155] proposed a Peer-to-Peer framework to exchange records across HIEs and the use of semantics to enable dynamic queries. The study explores integrating the Swiss healthcare system without reliance on a central authority. The model has a few limitations. For example, a provider needs predefined knowledge on where target records are stored and changes to a record are not propagated to providers. Com-

munication is based on a tuple space architecture called TuCSoN [121], a derivative of Linda [51]. A tuple space is a distributed shared memory programming paradigm that creates a series of shared objects between parties called *tuples*. The TuCSoN triple store ontology was modeled in OWL. This allowed the distributed objects to be defined and resolved at runtime. The underlying P2P architecture is a pure model that uses TOM P2P [21], a Kademlia DHT, for resource discovery. Each node, called a community, contains a replica of the tuple primitives with agents reacting to tuple changes. The implementation consists of three basic agents: log, update, and search. For each community, a single coordinator called a Policy Tuple Center is used to meditate the tuple actions. The available actions included connect, subscribe, and search. The system could query the network for a resource and then use the network's response to access the record. The agents could also respond to network requests. Three types of subscriptions were modeled: events, service changes, and policy changes. The implemented model was tested on Amazon cloud. The network's search complexity was in line with theoretical estimates but experienced extra latency as more communities subscribed. Overall, the use of tuple space offers a flexible distributed programming model whose definitions can be injected at runtime. However, several issues have been overlooked. For example, consensus, security, and data representation of current health domain ontologies have not been addressed.

2.3 Application Layer Protocol

Peer-to-peer networks are application protocols built on top of the transport layer of the OSI model [78]. Programs exchange data in the application layer in a known format and exchange pattern. Example application layer protocols are HTTP [46, 78] and FTP [78, 131]. The underlying transport layer delivers packets between applications. There are two common methods for packet delivery in the transport layer, TCP and UDP.

TCP is a connection-oriented protocol between a client and a server [8, 78]. It provides error-corrected and in-order delivery of packets. TCP provides a guarantee of the delivery of a message. A connection under the TCP protocol is a safe and reliable method to transfer data, at the cost of increased overhead.

UDP is a connection-less protocol [78, 130]. The sender sends packets to the receiver in a best-effort approach. The protocol cannot guarantee the arrival of a packet. The application protocol must re-request missing packets. The receiver verifies the packets with a checksum. This makes UDP suitable for real-time applications where low latency is a priority. It is up to the application layer to provide any additional guarantees.

2.4 EHR Datamining

Predicting a patient's future medical conditions is beneficial to patients, providers and the healthcare system. Patients can take proactive measures to prevent conditions, providers can recommend tests to perform, and the healthcare system can plan for future health needs. Identifying at-risk patients before conditions occur can lower the cost of medicine and saves lives. Current clinical research uses either randomized trials [32, 75] or cohort studies [32, 33, 96] to understand conditions and symptoms. This supports the approach of diagnosing patients by domain experts and Computer-aided Diagnosis (CAD) [39]. However, these methods are expensive, difficult to design, and suffer from selection bias [32, 75, 96].

Health providers have generated millions of patient records. Providers primarily use these records to treat patients and create workflows. This collection is valuable as it contains detailed health information for a population of patients rather than a small group. Researchers have proposed to use this wealth of data for the secondary purposes of clinical research, prediction systems, and clinical decision systems [34, 70, 108]. Present research into EHR datamining has had mixed results due to varying quality, encoding differences, and missing data [60, 159]. Researchers have proposed

the standardization and quality assessment of EHR data to overcome these issues [17]. Despite the problems with mining EHR repositories, researchers have conducted several large-scale studies [102]. They have used a variety of methods, such as Neural Network, Bayesian Statistics, Trees, and K-Nearest Neighbor [169]. However, medical institution control of the research data [90] makes evaluating these studies difficult.

Yadav [169] surveys the current research in EHR datamining. Applications of EHR datamining include: medical trajectories, cohort studies, risk prediction, intervention modeling, clinical guidelines, and detecting adverse events. Each of these applications have varying levels of implementation difficulties, with intervention modeling being notably difficult. EHRs contain three forms of health data with decreasing structure: structured, workflows, and clinical notes. Issues with EHR data include: difficulty in extraction, unobserved data, data fragmentation, bias, and irregular data.

Several recent papers propose utilizing Neural Networks to mine EHRs. Recent Neural Network approaches focus on many nested hidden layers of "neurons" to approximate complex functions. Given enough data, neural networks show a good performance on a variety of health modeling problems. Miotto et al. [102] and Trang et al. [128] approach EHR datamining from an unsupervised perspective with Neural Networks.

Miotto et al. [102] presents a large-scale study using Stack Denoising Autoencoders (SDA) and Random Forest to diagnose conditions such as diabetes, congestive heart failure, and schizophrenia. The authors choose autoencoders to counteract noise in EHR data and create a reduced representation of the patient records. The study used a heavily processed EHR repository of 700,000 patients. The authors annotated the dataset with the Open Biomedical Annotator [72] and extracted topics from clinical notes using regular expressions and Latent Dirichlet Allocation. EHR data needs heavy preprocessing to achieve good results [102] and the reduction of the feature space is key as EHR data has thousands of sparse features.

Trang et al. [128] proposed modeling a chronic patient's long-term outlook using Long Short-Term Memory (LSTM) neural networks. A LSTM is a variation of Recurrent Neural Network in which nodes store internal lossy data to replicate memory. The authors used a modified LSTM to model long term health trends and improve on methods like Hidden Markov Model for chronic conditions [128]. The authors trained and tested the LSTM with 12 years of EHR data comprising two datasets, with each dataset covering approximately 12,000 patients and 53,000 hospital admissions. The authors noted that the model struggled with patients who had small records.

2.4.1 Predictive Models and Bayesian Statistics

In this thesis, a Bayesian model is used to demonstrate predictive medical diagnosis as a service. Bayesian Statistics are probabilistic methods that model systems using Bayes theorem [125]. Bayes Theorem (Eq. 2.1) is used to infer the probability of events based on prior observations. These methods can be classifiers or generative models.

$$P(Outcome|Data) = \frac{P(Data|Outcome)P(Outcome)}{P(Data)} \tag{2.1}$$

where: $[P(Outcome|Data)]$ is the Posterior, $[P(Data|Outcome)]$ is the Likelihood, and $[P(Outcome)]$ is the Prior.

The simplest approach is Naive Bayes [136]. Naive Bayes is a classifier that models a set of features as random variables (RV) with the "naive" assumption that each is conditionally independent (Eq. 2.2). This simplifies learning and the inference process. Naive Bayes can perform well despite this simplification [136] and is used heavily in text mining [141].

$$P(C, E_n) = P(C) \prod_i P(E_i|C) \tag{2.2}$$

where: $[C]$ = outcome and $[E_n]$ = evidence

Another approach is Bayesian Network. Bayesian Network are Directed Acyclic Graph (DAG) with nodes representing RVs [136]. The space modeled is the joint distribution of the RVs but in a compact form. A connection between nodes indicates conditional dependency. The combined connections and probabilities create a model of the system representing interrelated events and their probabilities. By using Bayes Theorem and the graph structure, a query can traverse the network to infer the conditional probabilities of events given evidence. Bayesian Networks have advantages over Naive Bayes. The structure intuitively models the relationships between features and can represent complex models with conditional dependencies. Moreover, experts can use domain knowledge to define the graph structure, thereby bootstrapping the network. This is helpful as deriving the graphical structure of the Bayesian Network is otherwise NP-Hard [136].

Inferencing, computing the probabilities of an event given data, in complex Bayesian models is NP-Hard [136]. To overcome this issue, models can be trained using Expectation Maximization (EM) [136], Gibbs Sampling (a variation of Markov Chain Monte Carlo) [136], or Variational Methods [73]. The EM algorithm iteratively finds the probability distributions but can converge to local optima. Gibbs Sampling and Variational Methods use randomization or optimization techniques to approximate the distributions in order to avoid converging to a local optima.

2.4.2 Bayesian Applications to EHR Datamining

Lucas et al. [91] presents a review of Bayesian Networks for medical diagnosis. The representation of probabilistic knowledge as a graph creates a formalism that mirrors how medical domain experts see cause and effect relationships. It is an intuitive model, with easy to understand relationships and probabilities. Clinical experts can bootstrap the model allowing the application of prior diagnoses knowledge. Lucas demonstrates a theoretical application of Bayesian Networks to diagnose ventilator-associated pneumonia.

Sexias et al. [143] proposes using Bayesian Networks for Clinical Decision Support Systems targeting Dementia, Alzheimer's, and Mild Cognitive Impairment. The authors reviewed similar CDSSs that used Bayesian Network, Neural Networks, and Multiple Criteria Decision Analysis. The study used 1,804 patient neuropsychological tests from four institutions for the dataset. The authors also used Synthetic Minority Over-sampling Technique (SMOTE) [30] to balance the dataset. Another processing step was the removal of attributes not related to diagnosis and records missing values. An expert predefined the network structure. The expert noted that several features that were removed for low counts were highly relevant to the classification. The authors generated five Bayesian Networks using the Waikato Environment for Knowledge Analysis (WEKA) software [57] along with the GeNIe Modeler and SMILE Engine [41]. The Bayesian Networks used a three-level template and the Dirichlet Distribution for the nodes. The expert-defined Bayesian network outperformed the auto-generated Bayesian Network but had similar performance to other methods. This study shows the benefit of using an expert to bootstrap the network.

Other approaches use Dynamic Bayesian Network (DBN) to account for time. Dynamic Bayesian Network are a generalization of Bayesian Networks and Hidden Markov Model that can model a system over time. Nachimuthu et al. [110] used DBNs to predict sepsis, a rapidly progressing condition in Intensive Care Units (ICU). The authors chose DBNs to account for the increase in evidence over time in an ICU. The study used 3,100 patients from an Intermountain Healthcare ICU dataset. The data was skewed towards control patients by 4 to 1 and was in various unstructured formats. The researchers extracted lab and vital signs for the first six hours. To reduce sparseness, the researchers aggregated the data to 1-hour intervals. The authors did not remove records with missing data and all variables were discretized. The input space to the model was a set of time increasing vectors of clinical data and observations for each patient. A physician labeled the data for sepsis diagnosis. The researchers created the DBN using Projeny, a front end to MATLAB's Bayes Net Toolbox, and trained the network using the EM algorithm. The DBN structure was defined by

the researchers and used two time slices. Overall the network performed well, with all performance measures improving over time. Despite the promising performance, this study does have limitations. The authors noted that missing data prevented the diagnosis of specific types of sepsis such as septic shock.

Xu et al. [168] propose using a Hierarchical Bayesian Non-Parametric (BNP) to model patient's responses to treatment. Patients have varied responses to treatments over time which motivated Xu to consider BNPs. The stated novel aspects were a continuous response over time, flexibility in modeling, and probability estimates at the patient level. The paper notes that longitudinal EHR data has the "time-varying confounding issue", where treatments are present at a higher rate in sicker patients. This creates the impression that treatment caused the condition. Xu used the G-computation formula as a guide for creating the BNP. The dataset consisted of EHR time series data of 123 patients from the Boston Beth Israel Deaconess Medical Center. The patients averaged a 20.75-day stay and totaled 6,992 observations. The BNP modeled the patient's kidney function in response to Ischemic Heart Disease treatments and Continuous Renal Replacement Therapies. The authors trained the model using Gibbs Sampling and Metropolis-Hastings, another MCMC method. Results were measured with Root Mean Squared Error (RSME) for Creatinine (a measure of renal health) and compared against baseline responses from the model using different distributions. The model performed well against fully and partially observed data. Overall, this paper presents interesting ideas related to modeling noise, using BNPs for time series data, and the issue of cause and effect in longitudinal data. Gaps in this paper were the failure to examine the scalability of the BNP method and effects of the small population size on the produced results.

2.4.3 EHR Datasets

Current EHR repositories have known issues with quality and sparseness. Weiskopf et al. [158] and Hersh et al. [60] reviewed the quality of EHR data. Weiskopf [158] examined 3.9 million patient records from the New York Presbyterian Hospital for completeness. The study tested records for documentation, breadth, number of visits, and predictive quality using logistical regression. The authors found that clinical notes increased after EHR adoption. Many records were sparse, only 45% included diagnosis and 13% included medication. The repository had limited longitudinal value, only 23% of the records had 5 follow-up visits after the initial visit. The author concludes that only 0.6 % of the records are complete but recommends not generalizing the results to other hospital systems. Hersh [60] summarized the EHR issues from multiple studies. The author found common themes of inaccurate data, incomplete records, poor semantic encoding, data unrecoverable for research, unknown provenance, insufficient granularity, and incompatibility with research protocols. Both authors paint a dim outlook on using EHR data for clinical research. Moreover, Hersh [158] notes the limited usefulness of the International Classification of Diseases Revision 9 Clinical Modification (ICD-9-CM) [112] for studies.

The Health Insurance Portability and Accountability Act of 1996 (HIPAA) protects patient medical records from unauthorized access [156]. The majority of the reviewed literature on EHR datamining used a restricted dataset provided by a hospital. Yadav et al. [169] mentions that large scale studies before EHRs, used insurance claims data. Several open datasets target single conditions. Delen et al. [38] used the Surveillance, Epidemiology, and End Results Program (SEER) for predicting breast cancer survival rates. However, SEER is not a substitute for an EHR as it only covers cancer statistics in the United States. Xu et al. [168] used an open ICU EHR dataset, however, it does not include long term changes.

The Medical Expenditure Panel Survey (MEPS) [4] is a yearly survey of households by the Agency for Healthcare Research and Quality (ARHQ). ARHQ's goal is

to survey current patient healthcare in the United States. In each survey, ARHQ asks households questions about conditions, providers, costs and medication. Each survey group is followed over a two-year period termed a *panel* with questions over 5 rounds (Fig. 2.2). There are currently 20 panels available, starting from 1996. Each panel includes approximately 15,000 households with a subset of the previous year's respondents carried over. ARHQ warns that MEPS data is missing values, which affects the overall quality of the data. For example, in Panel 19, only 50% of the participants responded. ARHQ made attempts to address this issue by altering the survey design but the result is inconclusive. The questions and recorded information vary over the years.

ARHQ hosts the surveys online and provides the data files in SSP and fixed length format. Each survey is by year with two active panels per year [6](Fig. 2.2). ARHQ groups the information by household and event components. The household component covers survey questions while the event component tracks patient reported medical events. The household component contains tables covering topics such as conditions and medications. The link between the tables is the respondent's `DUPERSID`, a concatenation of the respondent's ID and the household ID.

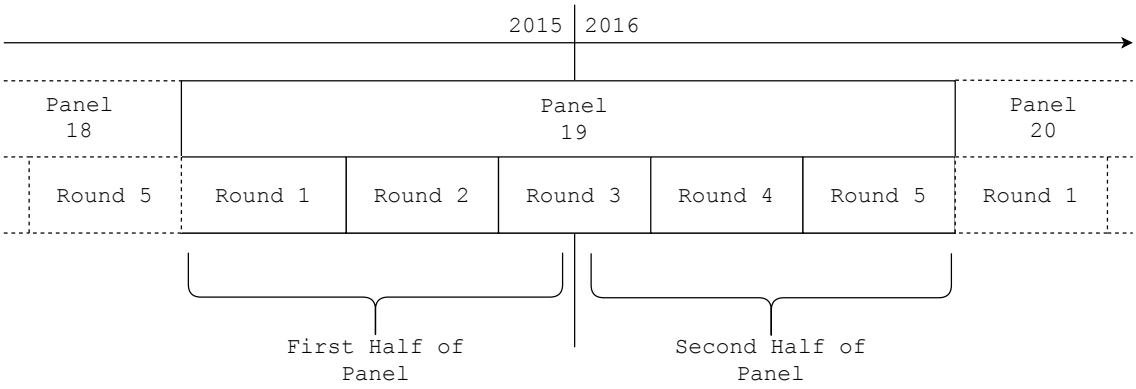


Fig. 2.2.: MEPS Panel Design

3. ARCHITECTURE

The architecture (Fig. 3.1) of the proposed system is a centralized Peer-to-Peer model, consisting of: an index server, a peer client, and registered network services. The peer client is an application with a Graphical-User-Interface (GUI) for patients and providers to manage health records and interact with the network. The index server controls network access and provides resource lookup for the peers. Once connected to the network, a peer can post document metadata and request transactions. The target peer can then authorize the transaction. Peers periodically pull authorized transactions from the network. A peer carries out the transaction by directly contacting the target peer.

The user stores their health records on their own computer. Each new record has associated metadata (Fig. 3.2). The metadata includes the origin, creation time, and a SHA256 [9] hash of the data to establish its uniqueness. It is this metadata that is uploaded to the network for discovery, not the underlying record. Access is only granted by following the established transaction protocol and with the approval of the owner. This makes the user in control of their health records and the system a Personal Health Record.

The index server manages the network for the peers and performs the signaling, signup and lookup services for the network. This centralized model tracks the registered users, peer IP addresses, documents, transactions, and services.

Patients and Providers use the peer client to register in the network, login, manage local health records, lookup documents and services, upload records into the network, and issue requests for transactions. In addition, the client operates a P2P client-server subsystem to carry out approved transactions.

A service client is a special peer client for network services. Regular peers can request these services, which are logged and carried out as transactions. The service

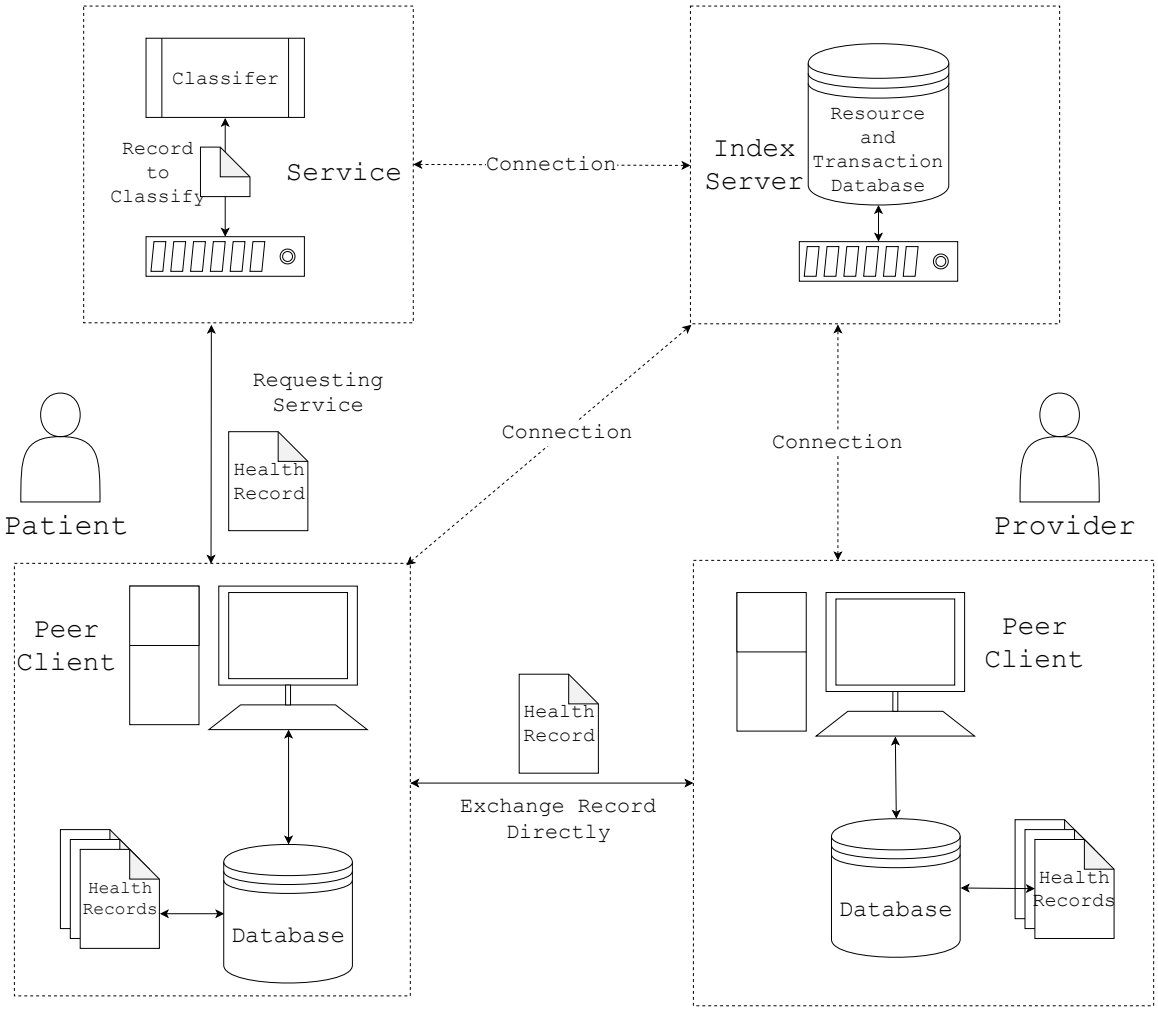


Fig. 3.1.: System Architecture

```
Document {
Hash   sha256
Time   time
Origin string
Data   fhir
}
```

Fig. 3.2.: Client Document

client periodically checks the network for new service transaction requests and approves them. The service client does not store any records. It returns the result of the serivce application to the requesting peer.

3.1 Network Actions

There are several types of peer actions in the network.

Registration: The peer client presents network actions as a series of pages. A user can register in the network by filling out the registration information (Fig. 3.3). The peer client parses the form information and creates a registration message which is submitted to the network. The server validates this information, and if valid, a new account is created and a success response is returned to the client. Otherwise, an error is returned.

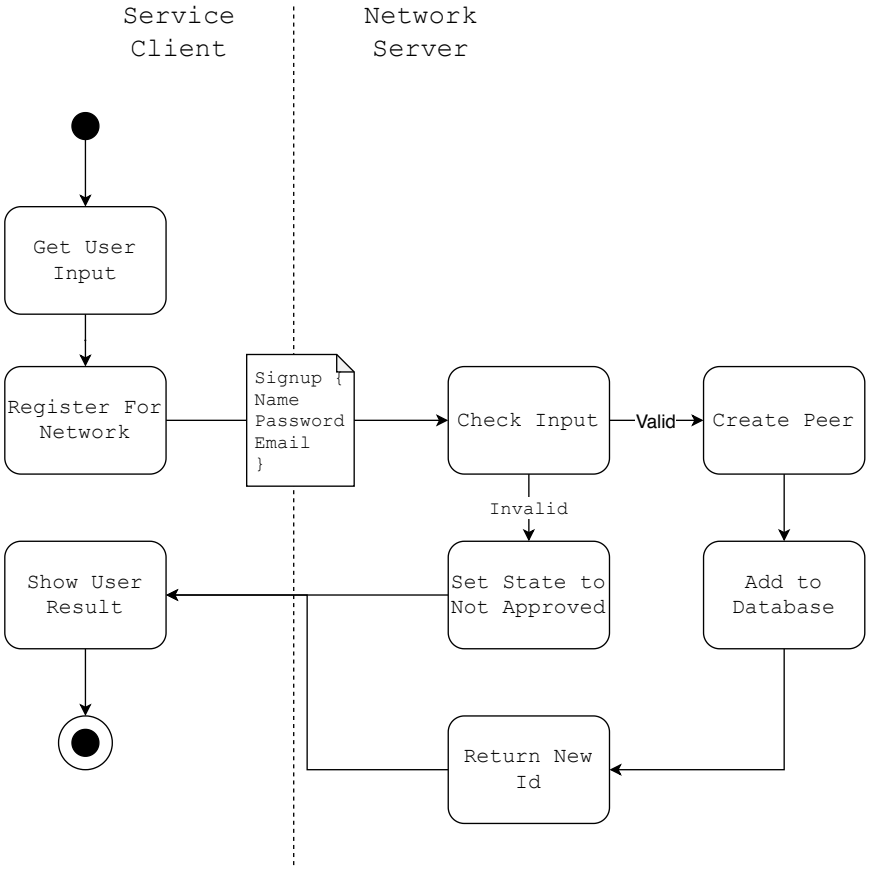


Fig. 3.3.: Registration

Login: The user logs into the network by entering their credentials. The client parses the information and creates a new network client. This network client creates a heartbeat process (Fig. 3.4) which continuously pings the server with route information on a predefined interval. Other peers use this route information to locate other clients.

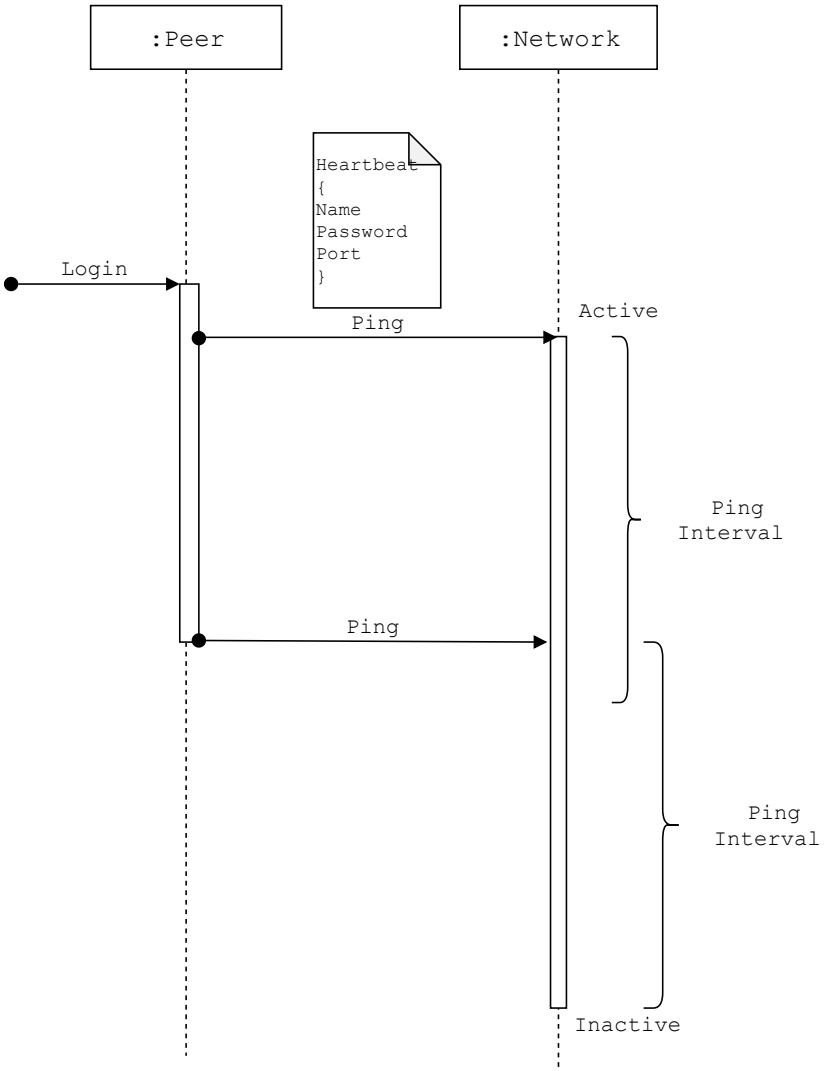


Fig. 3.4.: Heartbeat Process

Search: The client can search for network resources (Fig 3.5). Resources include account information, posted documents, transactions, portals, and services. Each query can be refined to narrow the search. The client parses the query and sends the result to the index server. The index server validates the query, executes it, and returns the results of the search to the client.

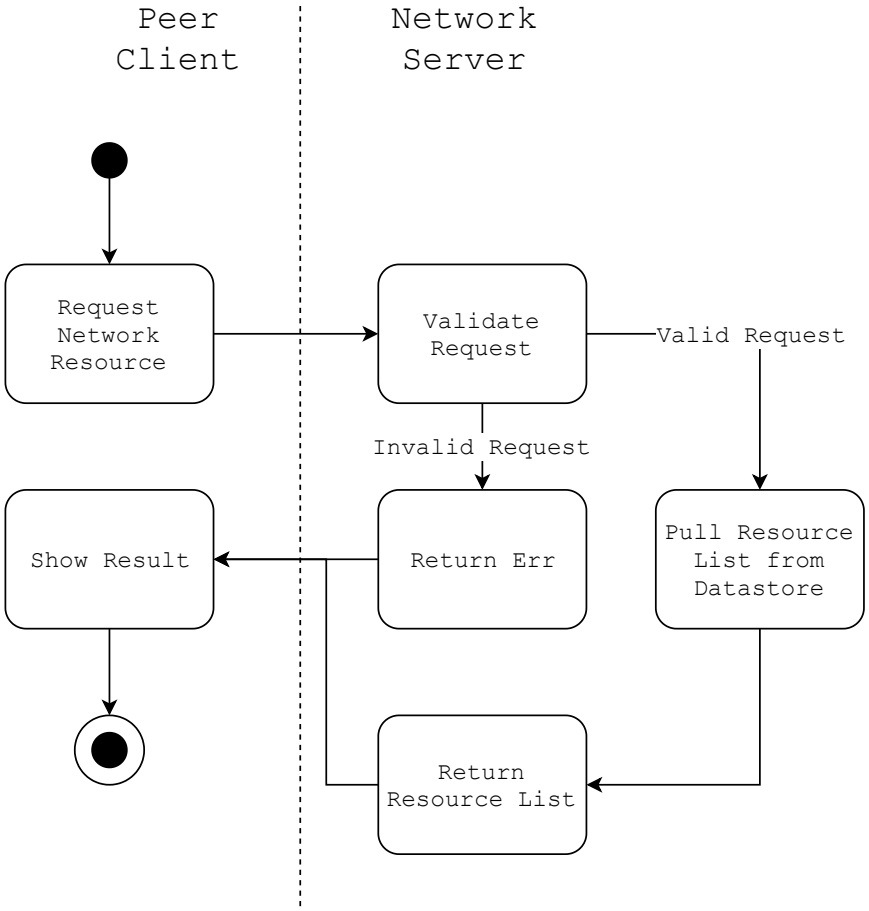


Fig. 3.5.: Search for Network Resources

Record Download: The client can download data from external sources. The index server holds a list of verified portal servers. The client queries the index server for this list and selects the desired portal. The user enters their portal login information and the client parses and creates a FHIR query. The client executes the query against the portal which returns the user's documents (Fig. 3.6). If the query fails, the user is alerted. The downloaded documents are then compared against the local records. If the documents are the same, the record is not added to the database. Otherwise, a record is created and added to the peer database.

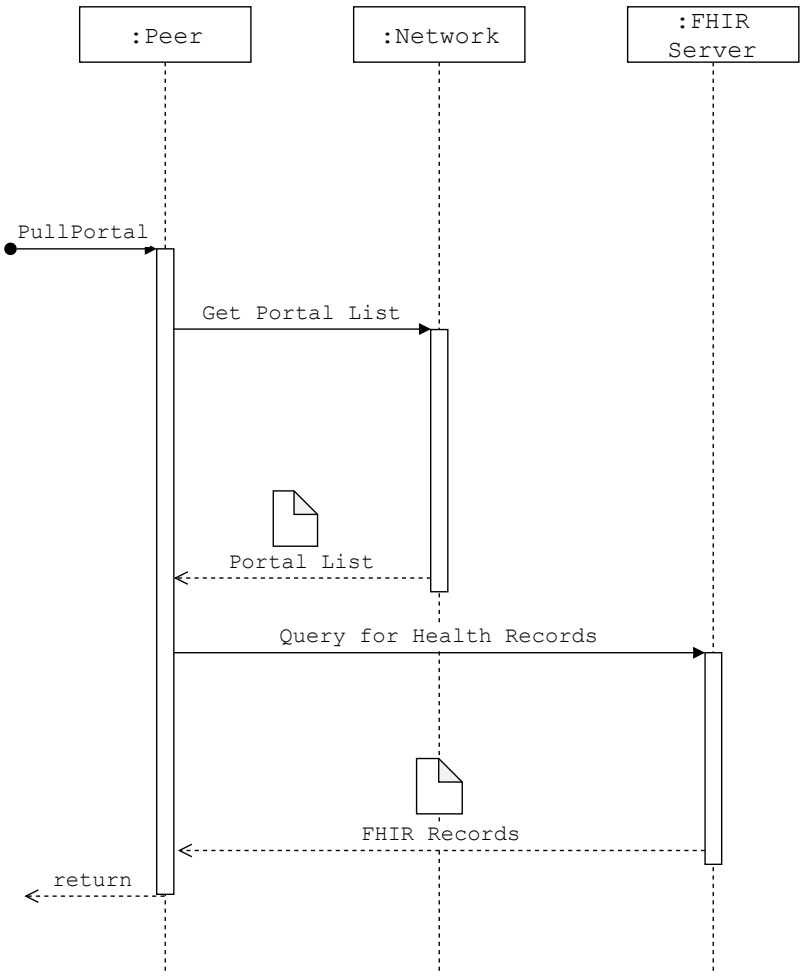


Fig. 3.6.: Fetch FHIR Documents from Portal

Record Registration: A user can register a document in the network (Fig. 3.7). The client parses the user's selection and grabs the requested document's metadata. The client sends the metadata to the index server. The index server validates the request and checks that no network document has the same SHA256 as the posted document. If the request is valid, then the document is assigned to the user. Otherwise, the request is rejected.

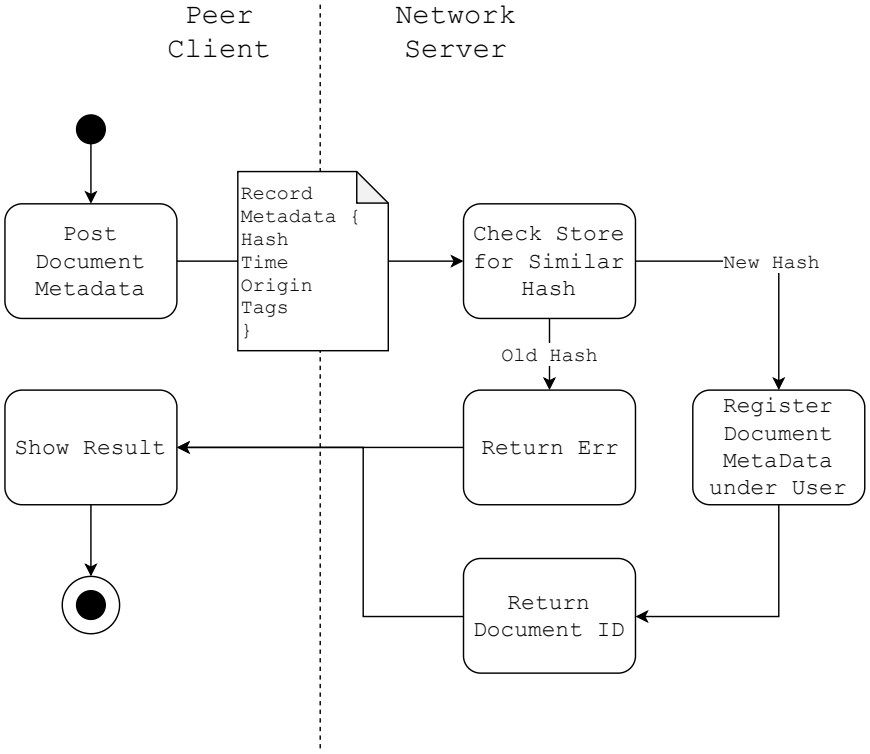


Fig. 3.7.: Record Registration

Service Registration: A service provider can issue a request for the registration of a new service by sending a registration request to the network (Fig. 3.8). The network administrator checks the request and approves or denies the new service. If the service is approved, a new peer client is created for the service and the service becomes visible to the peers.

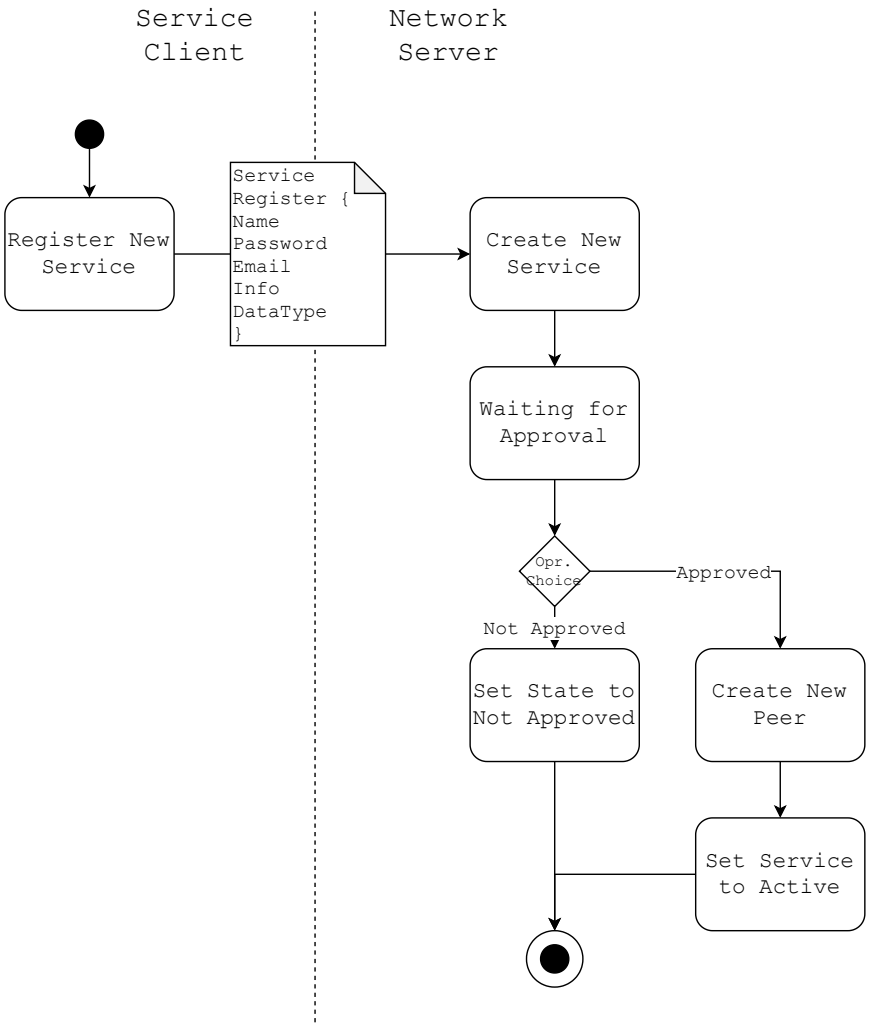


Fig. 3.8.: Registration and Approval of Service

3.2 Transactions

Transactions are exchanges of records between two peers. Each type of transaction is treated as a state machine, with the state of the transactions changing as conditions are met. The index server acts as the ground truth between the two peers and ensures that transactions are both valid and consistent. In addition, the index server provides an independent log of the transaction for audit purposes.

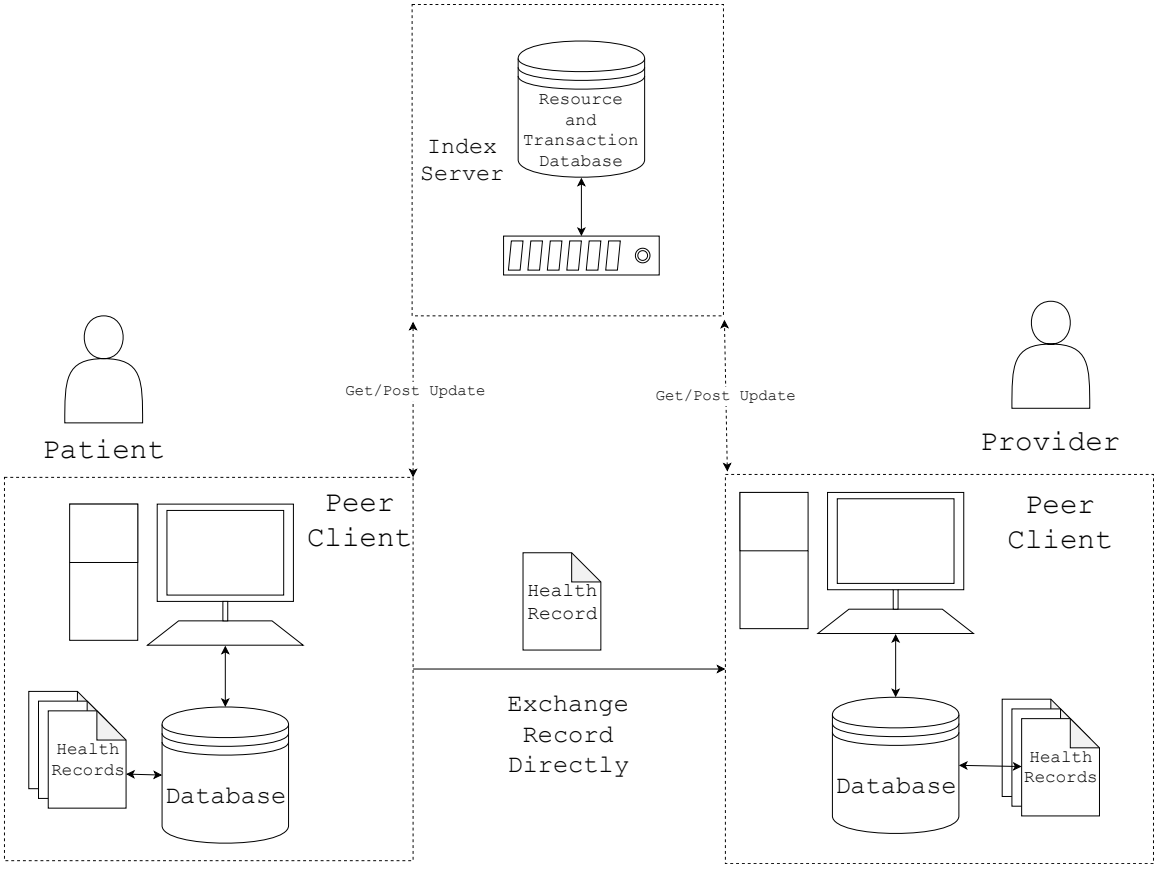


Fig. 3.9.: Network Transaction

3.2.1 Authorization and Operation

A peer, called the *requester*, can issue a transaction request on a network document to the index server. The client posts the document hash and the requested action to the server (Fig. 3.10). The server validates the request and creates a new transaction in the *pending* state (Fig. 3.11). The target peer can view the list of pending transactions. He or she can then approve, reject, or ignore these requests and posts the decisions to the index server. The server validates the peer and if the target approved the transaction, moves the transaction to the next state. If the target rejects the request, the state is changed to *canceled* and the transaction is non-executable by the clients.

```
Transaction Request {
Action     string
ServiceID  string
UserID     string
DocumentID string
}
```

Fig. 3.10.: Transaction Request Data

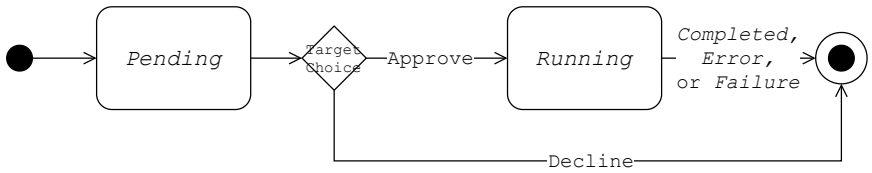


Fig. 3.11.: Transaction Request and Approval

Once a transaction is approved, the participating clients can pull the transaction from the server (Fig. 3.12). The transaction is added to each client's list of transactions that need to be executed.

```
Transaction {
ID          string
Hash        string
SenderID    string
SenderIP    string
ReceiverID  string
ReceiverIP  string
Port        string
Action      string
State       string
Time        time
}
```

Fig. 3.12.: Transaction Data

```
Update {
Tid  string
Type string
Data rawdata
}
```

Fig. 3.13.: Update Data

To execute transactions, the peers operate both as a client and a server. The requester initiates the connection with the target peer. The two peers carry out the protocol and state of the transaction. At the end of the action, the peers alert the index server of the result with an update message (Fig. 3.13). The index server, also operating as a state machine, checks the messages to see if a state progress condition is satisfied. If a transition condition is satisfied, the index server moves the transaction to the next protocol state. Otherwise, a failure occurs, and the clients will need to retry the transaction. The clients continue executing the transaction until the protocol is completed or an error occurs.

3.2.2 Protocol

The method of the transaction determines the type of state machine that is invoked. Each state machine is a template with the following states: *pending*, *final*, *failed*, and *canceled*. The *pending* state shows that the owner has not approved a transaction. The *final* state indicates the end of a successful transaction. The *failed* state indicates that a transaction failed. The *canceled* state shows that the owner of the underlying data canceled the transaction. All other states are variable and depend on the protocol. There are three protocols modeled in the network: *Get*, *Push*, and *Service*.

Get: The *Get* protocol transfers a document from the owner to the requesting peer (Fig. 3.14). It has three states: *pending*, *get*, and *final*. While waiting for approval, the transaction is in the pending state. The owner must approve or disapprove a transaction before the transaction can move to the next state. Once approved, the transaction is moved to the *get* state. The requester connects to the owner. The owner returns the approved document to the requester. The two clients alert the index server that the transfer is complete. The index server sets the transaction to *final* state, thus completing the transaction.

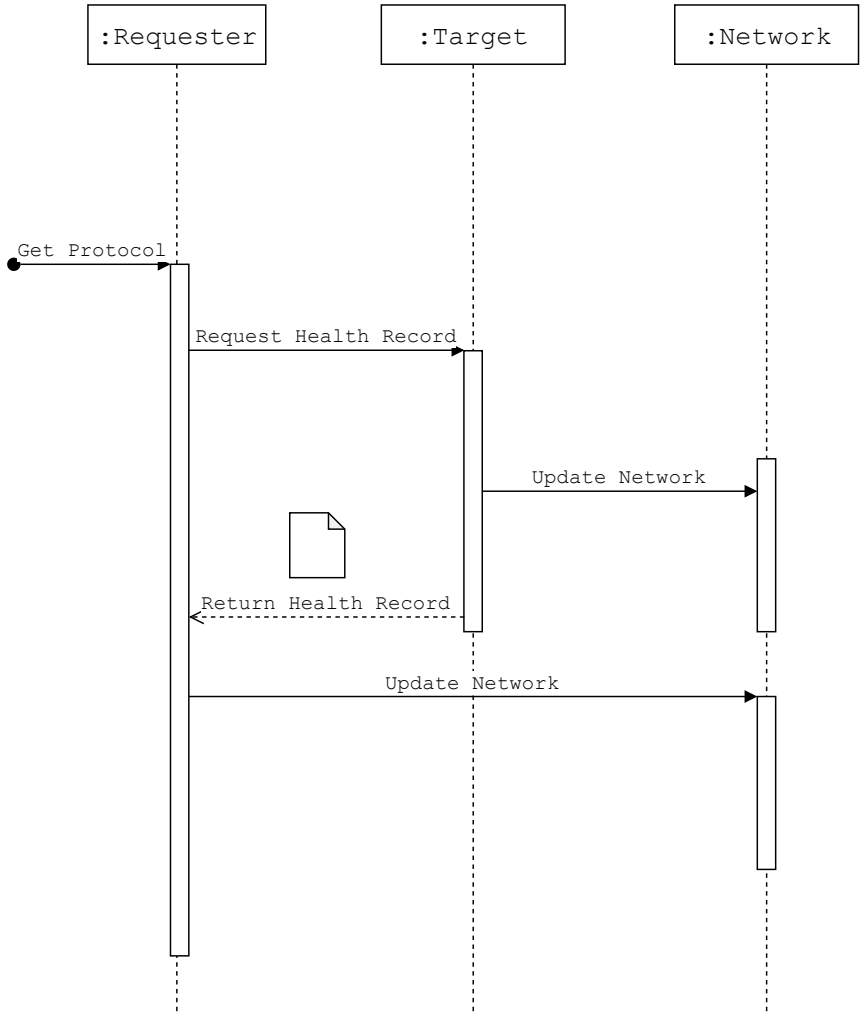


Fig. 3.14.: Get Protocol

Push: The *Push* protocol transfers a document from the owner to the accepting party (Fig. 3.15). It has three states: *pending*, *push*, and *final*. A push transaction starts in the *pending* state. The target approves the transaction and the index server moves it to the *push* state. The owner of the document opens a connection with the target and transfers the document. The target adds the record to their data store. The two peers then alert the index server that the transaction is complete. The index server moves the transaction to the *final* state, thus completing the transaction.

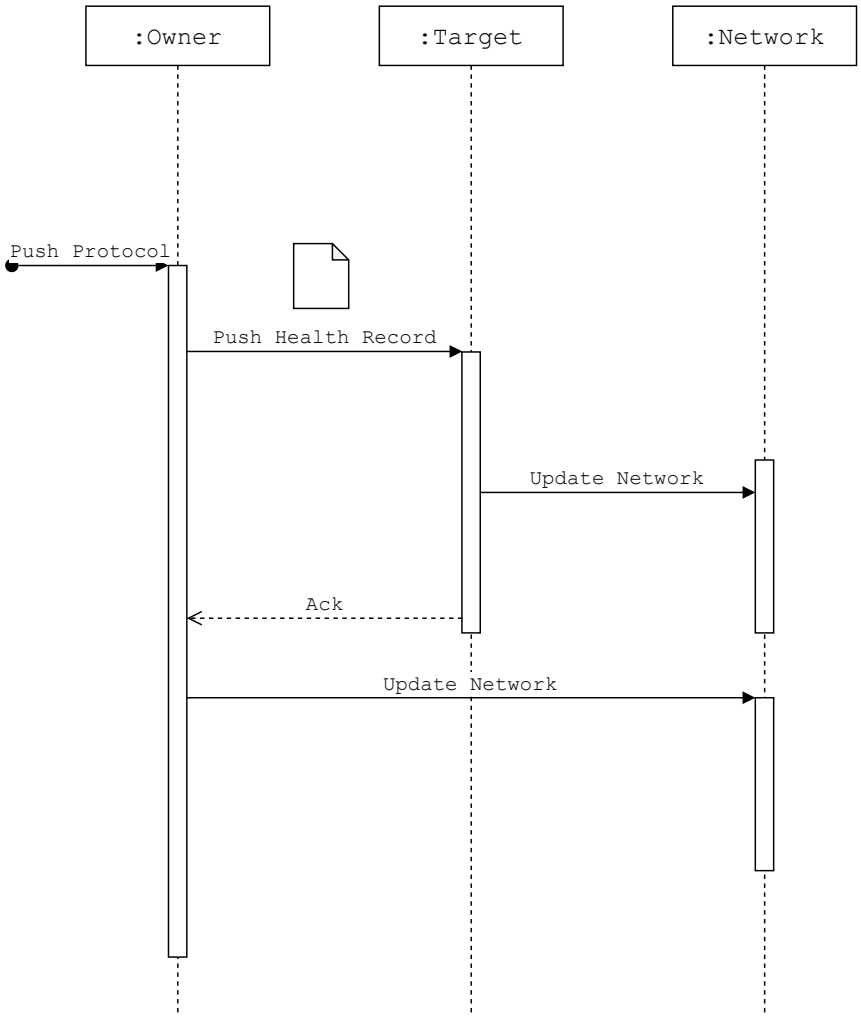


Fig. 3.15.: Push Protocol

Service: The *Service* protocol sends a document from an owner to a service and the service returns a new document in response (Fig. 3.16). It has three states: *waiting*, *service*, and *final*. The transaction starts in the *waiting* state. The service approves the transaction and the index service changes its state to *service*. The owner connects to the service and sends the agreed upon document. The service performs the requested action and returns the result to the owner. The owner adds the new document to their data store. The two peers alert the index server that the transaction is complete. The index server places the transaction in the *final* state.

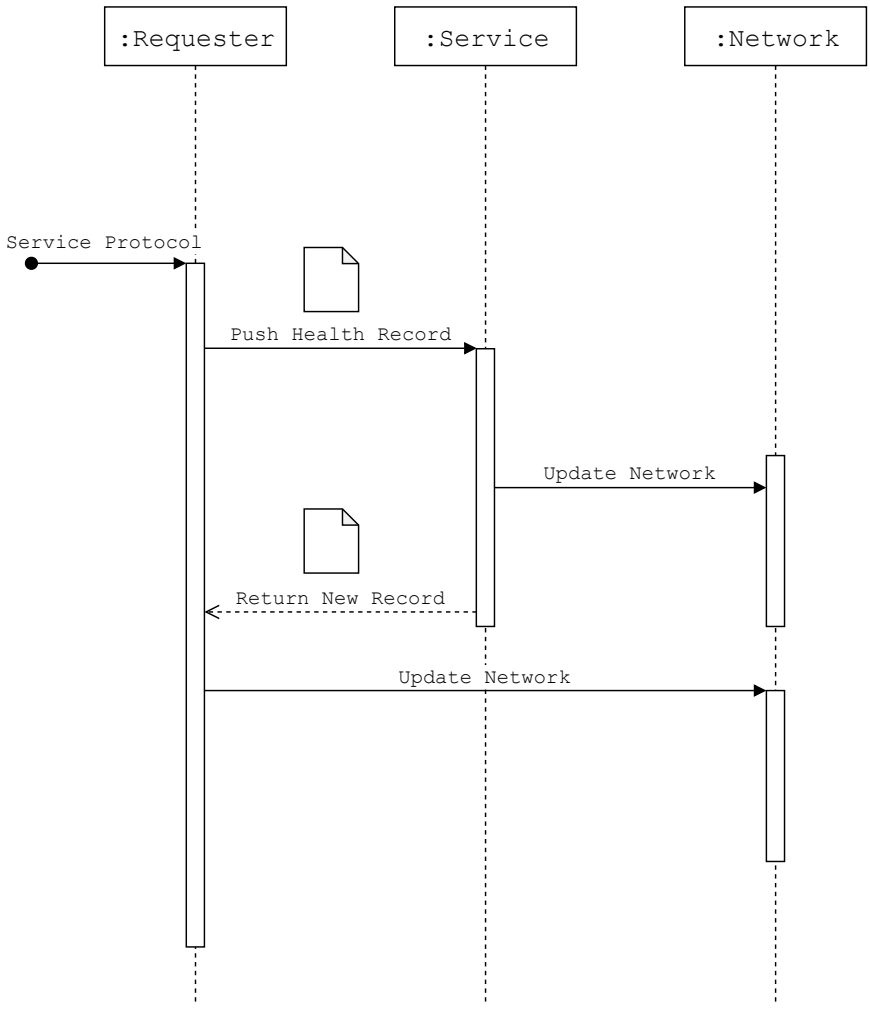


Fig. 3.16.: Service Protocol

3.3 Hypertension Prediction Service

The Hypertension prediction service is a network service that runs a Naive Bayes classifier. The classifier predicts a patient's risk of developing hypertension in the next year based on the patient's historical health record. The classifier is trained with data from the Medical Expenditure Panel Survey (MEPS) [4]. Though not a patient health record, the MEPS dataset is used to demonstrate the proposed approach and service protocol. The classifier targets Hypertension (High Blood Pressure), ICD-9-CM Code 401 [112]. Hypertension was chosen as a demonstration of the proposed service as it is the highest occurring condition in MEPS.

The prediction service has two stages: development and deployment. During the development stage, the classifier is trained and tested using a MEPS dataset. The model is then deployed as a service in the proposed PHR.

Records are extracted from MEPS and transformed into a vector of patient demographics, prior conditions, and labels of posterior conditions. The selected features included 30 related ICD-9-CM conditions and 17 health related survey questions. This dataset was split into training and test datasets, with 70% of the records for training and the remaining 30% for testing. The output of the classifier is a binary prediction that the patient will develop Hypertension over the next year.

The Hypertension prediction classifier is then added to the network and operates as the Hypertension prediction service. A peer can connect to the service and pass the target health record which conforms to the classifier's requested format. The service invokes the classifier on the supplied data and returns the result to the client as a new record.

3.3.1 Data Set

The dataset used to train and test the Hypertension Prediction Classifier is comprised of two MEPS tables: consolidated and conditions. The consolidated table contains the bulk of the respondent's information, including demographics, insurance, income, etc. The condition table contains the self-reported conditions coded in ICD-9-CM for each respondent. To protect respondents, ARHQ only recorded the first three characters of ICD-9-CM [5], dropping the number of codes from 13,000 to less than 1,000. ARHQ further obfuscated the information of children and at-risk groups.

The highest occurring condition in Panel 19 is Hypertension (ICD-9-CM 401) with 3,078 instances. The high counts of Hypertension and accessibility to heart-related information motivated the choice of Hypertension as a target disease for this thesis. A review of heart condition literature helped prune the list of conditions to 30 ICD-9-CM codes covering heart and respiratory conditions (Table 3.1). The consolidated table has few direct measures of health such as Body Mass Index (BMI) [12]. However, several questions on advice from doctors, major health events such as stroke, and smoking information are also available [5]. The respondent's answers to these questions are used as additional measure of the respondent's cardiovascular health. This resulted in an additional 17 features. The selected features are included in Table 3.2.

Table 3.1.: Selected Condition Features

ICD-9-CM Group	**Meps Code**	**Feature**
410-414: Heart Disease	410	Acute Myocardial Infraction
	413	Angina Pectoris
	414	Other forms of Ischemic Heart Disease

continued on next page

Table 3.1.: *continued*

ICD-9-CM Group	**Meps Code**	**Feature**
415-417: Heart Failure	415	Acute Pulmonary Heart Disease
420-429: Circulatory System Diseases	424	Other Diseases of Endocardium
	425	Cardiomyopathy
	427	Cardiac Dysrhythmis
	428	Heart Failure
	429	Ill-defined Descriptions and Complications of Heart Disease
430-438: Cardio Brain Hemorrhage, Stroke	436	Acute, but ill-defined, Cerebrovascular Disease
440-449: Restricted Blood Flow	440	Atherosclerosis
	441	Aortic Aneurysm and Dissection
	442	Other Aneurysm
	443	Other Peripheral Vascular Disease
	444	Arterial Embolism and Thrombosis
	447	Other Disorders of Arteries and Arterioles
451-459: Issues with Veins and Lymph System	454	Varicose Veins of Lower Extremities
	455	Hemorrhoids
	458	Hypotension

continued on next page

Table 3.1.: *continued*

ICD-9-CM Group	**Meps Code**	**Feature**
	459	Other Disorders of Circulatory System
490-496: Chronic Respiratory Issues	490	Bronchitis, not specified as Acute or Chronic
	491	Chronic Bronchitis
	492	Emphysema
	493	Asthma
	496	Chronic Airway Obstruction, not elsewhere classified
510-519: Secondary Respiratory Conditions	511	Pleurisy
	514	Pulmonary Congestion and Hypostasis
	518	Other Diseases of Lung
	519	Other Diseases of Respiratory System

ARHQ conducted panels 17, 18, and 19 from 2012 to 2015 and the related data is stored in the H15 to H18 datasets. The dataset covers 26,563 patients. The split of each dataset creates a natural break point to test prior and post conditions. The first half of the panel (i.e., rounds 1 and 2, and the first part of round 3) is taken as the set of features to predict hypertension. The second half (i.e., second part of round 3 and rounds 4 and 5) is used to build the outcome label of whether the patient contracted Hypertension. Not all patients fully responded to the second half of the survey which reduced the usable vectors. The conditions and consolidated datasets for each panel

Table 3.2.: Selected Consolidated Features

MEPS Feature	**Description**	**Type**
SEX	Sex	Discrete
RACE1VX	Race (edited/imputed)	Categorical
AGELAST	Person's age last time eligible	Categorical
HIBPDX	High Blood pressure diagnosis	Continuous
CHHDX	Coronary Heart Disease diagnosis	Discrete
ANGIDX	Angina diagnosis	Discrete
MIDX	Heart Attack (MI) Diagnosis	Discrete
OHRTDX	Other Heart Disease Diagnosis	Discrete
STRKDX	Stroke Diagnosis	Discrete
EMPHDX	Emphysema Diagnosis	Discrete
CHOLDX	High Cholesterol Diagnosis	Discrete
DIABDX	Diabetes Diagnosis	Discrete
ADSMOKE42	SAQ: Currently Smoke	Discrete
NOFAT53	Restrict High Fat/Cholesterol	Discrete
EXRCIS53	Advised to exercise more	Discrete
BMINDX53	Adult Body Mass Index	Continuous
POVCAT	Family Income as Percentage of Poverty Line	Categorical

are pulled from the ARHQ website and loaded into SAS [137]. SQL joins are then used to extract the targeted panels and features across the datasets.

3.3.2 Preprocessing

The compiled dataset is further processed (Fig. 3.17) with the Python Data Analysis Library (pandas) [97]. A custom pipeline converts the compiled prior to posterior pairs into a vector of demographic information, conditions prior the time split (i.e., historical information), and the conditions post time split (i.e., predictive outcome) for each patient. It consists of three stages: feature vector creation, label vector creation, and feature cleanup.

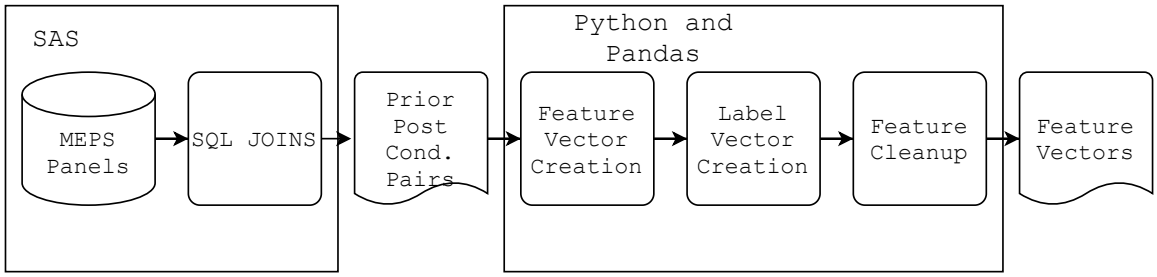


Fig. 3.17.: MEPS Data Preprocessing Pipeline

The fist stage converts the feature vector and labels. It compresses the list of conditions for each respondent into a single vector (Alg. 3.1 [1]). The next stage compresses the labels into a single vector (Alg. 3.2). Labels are prepended with `L` to differentiate labels from features.

[1]Error handling has been removed from all algorithm due to length.

```
label = ""
person = ""
feature = pd.Series()
l = []
for index, row in raw.iterrows():
    if row["LABEL"] != label or row["DUPERSID"] != person:
        if type(feature) != None:
            l.append(feature)
        feature = pd.Series()
        add_demographics(row, feature)
        add_condition(row, feature)
        label = row["LABEL"]
        person = row["DUPERSID"]

    else:
        add_condition(row, feature)
feature_vector = feature_vector.append(l)
```

Algorithm 3.1: Vector Creation

```
l = []
vector = pd.Series()
person = ""
for index, row in feature_set.iterrows():
    if row["DUPERSID"] != person:
        vector["LABEL"] = 0
        l.append(vector)
        vector = row
        person = row["DUPERSID"]
    label = row["LABEL"]
    vector["l" + str(label)] = 1.0
vector["LABEL"] = 0
l.append(vector)
data = data.append(l)
```

Algorithm 3.2: Label Creation

The final preprocessing step converts the features to categorical values and replaces unknown responses with default values. BMI is converted into the standard ranges of *underweight*, *normalweight*, *overweight*, and *obese* [12]. The age is converted into ranges of 10 years. Unknown values for demographic questions are converted to either *no* or to an empty variable depending on the count.

The resulting dataset had 26,653 vectors. The ICD-9-CM features and labels are binary. The other features are categorical or binary. Missing prior conditions or labels are set to 0. This may cause under-reporting. However, this was unavoidable since MEPS does not have definite *no* answers for many cases. The feature vector layout is shown in Figure 3.18 and an example feature vector is included in appendix B.

DUPERSID	Consolidated Features	ICD-9-CM Prior Features	ICD-9-CM Post Labels

Fig. 3.18.: Feature Vector

The dataset is randomly divided into training and test data. The split is 70%, or 18,657 records, for training and the remaining 30%, 7,996 records, for testing.

3.3.3 Algorithm

The proposed classifier uses Naive Bayes (Eq. 2.2). As mentioned in section 2.4.1, Naive Bayes is a supervised learning algorithm that requires labeled data. It is a restricted Bayesian Network with the assumption of conditional independence between features. This assumption makes inference of an outcome given evidence a product of the conditional probabilities of evidence. After inferring the probability of each event given the evidence, the most likely result, called the *maximum a posteriori* (MAP) [136], is selected as the prediction outcome.

3.3.4 Training and Testing

The model's accuracy is evaluated with hypothesis testing. Hypothesis testing creates a 2x2 matrix of the results of the binary classification. This matrix is sometimes called a confusion matrix. It contains the true positive (TP), false positive (FP), false negative (FN), and true negative (TN) counts of the prediction versus the label [80, 136]. High FP or FN rates may be unacceptable in certain environments (e.g., cancer diagnosis). The confusion matrix is aggregated to derive the *Precision* (Eq. 3.1), *Sensitivity* (Eq. 3.2), and *Specificity* (Eq. 3.3) of the model. The *Precision*, also called the positive predictive value (PPV), measures the performance of classifying a true positive given a positive result. The *Sensitivity* measures the performance of correct identification of positive results. The *Specificity* measures the classification of events that return negative results.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{3.1}$$

$$Sensitivity = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3.2}$$

$$Specificity = \frac{TrueNegative}{TrueNegative + FalsePositive} \tag{3.3}$$

4. IMPLEMENTATION

The network server, peer client, and service client are implemented with Go (Golang) [53] and MongoDB [105]. Libraries used include mgo [114] and the FHIR intervention engine [103]. The Hypertension classifier subsystem is implemented with Python3 [132], SAS [137], and pandas [97].

4.1 Index Server

The index server supports account management, resource lookup, and transaction management (Fig. 4.1). It is programmed with Golang and uses MongoDB for persistent storage. Clients access the service through a Representational State Transfer (REST) [47] API.

4.1.1 Database

The network server connects to a MongoDB [105] server though the mgo [114] driver. MongoDB stores all documents in BSON [107], a binary JSON format. Tables in a MongoDB are called *collections* [106]. The database has the following collections: user, document, transaction, service, and FHIR (Fig. 4.2).

The user collection holds the account information of the network users. The user's email address is the primary key.

The document collection holds the metadata of the documents that users have posted to the network. The document contents are not posted to the server. The *hash* field is the unique SHA256 hash value derived from the document's contents. The UID field is a foreign key to the user collection to link the document to the owner.

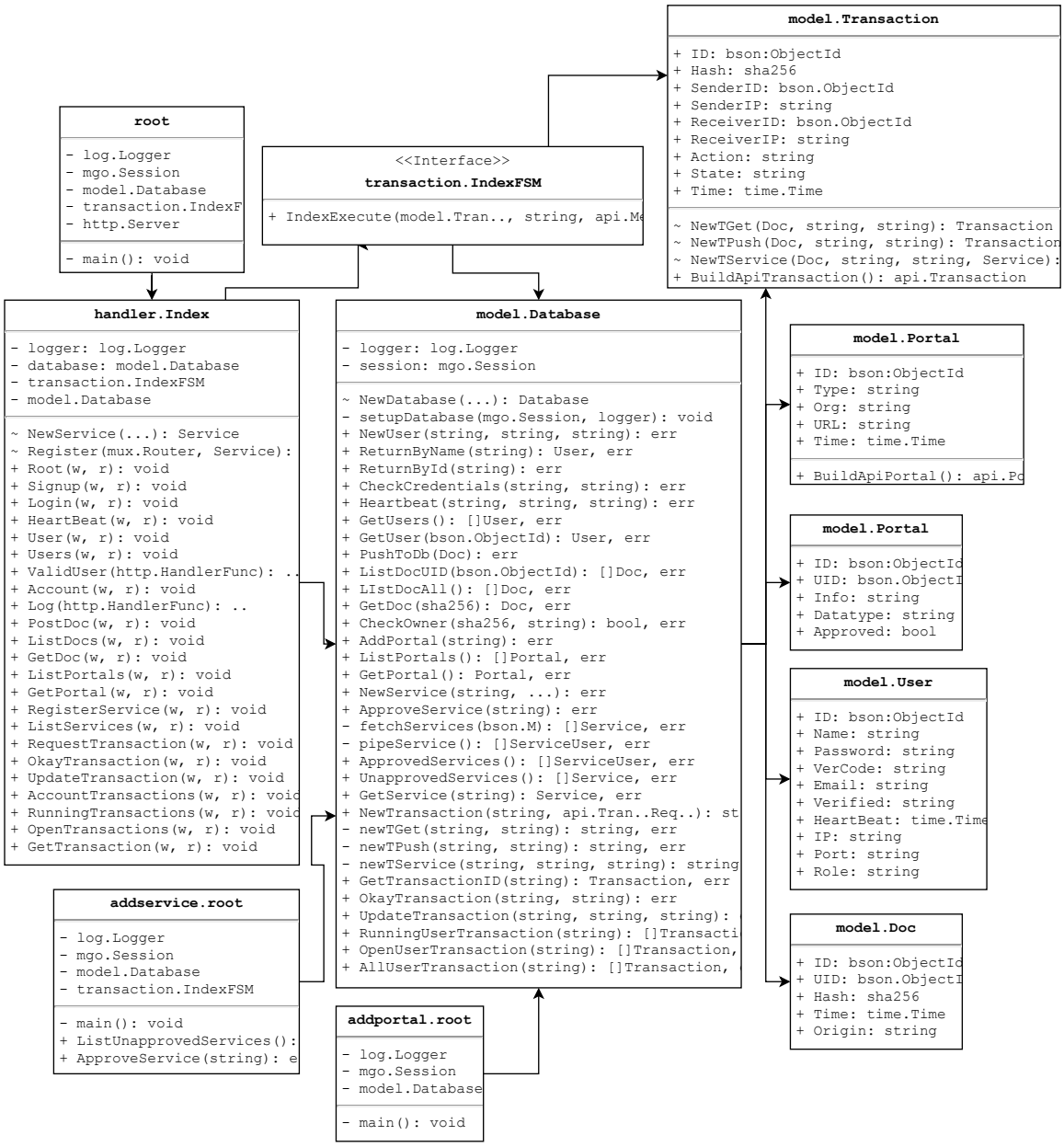


Fig. 4.1.: Index Server Classes

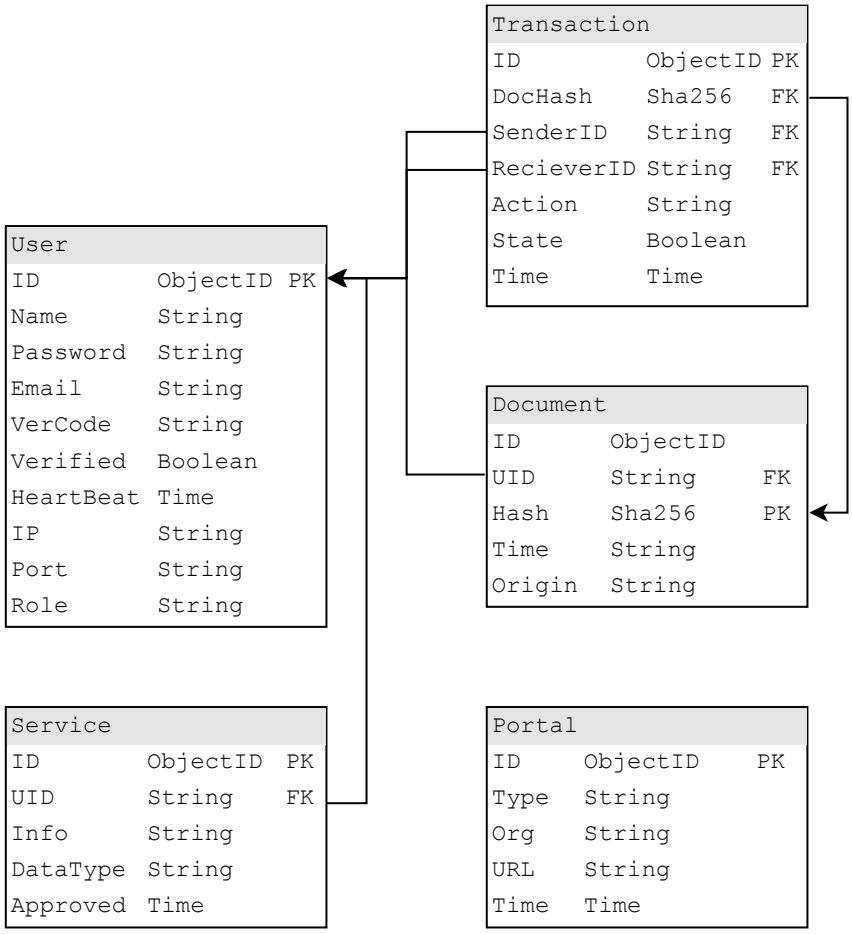


Fig. 4.2.: Database Collection

The transaction collection holds the network transactions. Each transaction record contains the state, requested data, and actors in a transaction. The actors, *senderID* and *receiverID*, are foreign keys to the user database. The *hash* field is a foreign key to the requested document. The ID field is a unique key assigned by the database to identify a transaction, allowing users to reissue a transaction without duplication.

The service collection holds the approved and unapproved services in the network. Network clients can only interact with approved services. A service record contains information on the data needed by the service and the foreign key UID of the linked peer who operates the service.

The portal collection holds connection information and details on FHIR portals provided by health providers.

4.1.2 API and Handlers

The Index server provides its services as a REST API. REST is a design pattern for network services in which resources are URL paths that are called using the standard HTTP routines (e.g. *GET* and *POST*). The network resources are organized as a hierarchy (Fig. 4.3), for example, the `/network` path includes resources for logged in clients.

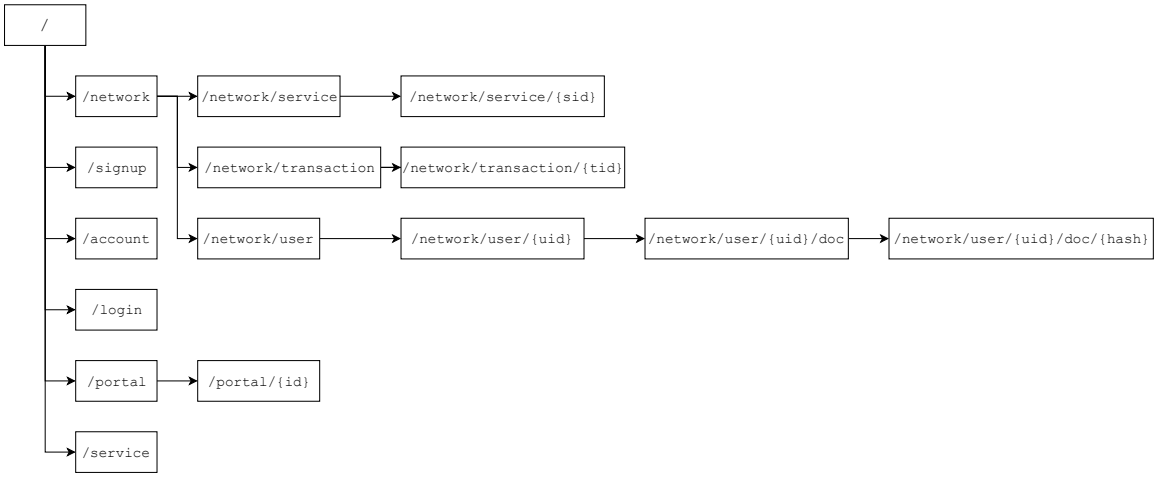


Fig. 4.3.: REST Access

Contacting a REST endpoint invokes a series of HTTP handlers. The handlers evaluate a request and interact with the database. The full communication details and routines between clients and the index server are in section 4.4.

4.2 Peer Client

A high-level diagram of the peer client system components and their interactions is in Figure 4.4.

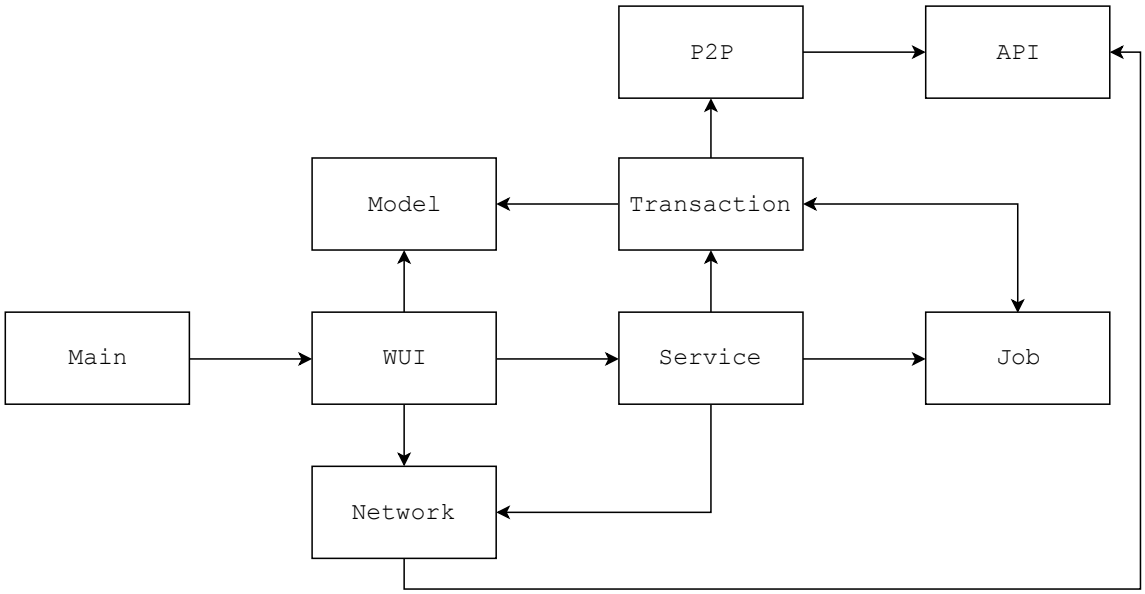


Fig. 4.4.: Peer Client Subsystems

The Peer Client database is implemented in MongoDB. It has two collections: *Record* and *PeerRecord* (Fig. 4.5). Each collection is independent. The record collection holds the set of FHIR records pulled from portal servers. Each record consists of the FHIR record data following the FHIR V3 standard and the metadata which is generated when adding the record to the database. The FHIR V3 golang framework is taken from the Intervention Engine [103], which is licensed under the Apache license, Version 2.0 [14].

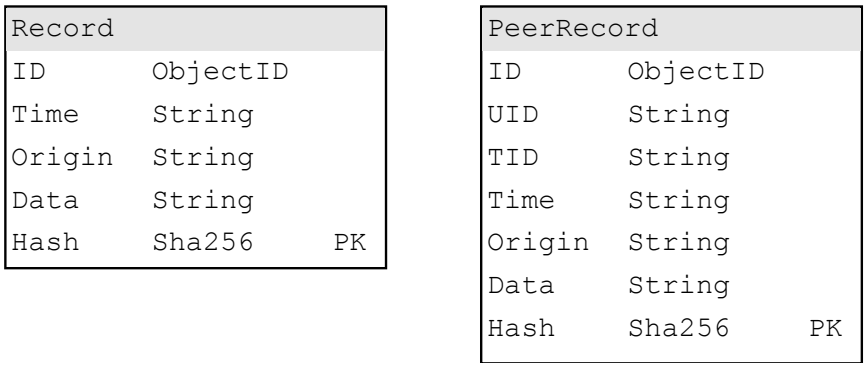


Fig. 4.5.: Peer Client Collections

New records are added to the database with the *Database.NewRecord* function (Alg. 4.1). This function generates the metadata for the record to be added, including: the origin, date, and a SHA256 hash of the data. The database method *StoreRecord* attempts to push the target record to the database.

```
1 func NewRecord(time time.Time, url string, resource interface{}) (Record,
    error) {
2         var record Record
3         buffer := json.Marshal(resource)
4         hash_byte := sha256.Sum256(buffer)
5         session := d.rootSession.Copy()
6         defer session.Close()
7         c := session.DB("client").C("record")
8         c.Insert(record)
9 }
```

Algorithm 4.1: New Record

The peer record collection holds the records aquired from other peers through transactions. This collection is similar to *record* but includes the *UID* to indicate the original owner and the *origin* field holds the ID of the transaction that transferred the record.

The Peer Client accesses the network through the Network Client (Fig. 4.6). The Network Client wraps the network calls and keeps the client logged into the network. The Network Client's interactions with the network are discussed in Section 4.4.

The client GUI is implemented as a Web User Interface (WUI). The user must start the client and then use a browser to connect to the WUI running on localhost. The application is laid out as HTML pages (Fig. 4.7), which are served to the user.

The root page provides access to the initial pages of registration, login, local records, and help. The registration page allows a user to register using a form (Fig. 4.8). After submitting the form, the WUI parses the form and invokes the network client. If the registration request succeeds, the user is redirected to the login page. On the login page, the client can submit their credentials (Fig. 4.9). The submitted form is parsed and a new network client is launched to connect to the network. This connected network client is necessary for network page access. It will run in the background until the user logs out. If the network client is not running, the WUI will redirect the user back to the login page.

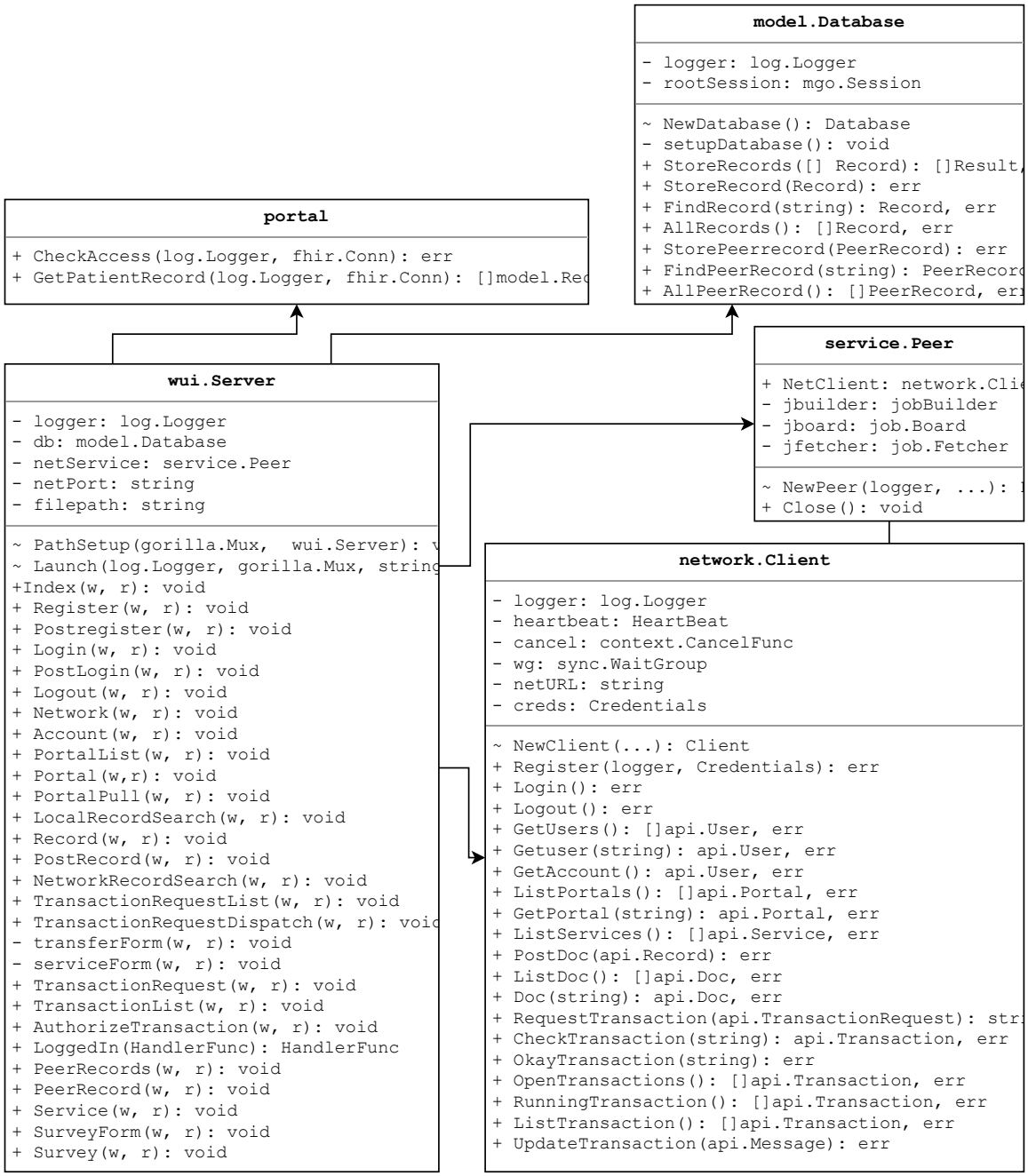


Fig. 4.6.: Network and WUI Classes

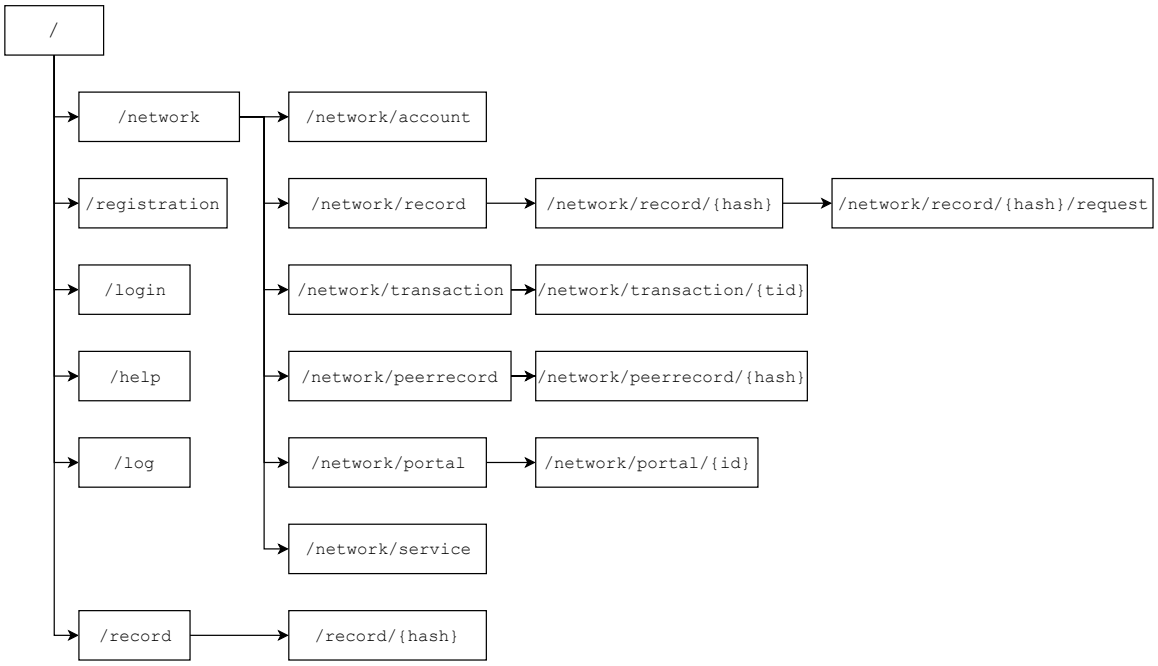


Fig. 4.7.: WUI Paths

Fig. 4.8.: Registration

Fig. 4.9.: Login

The local records page renders a table of the health records in the local database (Alg. 4.2). Each row shows the record metadata . A client can click on any individual record to show the content of the record (Fig. 4.10) . In addition, if the user is logged into the network, the user can request that the document metadata be posted to the network.

```
1 func (d *Database) AllRecords() ([]Record, error) {
2         session := d.rootSession.Copy()
3         c := session.DB("client").C("record")
4         var records []Record
5         c.Find(nil).All(&records)
6         return records, nil
7 }
```

Algorithm 4.2: Local Records

Fig. 4.10.: Record

The network menu lists pages that require network access. It is the base menu to other network actions. Accessing the account page triggers the network client to retrieve the client's account information from the index server. The returned information is rendered to the client (Fig. 4.11).

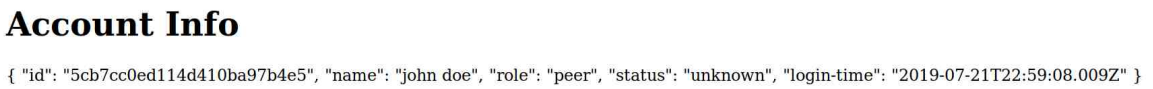
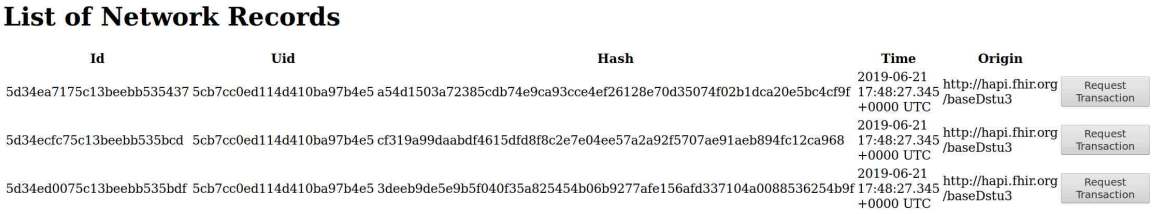


Fig. 4.11.: Account Page

The network document page triggers the network client to retrieve the list of documents in the network. The documents are rendered as a table of each record's metadata (Fig. 4.12(a)). Each document is a link, which when accessed, invokes the network to retrieve the list of network services. The transaction options are rendered to the client and the user can choose a transaction option from the list (Fig. 4.12(b)). After selection, a request form for the transaction (Fig. 4.12(c)) is presented. Once the form is completed, the client sends the request to the network. If the request is successful, the client is redirected to the transaction page. Otherwise, the client is receives a network error.



List of Network Records

Id	Uid	Hash	Time	Origin	
5d34ea7175c13beebb535437	5cb7cc0ed114d410ba97b4e5	a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f	2019-06-21 17:48:27.345 +0000 UTC	http://hapi.fhir.org/baseDstu3	Request Transaction
5d34ecfc75c13beebb535bcd	5cb7cc0ed114d410ba97b4e5	cf319a99daabdf4615dfd8f8c2e7e04ee57a2a92f5707ae91aeb894fc12ca968	2019-06-21 17:48:27.345 +0000 UTC	http://hapi.fhir.org/baseDstu3	Request Transaction
5d34ed0075c13beebb535bdf	5cb7cc0ed114d410ba97b4e5	3deeb9de5e9b5f040f35a825454b06b9277afe156afd337104a0088536254b9f	2019-06-21 17:48:27.345 +0000 UTC	http://hapi.fhir.org/baseDstu3	Request Transaction

(a) Document List

Transaction Request

Record Metadata

{5d34ea7175c13beebb535437 5cb7cc0ed114d410ba97b4e5 a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f 2019-06-21 17:48:27.345 +0000 UTC http://hapi.fhir.org/baseDstu3}

Basic Transactions

Choose an option

- Transfer
- Push

Submit

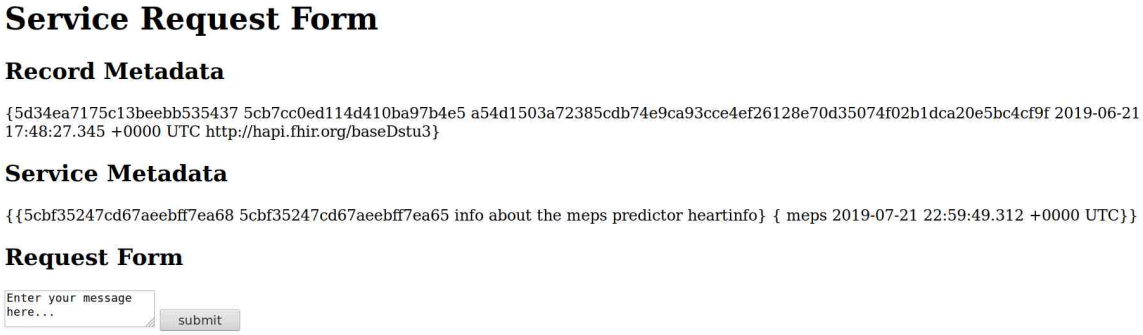
Services List

Service List

	Id	Name	Uid	Heartbeat	Info	DataType
○	5cbf35247cd67aeebff7ea68	meps	5cbf35247cd67aeebff7ea65	2019-07-21 22:59:39.313 +0000 UTC	info about the meps predictor	heartinfo

Submit

(b) Transaction List



Service Request Form

Record Metadata

{5d34ea7175c13beebb535437 5cb7cc0ed114d410ba97b4e5 a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f 2019-06-21 17:48:27.345 +0000 UTC http://hapi.fhir.org/baseDstu3}

Service Metadata

{{5cbf35247cd67aeebff7ea68 5cbf35247cd67aeebff7ea65 info about the meps predictor heartinfo} { meps 2019-07-21 22:59:49.312 +0000 UTC}}

Request Form

Enter your message here...

submit

(c) Transaction Request

Fig. 4.12.: Network Documents and Transaction Request

The transaction page triggers the network client to retrieve the transactions in which the client is a participant. The results are rendered as a table for the user (Fig. 4.13).

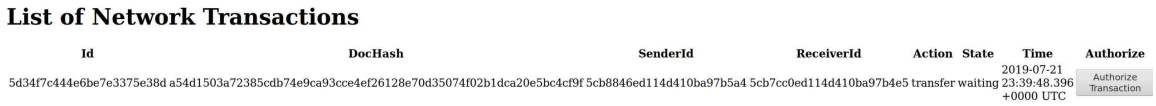


Fig. 4.13.: Transaction Page

The user can view the list of records that have been transferred to the client on the peer record page. The peer record page renders the records in the `PeerRecord` collection as a table (Fig. 4.14).

Fig. 4.14.: Peer Records Table

Other network pages that the client can access include the service page and the survey page. The service page lists the active services in the network. The list is generated by a network client request to the index server. The survey page allows the client to fill out the equivalent MEPS survey data for the MEPS Hypertension service.

4.2.1 FHIR Client

An access of the portal page triggers a network request to fetch the list of portals maintained by the index server. The returned list is rendered to the client as a table. Each portal can be further accessed to make a request. A portal request allows the client to fill out a form to request records in accordance with the FHIR protocol. The form is submitted, parsed, and a FHIR query (Fig. 4.15) is constructed. The FHIR query is then sent to the target server (Alg. 4.3). The returned FHIR records are added to the database, except in cases where the record is not new.

```
http://hostname/Patient/{user}/$everything
```

Fig. 4.15.: FHIR Query

```
1  func GetPatientRecord(logger *log.Logger, conn fhir.Connection) ([]model.
       Record, error) {
2          httpClient := &http.Client{Timeout: time.Second * 10}
3          fhirClient := fhir.NewClient(logger, httpClient)
4          entries := fhirClient.FetchPatientRecord(conn)
5          time := time.Now()
6          records := make([]model.Record, len(entries))
7          for i, entry := range entries {
8                 records[i] = model.NewRecord(time, conn.Url, entry.Resource)
9          }
10         return records, err
11 }
```

Algorithm 4.3: Fetch FHIR Records

4.2.2 P2P Subsystem

The P2P subsystem manages the transactions with other peers (Fig. 4.16). It operates independently from the user's input, with the user only controlling the launch and closing of the system. The P2P subsystem acts as a client and server, pulling and carrying out transactions from the network and serving transaction requests from other peers.

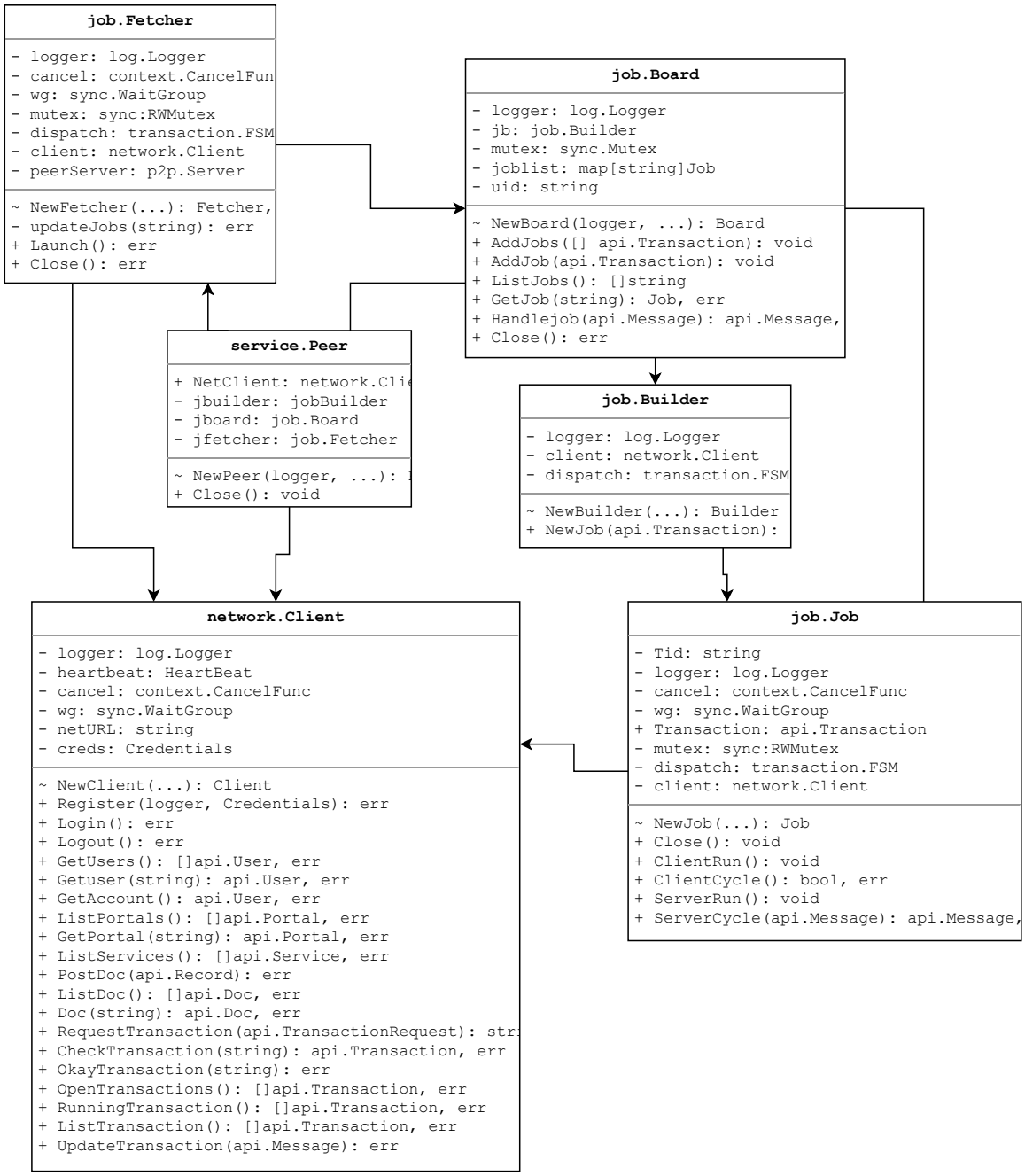


Fig. 4.16.: Peer-to-Peer Classes

The system polls the index server for new transactions with the *fetcher.updateJobs* routine (Alg. 4.4). These transactions are compared against the already running set of transactions, termed *jobs* (Fig. 4.17). If the transaction is new, then a new job is created by the factory *JobBuilder* class and stored in a hash table of jobs. Each job is given its own thread to prevent blocking the main client.

```
...
ticker := time.NewTicker((api.HeartBeatInterval / 2) * time.Second)
for {
        select {
        case <-ctx.Done():
                ps.logger.Println("ending transaction update")
                return
        case <-ticker.C:
                err := ps.updateJobs(uid)
        }
}
...
func (ps *Fetcher) updateJobs(uid string) error {
        transactions := ps.netclient.RunningTransactions()
        err = ps.jboard.AddJobs(transactions)
        return nil
}
```

Algorithm 4.4: Fetch Jobs

A *job*'s operation is defined by the user's role in the transaction. A user that is the *requester* will try to execute client-side transactions, while a *receiver* will wait for a client to connect before attempting a transition. A transition is a cycle of the current state of the protocol and any messages. Both operations implement a loop (Alg. 4.5) to poll the server for transaction updates, attempt transitions, and for sending updates back to the network or caller (Alg. 4.6 and 4.7). After polling the network for the transaction, the transaction's state is examined. If the transaction state is in the *final* state, the job loop is exited. Otherwise a state transition is performed. The job sends the returned transition messages to the index server and if a server job, the client.

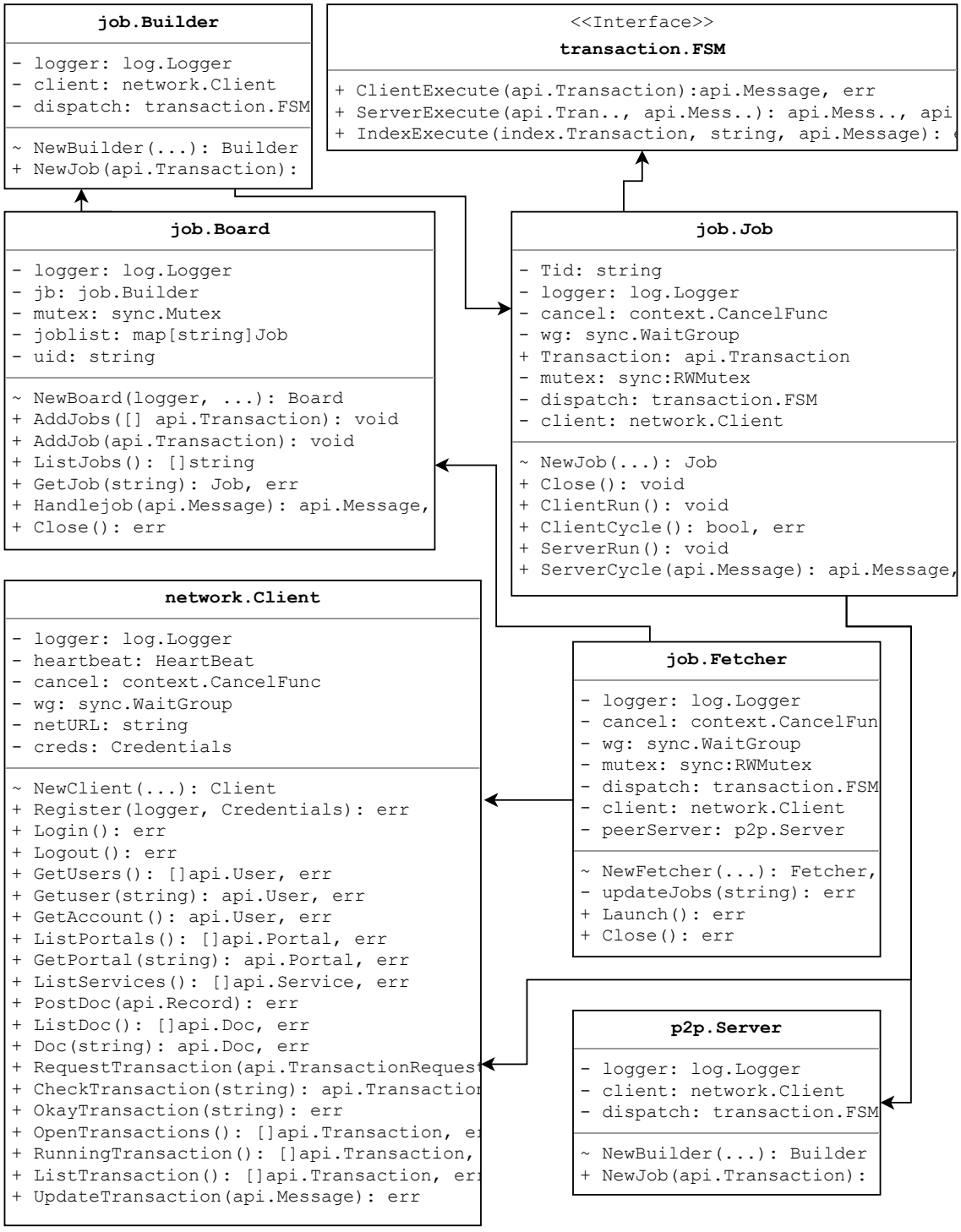


Fig. 4.17.: Job Classes

```
...
ticker := time.NewTicker(1 * time.Second)
for {
        select {
        case <-ctx.Done():
                return
        case <-ticker.C:
                run := j.ClientCycle() ‘‘or’’ j.ServerCycle()
                if !run {
                        return
                }
        }
}
...
```

Algorithm 4.5: Job Cycle

```
func (j *Job) ClientCycle() (bool, error) {
        update := j.client.CheckTransaction(j.Tid)
        if update.State == "final" {
                return false, nil
        }
        temp := j.dispatch.ClientExecute(update)
        j.client.UpdateTransaction(temp)
}
```

Algorithm 4.6: Client Job

```
func (j *Job) ServerCycle(message *api.Message) (*api.Message, error) {
        update := j.client.CheckTransaction(j.Tid)
        if update.State == "final" {
                return nil, errors.New("final")
        }
        sender, index := j.dispatch.ServerExecute(update, message)
        j.client.UpdateTransaction(index)
        return sender, err
}
```

Algorithm 4.7: Server Job

4.2.3 TCP Library

Peers use a small library to pass messages based on TCP (Fig. 4.18). The TCP library uses a standard *Request-Response* pattern utilizing JSON (Alg. 4.8, 4.9) which can pass arbitrary data. The client side is shown in algorithm 4.10 and the server side and handler are shown in algorithms 4.11 and 4.12. Each client registers a p2p listener which dispatches (Alg. 4.13) to the correct job when a peer connects. If the job is not present the connection is terminated.

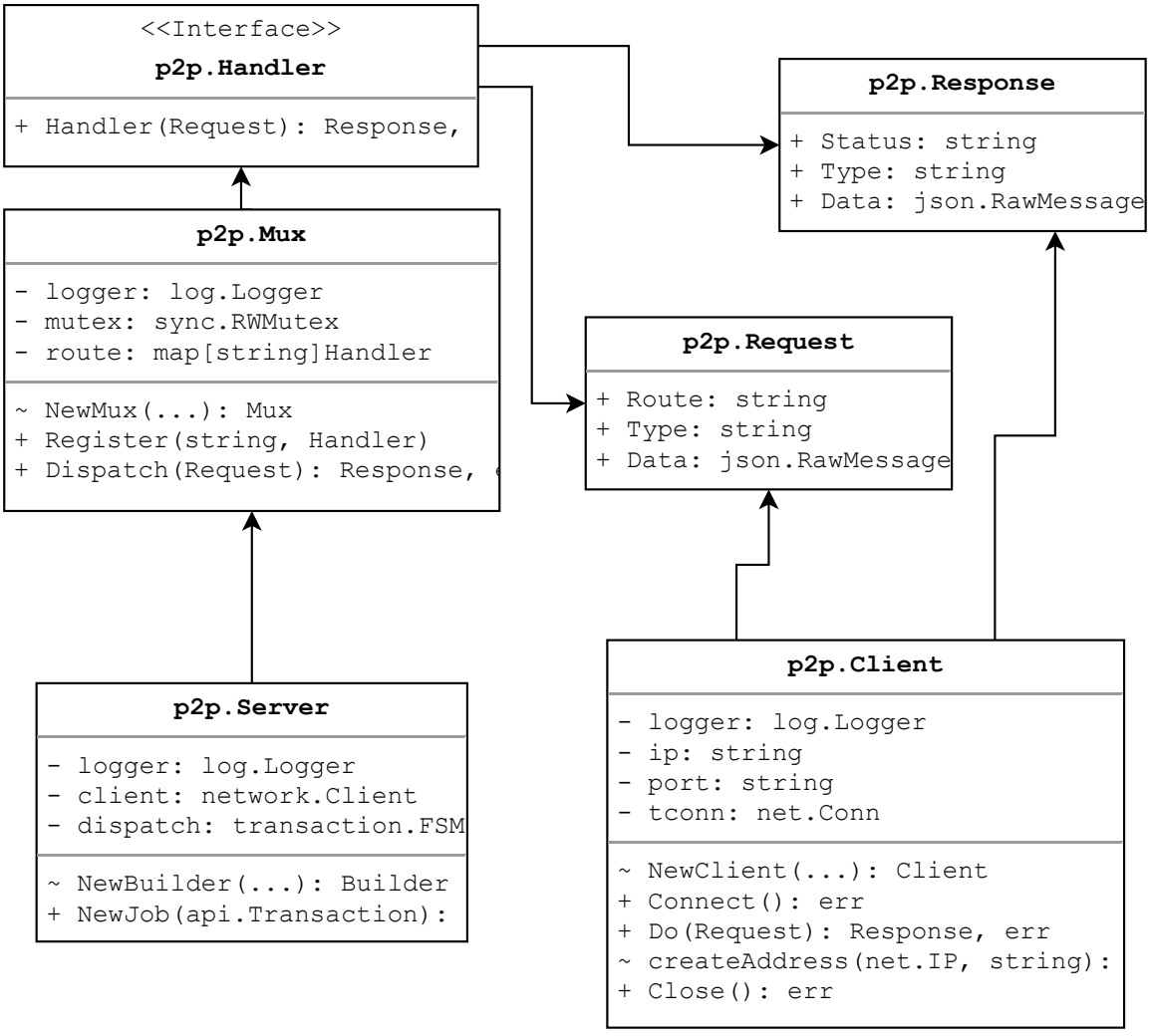


Fig. 4.18.: TCP library

```
type Request struct {
        Route string
        Type string
        Data json.RawMessage
}
```

Algorithm 4.8: Request

```
type Response struct {
        Status string
        Type string
        Data json.RawMessage
}
```

Algorithm 4.9: Response

```
        ...
        address := createAddress(c.ip, c.port)
        tconn, err := net.DialTimeout("tcp", address, IdleTimeout)
        c.tconn = conn
        ...
func (c *Client) Do(request *Request) (*Response, error) {
        encoder := json.NewEncoder(c.tconn)
        decoder := json.NewDecoder(c.tconn)
        err := encoder.Encode(request)
        checkErr(err) // if err close connection, check if OpErr
        var response Response
        err = decoder.Decode(&response)
        checkErr(err) // if err close connection, check if OpErr
        return &response, nil
}
```

Algorithm 4.10: Client Side TCP

```
func (c *conn) serve(cid int) {
        defer func() {
                c.tconn.Close()
        }()
        decoder := json.NewDecoder(c.tconn)
        encoder := json.NewEncoder(c.tconn)
        for {
                var request Request
                err := decoder.Decode(&request)
                checkErr(err) // close connection if err, check err.type
                response := c.mux.Dispatch(&request)
                deadline := time.Now().Add(IdleTimeout)
                c.tconn.SetDeadline(deadline)
                err = encoder.Encode(response)
                checkErr(err) // close connection if err, check err.type
        }
}
```

Algorithm 4.11: Serve Connection

```
1  func (s *Server) ListenAndServe() error {
2         listener := net.Listen("tcp", ":"+s.port)
3         s.listener = listener
4         go func() {
5                 var cLock sync.Mutex
6                 connMap := make(map[int]*conn)
7                 cid := 0
8         OuterLoop:
9                 for {
10                        tconn, err := s.listener.Accept()
11                        checkErr(err) // break to OuterLoop if closed else
                              contine
12                        tconn.SetDeadline(time.Now().Add(IdleTimeout))
13                        c := newConn(tconn, s.logger, s.mux)
14                        cLock.Lock()
15                        cid++
16                        id := cid
17                        connMap[cid] = c
18                        cLock.Unlock()
19                        s.wg.Add(1)
20                        go func(conn_id int) {
21                               defer s.wg.Done()
22                               c.serve(cid)
23                               cLock.Lock()
24                               delete(connMap, conn_id)
25                               defer cLock.Unlock()
26                        }(id)
27                }
28                for _, v := range connMap {
29                        v.tconn.Close()
30                }
31                s.wg.Wait()
32                s.result <- nil
33        }()
34        return nil
35 }
```

Algorithm 4.12: Listen for Connection

```
1  func TransactionHandler(logger *log.Logger, board *Board) p2p.Handler {
2         return func(r *p2p.Request) (*p2p.Response, error) {
3                 var message api.Message
4                 json.Unmarshal(r.Data, &message)
5                 tid := message.Tid
6                 response := board.HandleJob(&message)
7                 jsonData := json.Marshal(response)
8                 return &p2p.Response{"ok", "message", jsonData}, nil
9         }
10 }
```

Algorithm 4.13: Server Dispatch

4.3 Service Client

The Service Client allows the execution of approved services for the network (Fig. 4.19). It implements similar functionalities to the Peer Client with several key differences. A service client only implements the Transaction Service Protocol. The Service Client has no local database. It only invokes the approved program. Additionally, on regular intervals it pulls all unapproved service transaction and approves them (Alg. 4.14). This approach allows the service client to remain independent of the index service while providing the network the necessary services on demand.

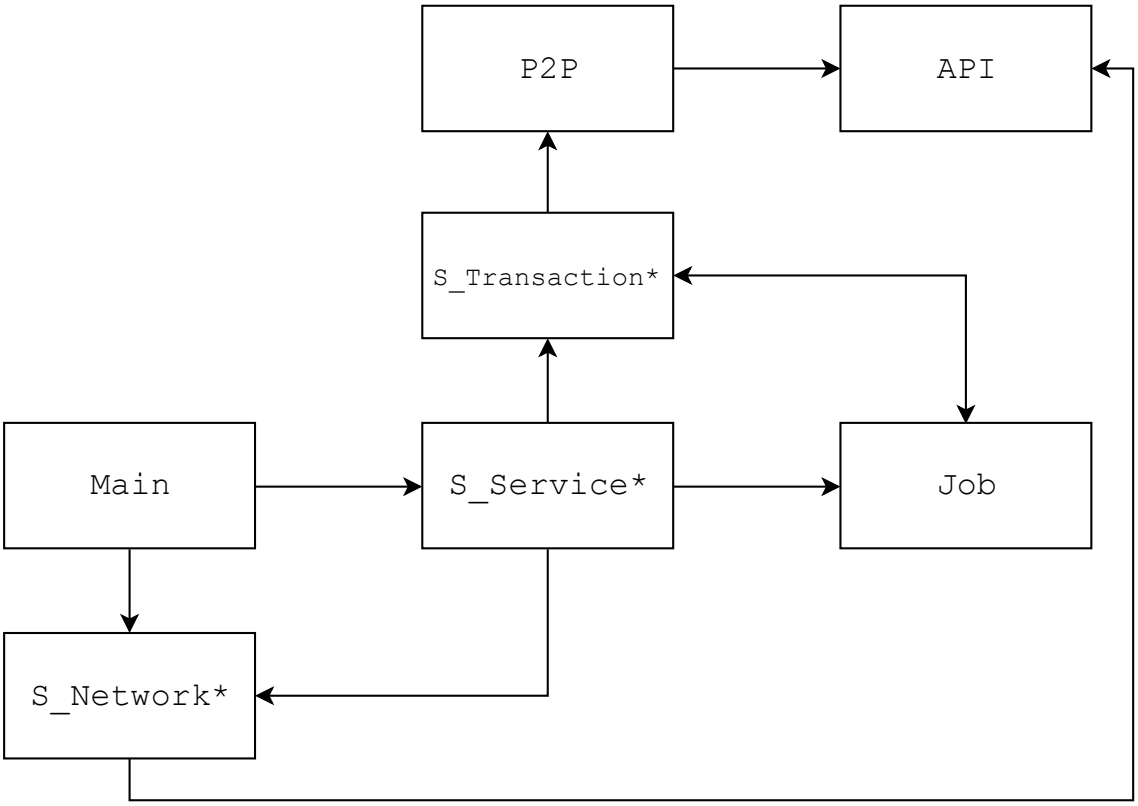


Fig. 4.19.: Service Client Subsystems

∗ Subsystem was modified from client version.

```
1 func (a *Approve) approveTransactions() error {
2         transactions, err := a.netclient.OpenTransactions()
3         handleErr(err)
4         for _, t := range transactions {
5                 err := a.netclient.OkayTransaction(t.ID)
6                 handleErr(err)
7         }
8         return nil
9 }
```

Algorithm 4.14: Approval All Transactions

A service registers with the network using the Network Client's *ServiceRegister* function. This sends a service registration request to the index server. The index server creates a new service and a peer database record using the received information but leaves the service unapproved. The network admin can examine the service requests and approve them.

4.4 Network

The Peer and Service clients connect and interact with the Network through the Index Server's by invoking the appropriate REST API (Fig. 4.3) of the Index Server. These API are discussed next.

4.4.1 Registration

A peer client can register for an account by calling the Network Client function *Register* (Alg. 4.15). The Network Client creates the JSON registration packet (Alg. 4.16) and sends it to the Index Server endpoint `/signup`. The index server parses and validates the request (Alg. 4.17). If the client is new and the request is valid, a new User is created and pushed to the database (Alg. 4.18). Any subsequent network actions require that the user post their login information as part of the HTTP basic authentication header.

```
1  func Register(logger *log.Logger, netURL string, name string, pass string,
       email string) error {
2          var httpClient = &http.Client{Timeout: timeout}
3          signup := api.Signup{name, pass, email}
4          jsonReq, err := json.Marshal(signup)
5          url := netURL + api.SignUpRoute
6          request, err := http.NewRequest("POST", url, bytes.NewBuffer(jsonReq
               ))
7          request.Header.Set("Content-Type", "application/json")
8          response, err := httpClient.Do(request)
9          if response.Status != "200 OK" {
10                 return errors.New(response.Status)
11         }
12         return nil
13 }
```

Algorithm 4.15: Client Register

```
1  type Signup struct {
2          Name string `json:"name"`
3          Password string `json:"password"`
4          Email string `json:"email"`
5  }
```

Algorithm 4.16: Signup JSON

```
1  func (s *Service) Signup(w http.ResponseWriter, r *http.Request) {
2          body, err := ioutil.ReadAll(r.Body)
3          var signup api.Signup
4          err = json.Unmarshal(body, &signup)
5          err = s.database.NewUser(signup.Name, signup.Password, signup.Email)
6          w.WriteHeader(http.StatusOK)
7  }
```

Algorithm 4.17: New User Handler

```
1  func (d *DataBase) NewUser(name string, password string, email string)
       error {
2          session := d.rootSession.Copy()
3          user := User{Name, Password, email, role}
4          c := session.DB("index").C("user")
5          err := c.Insert(user)
6          return err
7  }
```

Algorithm 4.18: New User Database

4.4.2 Login

The `Login` function logs the client into the network. It spawns a background process that continuously pings the API heartbeat endpoint within the network's timeout period of 10 seconds (Alg. 4.19).

```
1  func (client *Client) launchHeartBeat() {
2         var ctx context.Context
3         ctx, client.cancel = context.WithCancel(context.Background())
4         client.wg.Add(1)
5         go func(client *Client, ctx context.Context, status chan<- conn) {
6                 defer client.wg.Done()
7                 ticker := time.NewTicker(api.HeartBeatInterval * time.Second)
8                 for {
9                         select {
10                        case <-ctx.Done():
11                                return
12                        case <-ticker.C:
13                                checkedin, err := client.Heartbeat()
14                                var result conn
15                                checkErr(err) // set conn to result
16                                client.mutex.Lock()
17                                client.stat = result
18                                client.mutex.Unlock()
19                        }
20                }
21        }(client, ctx, client.status)
22 }
23
24 func (client *Client) Heartbeat() (bool, error) {
25        var httpClient = &http.Client{Timeout: timeout}
26        heartbeat := api.Heartbeat{client.credentials.Name, client.
              credentials.Password}
27        jsonReq, err := json.Marshal(heartbeat)
28        url := client.netURL + api.HeartBeatRoute
29        request, err := http.NewRequest("POST", url, bytes.NewBuffer(jsonReq
              ))
30        request.Header.Set("Content-Type", "application/json")
31        response, err := httpClient.Do(request)
32        if response.Status != "200 OK" {
33                return false, nil
34        }
35        return true, nil
36 }
```

Algorithm 4.19: Launch Heartbeat

When the index server receives a valid request (Alg. 4.20), it checks the IP of the client and the requested port. It then updates the client's user record with the new connection information (Alg. 4.21). A client is not considered part of the network if it fails to ping the network in the allotted time.

```
type Heartbeat struct {
        Name string `json:"name"`
        Password string `json:"password"`
        Port string `json:"port"`
}
```

Algorithm 4.20: Heartbeat JSON

```
func (s *Service) HeartBeat(w http.ResponseWriter, r *http.Request) {
        body, err := ioutil.ReadAll(r.Body)
        var heartbeat api.Heartbeat
        err = json.Unmarshal(body, &heartbeat)
        check, err := s.database.CheckCredentials(heartbeat.Name, heartbeat.
            Password)
        ip, port, err := net.SplitHostPort(r.RemoteAddr)
        err = s.database.HeartBeat(heartbeat.Name, heartbeat.Password, ip)
        w.WriteHeader(http.StatusOK)
}
```

Algorithm 4.21: Server Heartbeat

When the user elects to logout of the network, the close routine is invoked which shuts down the heartbeat process (Alg. 4.22).

```
func (client *Client) Logout() error {
        client.cancel()
        client.wg.Wait()
        client.stat = conn{errors.New("no heartbeat thread running"), false}
        return nil
}
```

Algorithm 4.22: Close Client

4.4.3 Resource Lookup

The peer can lookup network resources from the index server using various client network functions and their corresponding REST endpoints. These functions are similar in implementation and a generic form of each is provided in algorithms 4.23, 4.24, and 4.25. Resource lookup can return details on: Users, Documents, Services, Portals, and Transactions.

```
1  func (client *Client) ListData() ([]api.Data, error) {
2         var httpClient = &http.Client{Timeout: timeout}
3         url := client.netURL + api.PortalsRoute
4         request, err := http.NewRequest("GET", url, nil)
5         request.SetBasicAuth(client.credentials.Name, client.credentials.
              Password)
6         response, err := httpClient.Do(request)
7         if response.Status != "200 OK" {
8                 return nil, errors.New("failed request")
9         }
10        body, err := ioutil.ReadAll(response.Body)
11        var data api.Data
12        err = json.Unmarshal(body, &data)
13        return pb.Entries, nil
14 }
```

Algorithm 4.23: Client Resource Request

```
1  func (s *Service) GetDataList(w http.ResponseWriter, r *http.Request) {
2         datas, err := s.database.GetDataList()
3         apiData := make([]api.Data, len(datas))
4         for i, d := range datas {
5                 apiData[i] = d.BuildAPIData()
6         }
7         bundle := api.DataBundle{count, apiData}
8         jsonResp, err := json.Marshal(bundle)
9         w.Header().Set("Content-Type", "application/json")
10        w.Write(jsonResp)
11 }
```

Algorithm 4.24: Index Server Handler

```
1  func (ds *DataBase) GetDataList() ([]Data, error) {
2         session := ds.rootSession.Copy()
3         c := session.DB("index").C("data")
4         var data []Data
5         err := c.Find(nil).All(&data)
6         return datas, nil
7  }
```

Algorithm 4.25: Database Access

4.4.4 Posting Documents

The user can post documents to the network using the network client's *Post-Document* function (Fig. 4.26). The client constructs an HTTP *POST* request using the provided document metadata (Alg. 4.27) and sends it to the network `/network/document` endpoint (Alg. 4.26).

```
1  func (client *Client) PostDoc(record *model.Record) (bool, error) {
2         var httpClient = &http.Client{Timeout: timeout}
3         apiDoc := api.Doc{"", "", record.Hash, record.Time, record.Url}
4         jsonReq, err := json.Marshal(apiDoc)
5         url := client.netURL + api.DocsRoute
6         request, err := http.NewRequest("POST", url, bytes.NewBuffer(jsonReq
              ))
7         request.SetBasicAuth(client.credentials.Name, client.credentials.
              Password)
8         request.Header.Set("Content-Type", "application/json")
9         response, err := httpClient.Do(request)
10        if response.Status != "200 OK" {
11                return false, nil
12        }
13        return true, nil
14 }
```

Algorithm 4.26: Client Post Document

```
1  type Doc struct {
2         ID string `json:"id,omitempty"`
3         UID string `json:"userid,omitempty"`
4         Hash [sha256.Size]byte `json:"hash,omitempty"`
5         Time time.Time `json:"time,omitempty"`
6         Origin string `json:"origin,omitempty"`
7  }
```

Algorithm 4.27: Record JSON

The server parses and validates the request. If valid, the server pushes the document along with the owner ID to the `document` collection (Alg. 4.28). The `document` collection is unique on the hash of the document. The result is returned to the client.

```
1  func (s *Service) PostDoc(w http.ResponseWriter, r *http.Request) {
2          body, err := ioutil.ReadAll(r.Body)
3          var postdoc api.Doc
4          err = json.Unmarshal(body, &postdoc)
5          username, _, ok := r.BasicAuth()
6          user, err := s.database.ReturnByName(username)
7          doc := model.Doc{Hash: postdoc.Hash, UID:user.ID, Time: postdoc.Time
               , Origin: postdoc.Origin}
8          err = s.database.PushToDb(&doc)
9          w.Header().Set("Content-Type", "text/plain; charset=utf-8")
10         w.WriteHeader(http.StatusOK)
11 }
```

Algorithm 4.28: Server Add Document

4.4.5 Requesting and Approving Transactions

The `/network/transaction` path provides peers access to transactions. A client requests a transaction by sending a *POST* request with the JSON *TransactionRequest* (Alg. 4.29) to the index server (Alg. 4.30). The server verifies the request, and if valid, creates a new transaction and sets it to the *pending* state in the database. The new transaction ID is returned to the client.

```
1  type TransactionRequest struct {
2          Action string `json:"action"`
3          ServiceID string `json:"serviceId,omitempty"`
4          UserID string `json:"userId,omitempty"`
5          DocumentID string `json:"docid"`
6  }
```

Algorithm 4.29: Transaction Request JSON

```
1  func (client *Client) RequestTransaction(treq api.TransactionRequest) (
       string, error) {
2          var httpClient = &http.Client{Timeout: timeout}
3          jsonReq, err := json.Marshal(treq)
4          uri := client.netURL + api.TransactionsRoute
5          request, err := http.NewRequest("POST", uri, bytes.NewBuffer(jsonReq
               ))
6          request.SetBasicAuth(client.credentials.Name, client.credentials.
               Password)
7          request.Header.Set("Content-Type", "application/json")
8          response, err := httpClient.Do(request)
9          if response.Status != "200 OK" {
10                return "", nil
11         }
12         body, err := ioutil.ReadAll(response.Body)
13         var t api.Transaction
14         err = json.Unmarshal(body, &t)
15         return t.ID, nil
16 }
```

Algorithm 4.30: Request Transaction

A client can request the list of all waiting transactions by posting a *GET* request to`/network/transaction` endpoint with the query string parameter of `state=waiting`. The server checks that the request is valid and queries the database for the transactions in which the client is the *receiver* and the state is *waiting* (Alg. 4.31). The resulting list of transactions (Alg. 4.32) is returned to the client.

```
1  func (ds *DataBase) OpenUserTransaction(username string) ([]Transaction,
       error) {
2          user, err := ds.ReturnByName(username)
3          uid := user.ID
4          session := ds.rootSession.Copy()
5          c := session.DB("index").C("transaction")
6          var transactions []Transaction
7          query := bson.M{"$and": []bson.M{bson.M{"$or": []bson.M{bson.M{"
               sender": uid},bson.M{"receiver": uid},},}, bson.M{"state": bson.
               M{"$eq": "waiting"}},},}
8          err = c.Find(query).All(&transactions)
9          return transactions, nil
10 }
```

Algorithm 4.31: Open Transactions

```
type Transaction struct {
        ID string `json:"id"`
        DocHash string `json:"dochash"`
        SenderID string `json:"sender"`
        SenderIP string `json:"senderip"`
        ReceiverID string `json:"receiver"`
        ReceiverIP string `json:"receiverip"`
        Port string `json:"receiverport"`
        Action string `json:"action"`
        State string `json:"pending"`
        Time time.Time `json:"time"`
}
```

Algorithm 4.32: Transaction JSON

A client can approve a transaction by sending a *POST* request to the `/network/transaction/{TID}` endpoint. The `{TID}` path variable refers to the transaction ID. The server validates the user and updates the transaction state from *pending* to the next protocol state (Alg. 4.33). If the transaction is not in the *waiting* state, an error is returned to the client.

```
func (ds *DataBase) OkayTransaction(username string, tid string) error {
        user, err := ds.ReturnByName(username)
        uid := user.ID.Hex()
        session := ds.rootSession.Copy()
        c := session.DB("index").C("transaction")
        sel := bson.M{"$and": []bson.M{bson.M{"_id": bson.ObjectIdHex(tid)},
            bson.M{"receiver": bson.ObjectIdHex(uid)},}}
        upd := bson.M{"$set": bson.M{"state": "transfer"}}
        err = c.Update(sel, upd)
        return nil
}
```

Algorithm 4.33: Approve Transaction

4.5 Transaction

The transaction begins processing after the *receiver* approves. The transactions classes are shown in Figure 4.20.

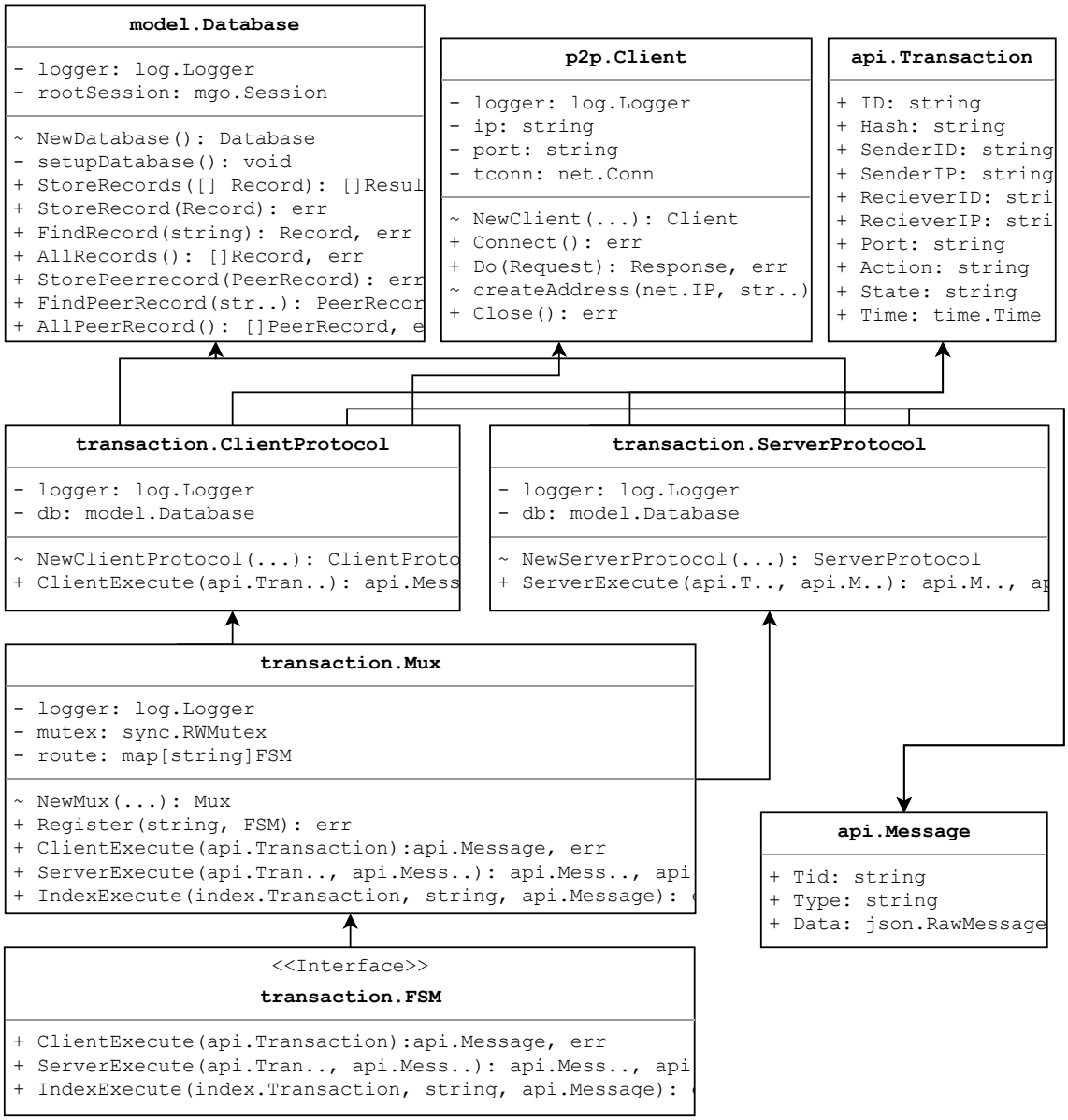


Fig. 4.20.: Transaction Process

A client requests the list of running transactions by posting a *GET* request to the `/network/transaction` path with the query string parameter of `state=running`. The network server validates the request and then creates two MongoDB pipelines to pull the list of running transactions in which the user is participating (Alg. 4.34). The pipelines are necessary to join the connection information from the `User` collection with the results from `transaction` collection. The two pipelines (Alg. 4.35) are separated by the clients role in the transaction, one for *requester* and another for the *receiver*. The two results are returned to the user as a single list of transactions. The clients add the new transactions to their respective job pools.

```
1  func (ds *DataBase) RunningUserTransaction(username string) ([]Transaction,
       error) {
2          user, err := ds.ReturnByName(username)
3          uid := user.ID
4          session := ds.rootSession.Copy()
5          c := session.DB("index").C("transaction")
6          var senderTransactions []Transaction
7          var recieverTransactions []Transaction
8          open := bson.M{"$match": bson.M{"state": bson.M{"$nin": []string{"
               final", "waiting"}}}}
9          senderID := bson.M{"$match": bson.M{"sender": uid}}
10         ljReceiverID := bson.M{"$lookup": bson.M{"from": "user", "localField
               ": "receiver", "foreignField": "_id", "as": "out"}}
11         combine := bson.M{"$project": bson.M{"_id": 1, "sender": 1, "
               receiver": 1, "receiverip": bson.M{"$arrayElemAt": []interface
               {}{"$out.ip", 0}}, "dochash": 1, "state": 1, "time": 1}}
12         receiverID := bson.M{"$match": bson.M{"receiver": uid}}
13         senderPipe := []bson.M{senderID, open, ljReceiverID, combine}
14         receiverPipe := []bson.M{receiverID, open}
15         err = c.Pipe(receiverPipe).All(&recieverTransactions)
16         err = c.Pipe(senderPipe).All(&senderTransactions)
17         transactions := append(recieverTransactions, senderTransactions...)
18         return transactions, nil
19 }
```

Algorithm 4.34: Running Transactions

```
1  db.transaction.aggregate([{$facet: {"sender" : [{$match: {receiver:
       ObjectId($UserId) }},{ $lookup: { from: "user", localField: "receiver",
        foreignField: "_id", as: "out"}},{"$project":{"_id" : 1, "sender" : 1,
        "receiver": 1, "recip": {"$arrayElemAt":["$out.ip",0]}}},{$match:{
       sender: ObjectId($UserId)}}],"reciever": [{$match: {sender:ObjectId(
       $UserID) }}]}}])
```

Algorithm 4.35: Example Pipeline Query

Clients in the *sender* role initiate a connection with the *receiver* and attempt a transition from the current state of protocol's state machine. Transitions in the network are modeled as a set of interfaces (Alg. 4.36). Each role uses the last known transaction data (Alg. 4.32) and, in the case of the server or index roles, any messages (Alg. 4.37). The job sends the returned transition execution messages to the client or the index server.

```
1 type ClientFSM interface {
2         ClientExecute(*api.Transaction) (*api.Message, error)
3 }
4 type ServerFSM interface {
5         ServerExecute(*api.Transaction, *api.Message) (*api.Message, *api.
              Message, error)
6 }
7 type IndexFSM interface {
8         IndexExecute(*index.Transaction, string, *api.Message) error
9 }
```

Algorithm 4.36: Transition Interfaces

```
1 type Message struct {
2         Tid string `json:"tid"`
3         Type string `json:"type"`
4         Data json.RawMessage `json:"data"`
5 }
```

Algorithm 4.37: Message JSON

To correctly dispatch to the protocol and state, a *mux* acts as a dispatch layer between the interface and the routines. A *mux* is a design pattern to register a path to an action. In this case, the mux allows for state machine transitions to be registered to a transaction protocol and state. A caller passes a path to a requested action and the mux returns the registered action. The implemented *mux* fulfills all three interfaces, allowing the *mux* to hide all concrete details from the caller. Each concrete routine is registered by a path idiom of `/{action}/{state}`. When the *mux* is invoked, it examines the transaction's state and action and dispatches it to the correct routine. The results are returned to the caller. If no routine is registered, an error is returned.

After executing a transition, the client tries to update the index server of the result by calling the Network Client *UpdateTransaction* function (Alg. 4.38) with the

Message JSON packet. The function posts the packet to the transaction endpoint. The network server checks if the transaction is valid, pulls the current transaction from the database, and attempts an index side transition with the transaction and message information. To ensure correctness, this transition attempt is similar to an atomic compare-and-swap operation [3]. If the update succeeds, then the transaction is moved to its next state.

```
1  func (s *Service) UpdateTransaction(w http.ResponseWriter, r *http.Request)
       {
2          body, err := ioutil.ReadAll(r.Body)
3          var update api.Message
4          err = json.Unmarshal(body, &update)
5          username, _, ok := r.BasicAuth()
6          tid := update.Tid
7          transaction, err := s.database.ReturnTransactionByID(tid)
8          err = s.indexFSM.IndexExecute(transaction, username, &update)
9          w.Header().Set("Content-Type", "text/plain; charset=utf-8")
10         w.WriteHeader(http.StatusOK)
11 }
12
13 func (ds *DataBase) UpdateTransaction(tid string, last string, update
       string) error {
14         session := ds.rootSession.Copy()
15         c := session.DB("index").C("transaction")
16         sel := bson.M{"_id": bson.ObjectIdHex(tid), "state": last}
17         upd := bson.M{"$set": bson.M{"state": update}}
18         err := c.Update(sel, upd)
19         return nil
20 }
```

Algorithm 4.38: Update Index Transaction

4.5.1 Get

A *Get* transaction transfers a record from the owner to the requester (Alg. 4.39). The sender initiates the connection with the TCP library and sends the transaction information to the receiver. The receiver checks the request, pulls the record from its local record database, and returns the result to the sender. The sender adds the returned document to its `peerrecord` collection. After the exchange the two clients send messages to the index server to finalize. The index server checks the messages and moves the transaction to the *final* state.

```
1  func (ct *ClientTransfer) ClientExecute(transaction *api.Transaction) (*api
       .Message, error) {
2          if transaction.State != "transfer" {
3                 return nil, errors.New("wrong state")
4          }
5          message := api.Message{Tid: transaction.ID}
6          m, err := json.Marshal(message)
7          request := p2p.Request{"/", "message", m}
8          ip := net.ParseIP(transaction.ReceiverIP)
9          port := transaction.Port
10         p2pClient := p2p.NewClient(ct.logger, ip, port)
11         err = p2pClient.Connect()
12         defer p2pClient.Close()
13         response, err := p2pClient.Do(&request)
14         record, err := getRecord(response)
15         peerRecord := &model.PeerRecord{transaction.ReceiverID, transaction.
               ID, record.Time, record.Url, record.Data, record.Hash}
16         err = ct.database.StorePeerRecord(peerRecord)
17         data, _ := json.Marshal(api.Basic{"done"})
18         indexUpdate := api.Message{transaction.ID, "basic", data}
19         return &indexUpdate, nil
20 }
21
22 func (st *ServerTransfer) ServerExecute(current *api.Transaction, message *
       api.Message) (*api.Message, *api.Message, error) {
23         hash := current.DocHash
24         record, err := st.database.FindRecord(hash)
25         update, err := json.Marshal(record)
26         updateIndex, err := json.Marshal(api.Basic{"done"})
27         sender := api.Message{current.ID, "record", update}
28         broadcast := api.Message{current.ID, "basic", updateIndex}
29         return &sender, &broadcast, nil
30 }
31
32 func (i *IndexTransfer) IndexExecute(transaction *index.Transaction, user
       string, message *api.Message) error {
33         tid := transaction.ID.Hex()
34         err := i.db.UpdateTransaction(tid, transaction.State, "final")
35         return nil
36 }
```

Algorithm 4.39: Get Algorithm

4.5.2 Push

A *Push* transaction transfers a record from the requesting peer to the target peer (Alg. 4.40). The requesting peer connects to the target by using the p2p library and passes the agreed upon document to the target. The target peer checks the document and saves it to their `PeerRecord` collection. The two peers alert the index server of

the transfer. The index server checks the message and transitions the transaction to the *final* state.

```
1  func (cp *ClientPush) ClientExecute(transaction *api.Transaction) (*api.
       Message, error) {
2          if transaction.State != "transfer" {
3                  return nil, errors.New("wrong state")
4          }
5          record, err := cp.database.FindRecord(transaction.Hash)
6          push, err := json.Marshal(record)
7          message := api.Message{transaction.ID, "record", push}
8          m, err := json.Marshal(message)
9          request := p2p.Request{"/", "message", m}
10         ip := net.ParseIP(transaction.ReceiverIP)
11         port := transaction.Port
12         p2pClient := p2p.NewClient(cp.logger, ip, port)
13         err = p2pClient.Connect()
14         defer p2pClient.Close()
15         response, err := p2pClient.Do(&request)
16         data, _ := json.Marshal(api.Basic{"done"})
17         indexUpdate := api.Message{transaction.ID, "basic", data}
18         return &indexUpdate, nil
19 }
20
21 func (sp *ServerPush) ServerExecute(current *api.Transaction, message *api.
       Message) (*api.Message, *api.Message, error) {
22         record, err := getRecord(message)
23         peerRecord := &model.PeerRecord{transaction.SenderID, transaction.ID
               , record.Time, record.Url, record.Data, record.Hash}
24         err = sp.database.StorePeerRecord(peerRecord)
25         updateIndex, err := json.Marshal(api.Basic{"done"})
26         sender := api.Message{current.ID, "basic", updateIndex}
27         broadcast := api.Message{current.ID, "basic", updateIndex}
28         return &sender, &broadcast, nil
29 }
30
31 func (i *IndexPush) IndexExecute(transaction *index.Transaction, user
       string, message *api.Message) error {
32         tid := transaction.ID.Hex()
33         err := i.db.UpdateTransaction(tid, transaction.State, "final")
34         return nil
35 }
```

Algorithm 4.40: Push Algorithm

4.5.3 Service

A *Service* transaction involves one regular peer and a service (Alg. 4.41). The peer connects to the service using the P2P library. After connecting, the client passes the agreed upon record to the service. The service invokes the requested application

and returns the result to the client. The client saves the result to its database as a new record. If the operation is successful, the client and service both alert the index server. The index server checks the messages and moves the transaction to the *final* state.

```
1  func (c *ClientProcess) ClientExecute(transaction *api.Transaction) (*api.
       Message, error) {
2          if transaction.State != "transfer" {
3                 return nil, errors.New("not process state")
4          }
5          record, err := c.db.FindRecord(transaction.DocHash)
6          jsonRecord, err := json.Marshal(record)
7          message := api.Message{transaction.ID, "record", jsonRecord}
8          m, err := json.Marshal(message)
9          request := p2p.Request{"/", "message", m}
10         ip := net.ParseIP(transaction.ReceiverIP)
11         port := transaction.Port
12         p2pClient := p2p.NewClient(c.logger, ip, port)
13         err = p2pClient.Connect()
14         defer p2pClient.Close()
15         response, err := p2pClient.Do(&request)
16         var respM api.Message
17         err = json.Unmarshal(response.Data, &respM)
18         surveyResult := model.SurveyResult{string(respM.Data)}
19         t := time.Now()
20         newRecord, err := model.NewRecord(t, transaction.ID, surveyResult)
21         err = c.db.StoreRecord(&newRecord)
22         data, _ := json.Marshal(api.Basic{"done"})
23         indexUpdate := api.Message{transaction.ID, "basic", data}
24         return &indexUpdate, nil
25 }
26
27 func (s *ServerProcess) ServerExecute(transaction *api.Transaction, message
       *api.Message) (*api.Message, *api.Message, error) {
28         peerRecord, err := getRecordMess(message)
29         j, err := json.Marshal(peerRecord.Data)
30         out, err := s.classifer.Run(j)
31         updateIndex, err := json.Marshal(api.Basic{"done"})
32         sender := api.Message{transaction.ID, "record", []byte(out)}
33         broadcast := api.Message{transaction.ID, "basic", updateIndex}
34         s.logger.Println(sender, broadcast)
35 }
36
37 func (i *IndexProcess) IndexExecute(t *index.Transaction, user string, m *
       api.Message) error {
38         tid := t.ID.Hex()
39         err := i.db.UpdateTransaction(tid, t.State, "final")
40         return nil
41 }
```

Algorithm 4.41: Service Algorithm

4.6 Hypertension Prediction Service

The MEPS Hypertension prediction service has three stages of implementation: training, testing, and deployment. In the training stage, the processed training data is feed into the model constructing the Naive Bayes classifier (Fig. 4.21). The training data feature list consisted of the selected 30 ICD-9-CM prior conditions and the 17 consolidated features. The model is saved as a binary object [133] that can be invoked every time the service receives a request. The model is then deployed as part of the service client. The service application reloads the model, reads the input JSON record, classifies the record for Hypertension, and writes out the result.

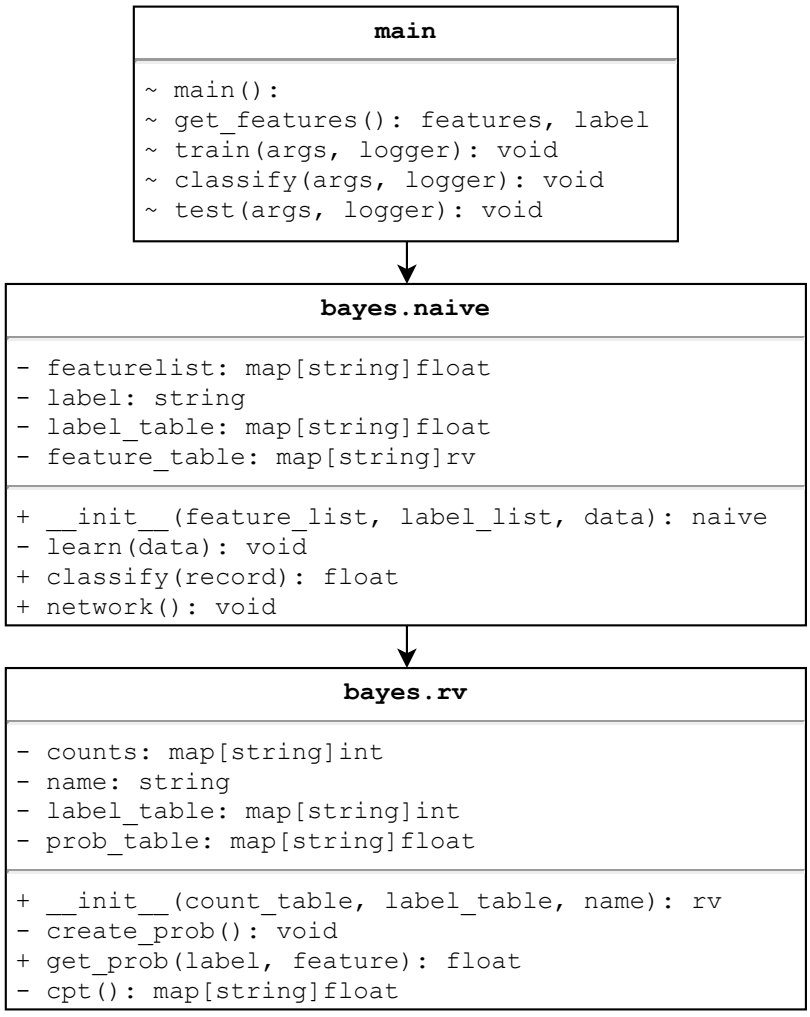


Fig. 4.21.: Hypertension Prediction Classes

```
type SurveyResult struct {
        Value string `json:"result" bson:"result"`
}
```

Algorithm 4.42: Returned Data

4.6.1 Naive Bayes Model

The Naive Bayes is modeled as the *nbayes* class (Fig. 4.21). The class constructor self-calls the *learn* function with the passed data to construct the model. The class wraps a hash table of features each with a Conditional Probability Table (CPT). Each feature is of the RV class. The Naive Bayes creates a database of conditional probabilities by constructing a RV object for each feature and then adding it to the hash table (Alg. 4.43).

```
    def learn(self, train_data):
        total = train_data.shape[0]
        label_true = train_data[self.label].value_counts().get(1)
        label_false = total - label_true
        p_label_true = label_true / total
        p_label_false = label_false / total
        label_table = {"0.0": p_label_false, "1.0": p_label_true}
        self.label_table = label_table
        feature_table = {}
        for feature, value in self.feature_list.items():
            gb = train_data.groupby([self.label, feature])
            count = gb.size().reset_index(name="count")
            count.set_index([self.label, feature], inplace=True)
            r = rv(count, label_table, [self.label, feature])
            feature_table[feature] = r
        self.feature_table = feature_table
```

Algorithm 4.43: Model Construction

The RV class creates the CPT by counting the features frequency intersection with the label and then conditioning it on the probability of that label (Alg. 4.44). The RV class *prob* method then retrieves a requested conditional probability from the CPT.

```
1     def __init__(self, count_table, label_table, name):
2         self.counts = count_table
3         self.name = name
4         self.label_table = label_table
5         self.create_prob()
6
7     def create_prob(self):
8         prob_table = self.counts / self.counts.sum()
9         r = prob_table.loc[0.0, :].apply(lambda x: x / self.label_table["0.0
              "])
10        prob_table.loc[0.0, ].update(r)
11        r = prob_table.loc[1.0, :].apply(lambda x: x / self.label_table["1.0
              "])
12        prob_table.loc[1.0, ].update(r)
13        self.prob_table = prob_table
```

Algorithm 4.44: Calculating Conditional Probability Table

The hypertension event is a Boolean value, either true or false. To classify instances, the Naive Bayes constructs the probability of both events using the provided evidence (Alg. 4.45). The probability of each event is calculated by taking the product of all the features with the evidence. The function returns the label with the higher probability.

```
1     def classify(self, record):
2         sum_cond_prob_x_0 = 1
3         for var, value in self.feature_list.items():
4             val = record[var]
5             r = self.feature_table[var]
6             cond_prob_var = r.get_prob(0.0, val)
7             sum_cond_prob_x_0 = sum_cond_prob_x_0 * cond_prob_var
8         p_cond_0_x = sum_cond_prob_x_0 * self.label_table["0.0"]
9         sum_cond_prob_x_1 = 1
10        for var, value in self.feature_list.items():
11            val = record[var]
12            r = self.feature_table[var]
13            cond_prob_var = r.get_prob(1.0, val)
14            sum_cond_prob_x_1 = sum_cond_prob_x_1 * cond_prob_var
15        p_cond_1_x = sum_cond_prob_x_1 * self.label_table["1.0"]
16        if p_cond_0_x > p_cond_1_x:
17            cl = 0.0
18        elif p_cond_0_x < p_cond_1_x:
19            cl = 1.0
20        else:
21            cl = 0.0
22        return cl
```

Algorithm 4.45: Classifying Instances

4.6.2 Performance

An initial test of the classifier resulted in a 95% accuracy. This performance was caused by cases of prior hypertension, which are easy to classify. After removing prior instances of hypertension from the dataset, the classifier performance is inline with the reviewed literature. The results are shown in Table 4.1. The table indicates that the model struggles when predicting positive results. A potential source of improvement is the addition of network records to the training data over time.

Table 4.1.: Classifier Performance with Panels 17, 18, and 19

Performance Metric	**Percentage (%)**
Accuracy	75.2
Precision	58.9
Sensitivity	66.3
Specificity	79.1

5. CONCLUSION

A Personal Health Record would benefit patients, providers, and third parties. Patients can control and track the access of records, aggregate multiple providers, and organize the records according to their wishes. Providers can use the PHR as a hub to other providers and to interact with patients. Third parties can directly work with patients through the PHR rather than go through a provider or HIE.

A Peer-to-Peer Personal Health Record Network is patient-centric, highly scalable and can be made resilient to faults. Moreover, peers directly exchange records, limiting a record's exposure. Finally, the network allows for new services to be added at run time.

The traceability of a record accesses is important in making the patients aware of how their records are used. A log of transactions provides a trace of each access to record where each access had to be approved by the patient.

Predicting conditions from historical health data can help improve patient care. Patients and providers can proactively prevent conditions and plan for future care. EHR data provides a broad view of the public's health which can be used to build comprehensive predictive models for important disease conditions. These models can be deployed as a service in the PHR in order to help identify potential conditions.

The system defined and implemented in this thesis is a prototype Peer-to-Peer Personal Health Record network. The network is maintained by an index server which is accessed through a REST API. Peer clients connect to the server to post the metadata of records and to request transactions. Peers directly exchange health records, carrying out the requested transactions independently and only using the index server to confirm and log progress. These transactions can be one of three types: *get*, *post*, and *service*. *get* and *post* transfer documents between clients. Service transactions provide services to other peers. To demonstrate this concept, a prototype service implementing a Hypertension predictor trained on MEPS data, was deployed to the network.

An important direction for future work is the implementation of a client for mobile devices. A majority of patients and health providers use smartphones. This application presents some challenges. Though golang can run on both Android and iOS, the support is still experimental as of 2019 [54]. The GUI would need to be reworked as the mobile support is mostly bindings around iOS and Android native languages. The client database storage would also need to be migrated to Couchbase Lite [35] or SQLite [62].

The current document representation only makes the metadata of the document available to the network. This limits a peer's ability to query for documents. The owner must inform the peer of the document contents and is a direction for future improvement.

The fine grain access control modeled in this thesis is difficult to propose to providers, as evidenced by the harsh reaction of providers to patient control of health records [140, 152]. Additionally, patients approving each transaction is cumbersome. Designing levels of access, with fine-grained control over sensitive records and coarse-grained over others, will be considered for future work.

A core issue with a centralized P2P model is the limited scalability compared to a pure P2P model. However, transactions in the pure model would be more difficult to implement, and the overall maintenance of the framework would increase. Exploring alternate architecture models can help improve the scalability of the network with limited increase in overhead.

In the current implementation the transactions are not fully secure. A document's ownership is unverified, a party can repost modified documents, and the transmission of records is unencrypted. These flaws need to be addressed and the system needs to conform to HIPAA regulations. Transactions are modeled using a simple state machine design. This model is limited. Distributed programming paradigms like Tuple Space and Bitcoin's stack language would allow for more expressive and extensible transactions to be injected at run time. However, this is difficult to implement in untrusted scenarios and would require further research.

REFERENCES

REFERENCES

[1] "Blue button connector," September 2017, (Last accessed: 05/29/2019), internet Archive, orginal link missing. [Online]. Available: https://web.archive.org/web/20170919215632/http://bluebuttonconnector.healthit.gov:80/

[2] 111th United States Congress, "American recovery and reinvest-ment act of 2009," 2009, (Last accessed: 04/04/2019). [Online]. Available: https://www.govinfo.gov/content/pkg/PLAW-111publ5/pdf/PLAW-111publ5.pdf

[3] S. Abraham, G. P. Baer, and G. Greg, *Operating System Concepts*, 9th ed. John Wiley & Sons Inc., 2013, ISBN: 978118063330, 9781118129388.

[4] "Medical expenditure panel survey," Agency for Healthcare Research and Quality, 2006, (Last accessed: 08/31/2018). [Online]. Available: https://meps.ahrq.gov/mepsweb/

[5] "Meps hc-180 2015 medical conditions," Agency for Healthcare Research and Quality, 2017, (Last accessed: 12/04/2018). [Online]. Available: https://meps.ahrq.gov/data_stats/download_data/pufs/h181/h181doc.pdf

[6] "panel design," Agency for Healthcare Research and Quality, 2017, (Last accessed: 12/04/2018). [Online]. Available: https://github.com/HHS-AHRQ/MEPS/blob/master/_images/panel_design.png

[7] B. Alpern and F. B. Schneider, "Defining liveness," Cornell University, Ithaca, NY, USA, Tech. Rep., 1984, (Last accessed: 03/09/2019). [Online]. Available: http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail&id=oai%3Ancstrlh%3Acornellcs%3ACORNELLCS%3ATR85-650

[8] Anonymous, "Transmission control protocol," Internet Engineering Task Force, Information Sciences Institute University of Southern California, 1981, (Last accessed: 05/07/2019). [Online]. Available: https://tools.ietf.org/html/rfc793

[9] Anonymous, "Fips pub 180-4: Secure hash standard (shs)," National Institute of Standards and Technology, Tech. Rep. 180-4, 2015, (Last accessed: 05/07/2019). [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[10] Anonymous, "Fips pub 202: Sha-3 standard: Permutation-based hash and extendable-output functions," National Institute of Standards and Technology, Tech. Rep. 202, August 2015, (Last accessed: 05/07/2019). [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[11] Anonymous, "Index - fhir v3.0.0," Health Level Seven International (HL7), 2017, (Last accessed: 10/14/2018), v3.0.0. [Online]. Available: https://www.hl7.org/fhir/

[12] Anonymous, "About adult bmi," Centers for Disease Control and Prevention, 2017, (Last accessed: 12/04/2018). [Online]. Available: https://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html

[13] D. L. Anthony, C. Campos-Castillo, and P. S. Lim, "Who isnt using patient portals and why? evidence and implications from a national sample of us adults," *Health Affairs*, vol. 37, no. 12, pp. 1948–1954, 2018, (Last accessed: 06/01/2019), pMID: 30633673. [Online]. Available: https://doi.org/10.1377/hlthaff.2018.05117

[14] "Apache license," The Apache Software Foundation, 2004, (Last accessed: 05/07/2019). [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0

[15] A. Back, "Hashcash - a denial of service counter measure," 2002, (Last accessed: 03/09/2019). [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf

[16] O. Ben-Assuli, "Electronic health records, adoption, quality of care, legal and privacy issues and their implementation in emergency departments," *Health Policy*, vol. 119, no. 3, pp. 287–297, 2015, (Last accessed: 06/03/2019). [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/25483873

[17] E. I. Benchimol, L. Smeeth, A. Guttmann, K. Harron, D. Moher, I. Petersen, H. T. Srensen, E. von Elm, S. M. Langan, and R. W. Committee, "The reporting of studies conducted using observational routinely-collected health data (record) statement," *PLOS Medicine*, vol. 12, no. 10, pp. 1–22, 10 2015, (Last accessed: 10/04/2018). [Online]. Available: https://doi.org/10.1371/journal.pmed.1001885

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

[19] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin, "Formal verification of smart contracts: Short paper," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, ser. PLAS '16. New York, NY, USA: ACM, 2016, (Last accessed: 03/10/2019), pp. 91–96. [Online]. Available: http://doi.acm.org/10.1145/2993600.2993611

[20] "Blue button rest api," Blue Button, CMS, 2013, (Last accessed: 05/29/2019), orginal link missing. [Online]. Available: http://blue-button.github.io/blue-button-plus-pull/#clinical-api

[21] T. Bocek, "Tomp2p," October 2016, (Last accessed: 05/30/2019). [Online]. Available: https://tomp2p.net/

[22] K. Caine and W. M. Tierney, "Point and counterpoint: Patient control of access to data in their electronic health records," *Journal of General Internal Medicine*, vol. 30, no. 1, pp. 38–41, Jan 2015, (Last accessed: 05/21/2019). [Online]. Available: https://doi.org/10.1007/s11606-014-3061-0

[23] "care.coach," care.coach corporation, 2019, (Last accessed: 05/29/2019). [Online]. Available: https://www.care.coach/

[24] "Carequality interoperability framework," Carequality, 2019, (Last accessed: 04/04/2019), organization website. [Online]. Available: https://carequality.org/resources/

[25] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002, (Last accessed: 03/05/2019). [Online]. Available: http://doi.acm.org/10.1145/571637.571640

[26] "Blue button 2.0," Center for Medicare & Medicaid Services, 2019, (Last accessed: 05/29/2019). [Online]. Available: https://bluebutton.cms.gov/

[27] Centers for Disease Control and Prevention, "International classification of diseases, tenth revision, clinical modification (icd-10-cm)," 2018, (Last accessed: 03/21/2019). [Online]. Available: https://www.cdc.gov/nchs/icd/icd10cm.htm

[28] Centers for Medicare & Medicaid Services, "Medicare program; hospital inpatient prospective payment systems for acute care hospitals and the long-term care hospital prospective payment system and policy changes and fiscal year 2019 rates; quality reporting requirements for specific providers; medicare and medicaid electronic health record (ehr) incentive programs (promoting interoperability programs) requirements for eligible hospitals, critical access hospitals, and eligible professionals; medicare cost reporting requirements; and physician certification and recertification of claims," Federal Register, pp. 41 144–41 784, 10 2018, (Last accessed: 05/21/2019). [Online]. Available: https://www.federalregister.gov/documents/2018/08/17/2018-16766/medicare-program-hospital-inpatient-prospective-payment-systems-for-acute-care-hospitals-and-the

[29] "Cerner," Cerner Corporation, 2019, (Last accessed: 04/04/2019), company website. [Online]. Available: https://www.cerner.com/

[30] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002, (Last accessed: 06/07/2019). [Online]. Available: http://dl.acm.org/citation.cfm?id=1622407.1622416

[31] N. Christin, "Traveling the silk road: A measurement analysis of a large anonymous online marketplace," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13. New York, NY, USA: ACM, 2013, (Last accessed: 06/07/2019), pp. 213–224. [Online]. Available: http://doi.acm.org/10.1145/2488388.2488408

[32] M. J. Clancy, "Overview of research designs," *Emergency Medicine Journal*, vol. 19, no. 6, pp. 546–549, 2002, (Last accessed: 06/03/2019). [Online]. Available: https://emj.bmj.com/content/19/6/546

[33] J. Concato, N. Shah, and R. I. Horwitz, "Randomized, controlled trials, observational studies, and the hierarchy of research designs," *The New England journal of medicine*, vol. 342, no. 25, pp. 1887–1892, Jun 2000, (Last accessed: 06/03/2019), 10861325[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/10861325

[34] P. Coorevits, M. Sundgren, G. O. Klein, A. Bahr, B. Claerhout, C. Daniel, M. Dugas, D. Dupont, A. Schmidt, P. Singleton, G. De Moor, and D. Kalra, "Electronic health records: new opportunities for clinical research," *Journal of Internal Medicine*, vol. 274, no. 6, pp. 547–560, 2013, (Last accessed: 09/07/2018). [Online]. Available: https://misclibrary.wiley.com/doi/abs/10.1111/joim.12119

[35] "couchbase lite," Couchbase, 2018, (Last accessed: 05/07/2019). [Online]. Available: https://docs.couchbase.com/couchbase-lite/2.5/introduction.html

[36] E. D. Hardt, "The oauth 2.0 authorization framework," IETF, October 2012, (Last accessed: 06/07/2019), rFC 6749. [Online]. Available: https://tools.ietf.org/rfc/rfc6749.txt

[37] J. L. C. de Moraes, W. L. de Souza, L. F. Pires, and A. F. do Prado, "A methodology based on openehr archetypes and software agents for developing e-health applications reusing legacy systems," *Computer Methods and Programs in Biomedicine*, vol. 134, pp. 267 – 287, 2016, (Last accessed: 05/21/2019). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016926071630298X

[38] D. Delen, G. Walker, and A. Kadam, "Predicting breast cancer survivability: a comparison of three data mining methods," *Artificial Intelligence in Medicine*, vol. 34, no. 2, pp. 113 – 127, 2005, (Last accessed: 09/18/2018). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0933365704001010

[39] K. Doi, "Computer-aided diagnosis in medical imaging: Historical review, current status and future potential," *Computerized Medical Imaging and Graphics*, vol. 31, no. 4, pp. 198 – 211, 2007, (Last accessed: 06/03/2019), computer-aided Diagnosis (CAD) and Image-guided Decision Support. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0895611107000262

[40] "Dossia consortium," Dossia, 2018, (Last accessed: 12/25/2018), internet Archive link, orginal gone. [Online]. Available: https://web.archive.org/web/20181225065950/http://dossia.org/

[41] M. J. Druzdzel, "Smile: Structural modeling, inference, and learning engine and genie: A development environment for graphical decision-theoretic models," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, ser. AAAI '99/IAAI '99. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1999, (Last accessed: 09/18/2018), pp. 902–903. [Online]. Available: http://dl.acm.org/citation.cfm?id=315149.315504

[42] "ehealth exchange," eHealth Exchange, 2019, (Last accessed: 04/04/2019), organization Website. [Online]. Available: https://ehealthexchange.org/

[43] "Epic," Epic Systems Corporation, 2019, (Last accessed: 04/04/2019), company website. [Online]. Available: https://www.epic.com/

[44] H. Eric, H. Shan, I. Kevin, K. Andy, and L. Katherine, "A framework for cross-organizational patient identity managment," The Sequoia Project and The Care Connective Consortium, 2018, (Last accessed: 05/21/2019), white paper. [Online]. Available: https://sequoiaproject.org/wp-content/uploads/2015/11/The-Sequoia-Project-Framework-for-Patient-Identity-Management.pdf

[45] Federal Register, "Medicare and Medicaid Programs; Electronic Health Record Incentive Program," pp. 44 313–44 588, 07 2010, (Last accessed: 03/25/2019), 75 FR 44313. [Online]. Available: https://www.federalregister.gov/documents/2010/07/28/2010-17207/medicare-and-medicaid-programs-electronic-health-record-incentive-program

[46] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," Internet Engineering Task Force, January 1997, (Last accessed: 05/10/2019), http/1.1. [Online]. Available: https://tools.ietf.org/html/rfc2068

[47] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[48] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985, (Last accessed: 03/09/2019). [Online]. Available: http://doi.acm.org/10.1145/3149.214121

[49] S. Frade, S. M. Freire, E. Sundvall, J. H. Patriarca-Almeida, and R. Cruz-Correia, "Survey of openehr storage implementations," in *Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems*, June 2013, pp. 303–307.

[50] J. H. Frost and M. P. Massagli, "Social uses of personal health information within patientslikeme, an online patient community: what can happen when patients have access to one another's data," *J Med Internet Res*, vol. 10, no. 3, pp. e15–e15, May 2008, (Last accessed: 05/07/2019), 18504244[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/18504244

[51] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, Jan. 1985, (Last accessed: 05/30/2019). [Online]. Available: http://doi.acm.org/10.1145/2363.2433

[52] A. Gettinger and A. Csatari, "Transitioning from a legacy ehr to a commercial, vendor-supplied, ehr: one academic health system's experience," *Applied clinical informatics*, vol. 3, no. 4, p. 367376, 2012, (Last accessed: 05/21/2019). [Online]. Available: http://europepmc.org/articles/PMC3613040

[53] "Go," Google, 2019, (Last accessed: 02/20/2019), programming language. [Online]. Available: https://golang.org/

[54] "Go mobile," Google and Open Source Contributors, 2019, (Last accessed: 05/07/2019). [Online]. Available: https://github.com/golang/go/wiki/Mobile

[55] J. Gray, "Notes on data base operating systems," in *Operating Systems, An Advanced Course*. London, UK, UK: Springer-Verlag, 1978, (Last accessed: 03/09/2019), pp. 393–481. [Online]. Available: http://dl.acm.org/citation.cfm?id=647433.723863

[56] D. A. Haggstrom, J. J. Saleem, A. L. Russ, J. Jones, S. A. Russell, and N. R. Chumbler, "Lessons learned from usability testing of the va's personal health record," *Journal of the American Medical Informatics Association : JAMIA*, vol. 18 Suppl 1, no. Suppl 1, pp. i13–i17, Dec 2011, (Last accessed: 05/21/2019), 21984604[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/21984604

[57] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009, (Last accessed: 04/04/2019). [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[58] Health and Human Services Department, "2015 edition health information technology (health it) certification criteria, 2015 edition base electronic health record (ehr) definition, and onc health it certification program modifications," Federal Register, pp. 62 601–62 759, 10 2016, (Last accessed: 05/21/2019). [Online]. Available: https://www.federalregister.gov/documents/2015/10/16/2015-25597/2015-edition-health-information-technology-health-it-certification-criteria-2015-edition-base

[59] W. R. Hersh, A. M. Totten, K. B. Eden, B. Devine, P. Gorman, S. Z. Kassakian, S. S. Woods, M. Daeges, M. Pappas, and M. S. McDonagh, "Outcomes from health information exchange: Systematic review and future research needs," *JMIR Med Inform*, vol. 3, no. 4, pp. e39–e39, Dec 2015, (Last accessed: 09/18/2018), 26678413[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/26678413

[60] W. R. Hersh, M. G. Weiner, P. J. Embi, J. R. Logan, P. R. O. Payne, E. V. Bernstam, H. P. Lehmann, G. Hripcsak, T. H. Hartzog, J. J. Cimino, and J. H. Saltz, "Caveats for the use of operational electronic health record data in comparative effectiveness research," *Med Care*, vol. 51, no. 8 Suppl 3, pp. S30–S37, Aug 2013, (Last accessed: 09/18/2018), 23774517[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/23774517

[61] "Improving medicare beneficiary access to health information: Blue button on fhir," HHS IDEA Lab, 2018, , (Last accessed: 05/29/2019), internet Archive, orginal link gone. [Online]. Available: https://web.archive.org/web/20181113071726/https://www.hhs.gov/idealab/projects-item/improving-medicare-beneficiary-access-to-health-information-blue-button-on-fhir/

[62] D. R. Hipp, "Sqlite," Hipp, Wyrick & Company, Inc., 2019, (Last accessed: 04/16/2019). [Online]. Available: https://www.sqlite.org/index.html

[63] "C-cda (hl7 cda r2 implementation guide: Consolidated cda templates for clinical notes - us realm)," HL7 International, December 2018, (Last accessed: 06/07/2019). [Online]. Available: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=492

[64] "Argonaut project," HL7 International, 2019, (Last accessed: 05/29/2019), r3 support, R4 upcoming. [Online]. Available: http://argonautwiki.hl7.org/index.php

[65] Humetrix, 2019, (Last accessed: 05/29/2019). [Online]. Available: http://www.ibluebutton.com/index.html

[66] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, (Last accessed: 03/05/2019), pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855851

[67] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[68] "Indiana health information exchange," Indiana Health Information Exchange, Inc, 2019, (Last accessed: 06/03/2019). [Online]. Available: https://www.ihie.org/

[69] "Health informatics - electronic health record communication," International Organization for Standardization, Geneva, CH, Standard, 2 2008.

[70] P. B. Jensen, L. J. Jensen, and S. Brunak, "Mining electronic health records: towards better research applications and clinical care," *Nature Reviews Genetics*, vol. 13, pp. 395 EP –, May 2012, , (Last accessed: 09/18/2018), review Article. [Online]. Available: https://doi.org/10.1038/nrg3208

[71] C. Johnson, Y. Pylypchuk, and V. Patel, "Methods used to enable interoperability among u.s. non-federal acute care hospitals in 2017," Office of the National Coordinator for Health Information Technology, 2018, (Last accessed: 06/03/2019), no.43. [Online]. Available: https://www.healthit.gov/sites/default/files/page/2018-12/Methods-Used-to-Enable-Interoperability-among-U.S.-NonFederal-Acute-Care-Hospitals-in-2017_0.pdf

[72] C. Jonquet, N. H. Shah, and M. A. Musen, "The open biomedical annotator," *Summit Transl Bioinform*, vol. 2009, pp. 56–60, Mar 2009, (Last accessed: 09/18/2018), 21347171[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/21347171

[73] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, no. 2, pp. 183–233, Nov 1999, (Last accessed: 06/07/2019). [Online]. Available: https://doi.org/10.1023/A:1007665907178

[74] D. Kaelber and E. C. Pan, "The value of personal health record (phr) systems," *AMIA ... Annual Symposium proceedings. AMIA Symposium*, vol. 2008, pp. 343–347, 2008, (Last accessed: 06/02/2019), 18999276[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/18999276

[75] J. M. Kendall, "Designing a research project: randomised controlled trials and their principles," *Emergency Medicine Journal*, vol. 20, no. 2, pp. 164–168, 2003, (Last accessed: 06/03/2019). [Online]. Available: https://emj.bmj.com/content/20/2/164

[76] Z. King, "P2HR, A Personalized Condition-Driven Person Health Record," August 2017, MSECE Thesis, Purdue University, Indianapolis.

[77] C. S. Kruse, D. A. Argueta, L. Lopez, and A. Nair, "Patient and provider attitudes toward the use of patient portals for the management of chronic disease: A systematic review," *J Med Internet Res*, vol. 17, no. 2, p. e40, Feb 2015, (Last accessed: 05/21/2019). [Online]. Available: http://www.jmir.org/2015/2/e40/

[78] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Upper Saddle River, New Jersey: Pearson, 2012, isbn-10: 0132856204.

[79] Kurtis, "Steam is no longer supporting bitcoin," Valve Corporation, December 2017, (Last accessed: 05/17/2019), blog post. [Online]. Available: https://steamcommunity.com/games/593110/announcements/detail/1464096684955433613

[80] A. G. Lalkhen and A. McCluskey, "Clinical tests: sensitivity and specificity," *BJA Education*, vol. 8, no. 6, pp. 221–223, 12 2008, (Last accessed: 05/10/2019). [Online]. Available: https://doi.org/10.1093/bjaceaccp/mkn041

[81] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Communications of the ACM 21, 7 (July 1978), 558-565. Reprinted in several collections, including Distributed Computing: Concepts and Implementations, McEntire et al., ed. IEEE Press, 1984.*, pp. 558–565, July 1978 (Last accessed: 03/09/2019), 2000 PODC Influential Paper Award (later renamed the Edsger W. Dijkstra Prize in Distributed Computing). Also awarded an ACM SIGOPS Hall of Fame Award in 2007. [Online]. Available: https://www.microsoft.com/en-us/research/publication/time-clocks-ordering-events-distributed-system/

[82] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998, , (Last accessed: 03/05/2019). [Online]. Available: http://doi.acm.org/10.1145/279227.279229

[83] L. Lamport, "Fast paxos," *Distributed Computing*, vol. 19, no. 2, pp. 79–103, Oct 2006, (Last accessed: 03/05/2019). [Online]. Available: https://doi.org/10.1007/s00446-006-0005-x

[84] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982, (Last accessed: 03/05/2019). [Online]. Available: http://doi.acm.org/10.1145/357172.357176

[85] J. Lee, "Understanding cryptocurrency & blockchain technology," IUPUI, 2018, (Last accessed: 03/10/2019). [Online]. Available: https://github.com/indystar1/BlockchainSeminarSeries

[86] D. A. B. Lindberg, B. L. Humphreys, and A. T. McCray, "The unified medical language system," *Yearb Med Inform*, vol. 02, no. 01, pp. 41–51, 1993, 41.

[87] P. Linz, *An Introduction to Formal Languages and Automata, Fifth Edition*, 5th ed. USA: Jones and Bartlett Publishers, Inc., 2011.

[88] L. S. Liu, P. C. Shih, and G. R. Hayes, "Barriers to the adoption and use of personal health record systems," in *Proceedings of the 2011 iConference*, ser. iConference '11. New York, NY, USA: ACM, 2011, (Last accessed: 06/01/2019), pp. 363–370. [Online]. Available: http://doi.acm.org/10.1145/1940761.1940811

[89] S. Lohr, "Google to end health records service after it fails to attract users," *New York Times*, 6 2011, (Last accessed: 06/02/2019). [Online]. Available: https://www.nytimes.com/2011/06/25/technology/25health.html

[90] H. J. Lowe, T. A. Ferris, P. M. Hernandez, and S. C. Weber, "Stride–an integrated standards-based translational research informatics platform," *AMIA Annu Symp Proc*, vol. 2009, pp. 391–395, Nov 2009, (Last accessed: 09/13/2018), 20351886[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/20351886

[91] P. Lucas, "Bayesian networks in medicine: a model-based approach to medical decision making," December 2001, (Last accessed: 06/03/2019). [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.4103

[92] C. T. Lye, H. P. Forman, R. Gao, J. G. Daniel, A. L. Hsiao, M. K. Mann, D. deBronkart, H. O. Campos, and H. M. Krumholz, "Assessment of US Hospital Compliance With Regulations for Patients' Requests for Medical Records," *JAMA network open*, vol. 1, no. 6, pp. e183 014–e183 014, Oct 2018, (Last accessed: 05/29/2019), 30646219[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/30646219

[93] M. R. Macedonia and M. J. Zyda, "A taxonomy for networked virtual environments," *IEEE MultiMedia*, vol. 4, no. 1, pp. 48–56, Jan 1997.

[94] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," IETF, 2010, (Last accessed: 03/21/2019). [Online]. Available: https://tools.ietf.org/html/rfc5766

[95] J. C. Mandel, D. A. Kreda, K. D. Mandl, I. S. Kohane, and R. B. Ramoni, "SMART on FHIR: a standards-based, interoperable apps platform for electronic health records," *Journal of the American Medical Informatics Association*, vol. 23, no. 5, pp. 899–908, 02 2016, (Last accessed: 05/29/2019). [Online]. Available: https://doi.org/10.1093/jamia/ocv189

[96] C. J. Mann, "Observational research methods. research design ii: cohort, cross sectional, and case-control studies," *Emergency Medicine Journal*, vol. 20, no. 1, pp. 54–60, 2003, (Last accessed: 06/03/2019). [Online]. Available: https://emj.bmj.com/content/20/1/54

[97] W. McKinney, "Python data analysis library," 2018, (Last accessed: 10/04/2018). [Online]. Available: https://pandas.pydata.org/

[98] "meditech," Medical Information Technology, Inc., 2019, (Last accessed: 04/04/2019), company website. [Online]. Available: https://ehr.meditech.com/

[99] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378.

[100] R. C. Merkle, "Secrecy, authentication, and public key systems." Ph.D. dissertation, Stanford, CA, USA, 1979, aAI8001972.

[101] "Health vault," Microsoft, 2019, (Last accessed: 06/02/2019), notice of shutdown. [Online]. Available: https://international.healthvault.com/us/en

[102] R. Miotto, L. Li, B. A. Kidd, and J. T. Dudley, "Deep patient: An unsupervised representation to predict the future of patients from the electronic health records," *Scientific Reports*, vol. 6, pp. 26 094 EP –, May 2016, (Last accessed: 09/13/2018), article. [Online]. Available: https://doi.org/10.1038/srep26094

[103] "Generic fhir server implementation in golang," mitre, 2018, (Last accessed: 02/20/2019). [Online]. Available: https://github.com/intervention-engine/fhir

[104] M. O. Mohsen and H. A. Aziz, "The Blue Button Project: Engaging Patients in Healthcare by a Click of a Button," *Perspectives in health information management*, vol. 12, no. Spring, pp. 1d–1d, Apr 2015, (Last accessed: 05/29/2019), 26755898[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/26755898

[105] "Mongodb database," MongoDB Inc., 2019, (Last accessed: 02/20/2019). [Online]. Available: https://www.mongodb.com

[106] *MongoDB 4.0 Manual*, 4th ed., MongoDB Inc., 2019, (Last accessed: 05/07/2019), under introduction section databases and collections. [Online]. Available: https://docs.mongodb.com/manual/

[107] "Bson," MongoDB, Inc., 2019, (Last accessed: 05/10/2019). [Online]. Available: http://bsonspec.org/

[108] T. B. Murdoch and A. S. Detsky, "The Inevitable Application of Big Data to Health Care," *JAMA*, vol. 309, no. 13, pp. 1351–1352, 04 2013, (Last accessed: 09/13/2018). [Online]. Available: https://dx.doi.org/10.1001/jama.2013.393

[109] S. Nachimson, "Summary of Responses to an Industry RFI Regarding a Role for CMS with Personal Health Records," Center for Medicare & Medicaid Services, 2005, (Last accessed: 05/21/2019). [Online]. Available: https://www.cms.gov/Medicare/E-Health/PerHealthRecords/Downloads/SummaryofPersonalHealthRecord.pdf

[110] S. K. Nachimuthu and P. J. Haug, "Early detection of sepsis in the emergency department using dynamic bayesian networks," *AMIA Annu Symp Proc*, vol. 2012, pp. 653–662, Nov 2012, (Last accessed: 09/18/2018), 23304338[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/23304338

[111] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, (Last accessed: 03/09/2019). [Online]. Available: https://bitcoin.org/bitcoin.pdf

[112] "International classification of diseases, ninth revision, clinical modification (icd-9-cm)," National Center for Health Statistics, 2015, (Last accessed: 09/03/2018). [Online]. Available: https://www.cdc.gov/nchs/icd/icd9cm.html

[113] C. Neumann, N. Prigent, M. Varvello, and K. Suh, "Challenges in peer-to-peer gaming," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 79–82, Jan. 2007, (Last accessed: 06/01/2019). [Online]. Available: http://doi.acm.org/10.1145/1198255.1198269

[114] G. Niemeyer and GlobalSign, "mgo a go library for mongodb," 2019, (Last accessed: 02/20/2019), v2. [Online]. Available: https://github.com/globalsign/mgo

[115] NORC at the University of Chicago, "Evaluation of the personal health record pilot for medicare fee-for service enrollees from south carolina," U.S. Department of Health and Human Services, 4350 East West Highway, Suite 800Bethesda, MD 20814, Tech. Rep., 2010, (Last accessed: 05/21/2019). [Online]. Available: https://aspe.hhs.gov/system/files/pdf/75991/report.pdf

[116] NORC at the University of Chicago, "Literature review and environmental scan," U.S. Department of Health and Human Services, 4350 East West Highway, Suite 800Bethesda, MD 20814, Tech. Rep., 2010, (Last accessed: 05/30/2019). [Online]. Available: https://aspe.hhs.gov/system/files/pdf/177721/litreview.pdf

[117] "2018 report to congress: Annual update on the adoption of a nationwide system for the electronic use and exchange of health information," Office of the National Coordinator for Health Information Technology, 2018, (Last accessed: 06/03/2019). [Online]. Available: https://www.healthit.gov/sites/default/files/page/2018-12/2018-HITECH-report-to-congress.pdf

[118] Office of the National Coordinator for Health Information Technology, "Connecting health and carefor the nationa shared nationwide interoperability roadmap," 2019, (Last accessed: 03/28/2019). [Online]. Available: https://www.healthit.gov/sites/default/files/nationwide-interoperability-roadmap-draft-version-1.0.pdf

[119] "Health it certification program overview," The Office of the National Coordinator for Health Information Technology, March 2019, (Last accessed: 05/21/2019). [Online]. Available: https://www.healthit.gov/sites/default/files/PUBLICHealthITCertificationProgramOverview.pdf

[120] "What information does an electronic health record (ehr) contain?" Office of the National Coordinator for Helath Information Technology, 2019, (Last accessed: 05/07/2019). [Online]. Available: https://www.healthit.gov/faq/what-information-does-electronic-health-record-ehr-contain

[121] A. Omicini and E. Denti, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277 – 294, 2001, (Last accessed: 05/30/2019). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642301000119

[122] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, (Last accessed: 03/05/2019), pp. 305–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643666

[123] "Openehr deplyed solutions," OpenEHR, 2019, (Last accessed: 05/21/2019), list of users of openEHR. [Online]. Available: https://www.openehr.org/openehr_in_use/deployed_solutions/

[124] "openehr," openEHR Foundation, 2019, (Last accessed: 04/04/2019), organization website. [Online]. Available: https://www.openehr.org/

[125] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. 1221 Avenue of the Americas, New York, NY 10020: McGraw-Hill, 2002.

[126] "patientslikeme," PatientsLikeMe, 2019, (Last accessed: 05/07/2019). [Online]. Available: https://www.patientslikeme.com/

[127] S. Peacock, A. Reddy, S. G. Leveille, J. Walker, T. H. Payne, N. V. Oster, and J. G. Elmore, "Patient portals and personal health information online: perception, access, and use by US adults," *Journal of the American Medical Informatics Association*, vol. 24, no. e1, pp. e173–e177, 07 2016, (Last accessed: 06/01/2019). [Online]. Available: https://doi.org/10.1093/jamia/ocw095

[128] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Predicting healthcare trajectories from medical records: A deep learning approach," *Journal of Biomedical Informatics*, vol. 69, pp. 218 – 229, 2017, (Last accessed: 09/18/2018). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532046417300710

[129] N. Popper and T. Hsu, "Bitcoin plummets more than 30 percent in less than a day," *New York Times*, 12 2017, (Last accessed: 06/07/2019). [Online]. Available: https://www.nytimes.com/2017/12/22/business/bitcoin-plunges-more-than-25-percent-in-24-hours.html

[130] J. Postel, "User datagram protocol," Internet Engineering Task Force, 1980, (Last accessed: 05/07/2019). [Online]. Available: https://tools.ietf.org/html/rfc768

[131] J. Postel, "File transfer protocol," Internet Engineering Task Force, June 1980, (Last accessed: 05/10/2019). [Online]. Available: https://tools.ietf.org/html/rfc765

[132] "Python," Python Software Foundation, 2019, (Last accessed: 02/20/2019), programming language. [Online]. Available: https://www.python.org

[133] *pickle - Python object serialization*, 3rd ed., Python Software Foundation, 2019, (Last accessed: 05/07/2019). [Online]. Available: https://docs.python.org/3/library/pickle.html

[134] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Proceedings First International Conference on Peer-to-Peer Computing*, Aug 2001, pp. 99–100.

[135] A. Roehrs, C. A. da Costa, and R. da Rosa Righi, "Omniphr: A distributed architecture model to integrate personal health records," *Journal of Biomedical Informatics*, vol. 71, pp. 70 – 81, 2017, (Last accessed: 03/25/2019). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532046417301089

[136] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, New Jersey: Pearson, 2010.

[137] "Sas university edition," SAS Institute Inc., 2018, (Last accessed: 08/31/2018). [Online]. Available: https://www.sas.com/en_us/software/university-edition.html

[138] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, Dec. 1990, (Last accessed: 03/09/2019). [Online]. Available: http://doi.acm.org/10.1145/98163.98167

[139] K. Schorsch, "Rauner ends struggling health care initiative," *Crain's Chicago Business*, September 2015, (Last accessed: 06/03/2019). [Online]. Available: https://www.chicagobusiness.com/article/20150922/NEWS03/150929963/gov-bruce-rauner-ends-struggling-illinois-health-information-exchange-initiative

[140] P. H. Schwartz, K. Caine, S. A. Alpert, E. M. Meslin, A. E. Carroll, and W. M. Tierney, "Patient preferences in controlling access to their electronic health records: a prospective cohort study in primary care," *Journal of general internal medicine*, vol. 30 Suppl 1, no. Suppl 1, pp. S25–S30, Jan 2015, (Last accessed: 05/21/2019), 25480721[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/25480721

[141] "1.9. naive bayes," scikitlearn.org, 2017, (Last accessed: 09/01/2018). [Online]. Available: http://scikit-learn.org/stable/modules/svm.html

[142] M. Scrimshire, "Bluebutton fhir api," HHS IDEA Lab and CMS, 2019, (Last accessed: 05/29/2019). [Online]. Available: https://github.com/ekivemark/BlueButtonFHIR_API

[143] F. L. Seixas, B. Zadrozny, J. Laks, A. Conci, and D. C. M. Saade, "A bayesian network decision model for supporting the diagnosis of dementia, alzheimers disease and mild cognitive impairment," *Computers in Biology and Medicine*, vol. 51, pp. 140 – 158, 2014, (Last accessed: 09/18/2018). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010482514000961

[144] S. Shah, D. C. Kaelber, A. Vincent, E. C. Pan, D. Johnston, and B. Middleton, "A cost model for personal health records (phrs)," *AMIA ... Annual Symposium proceedings. AMIA Symposium*, vol. 2008, pp. 657–661, 2008, (Last accessed: 06/01/2019), 18998988[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/18998988

[145] C. Showell, "Barriers to the use of personal health records by patients: a structured review," *PeerJ*, vol. 5, pp. e3268–e3268, Apr 2017, (Last accessed: 06/01/2019), 28462058[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/28462058

[146] P. Srisuresh, B. Ford, and D. Kegel, "State of peer-to-peer (p2p) communication across network address translators (nats)," IETF, 2008, (Last accessed: 03/21/2019). [Online]. Available: https://tools.ietf.org/rfc/rfc5128.txt

[147] P. C. Tang, J. S. Ash, D. W. Bates, J. M. Overhage, and D. Z. Sands, "Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption," *Journal of the American Medical Informatics Association : JAMIA*, vol. 13, no. 2, pp. 121–126, 2006, (Last accessed: 05/21/2019), 16357345[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/16357345

[148] The Linux Foundation, "Hyperledger architecture," 2019, (Last accessed: 03/27/2019). [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

[149] The Office of the National Coordinator for Health Information Technology, "2015 edition market readiness for hospitals and clinicians," U.S. Department of Health and Human Services, March 2019, (Last accessed: 05/21/2019), health IT Quick-Stat 55. [Online]. Available: https://dashboard.healthit.gov/quickstats/pages/2015-edition-market-readiness-hospitals-clinicians.php

[150] The Sequoia Project, "Fhir improves health it interoperability," 2018, (Last accessed: 03/21/2019). [Online]. Available: https://sequoiaproject.org/resources/fhir/

[151] The Skype Team, "Skype the journey weve been on," Microsoft, July 2016, (Last accessed: 05/09/2019), blog post. [Online]. Available: https://blogs.skype.com/news/2016/07/20/skype-the-journey-weve-been-on/

[152] W. M. Tierney, S. A. Alpert, A. Byrket, K. Caine, J. C. Leventhal, E. M. Meslin, and P. H. Schwartz, "Provider responses to patients controlling access to their electronic health records: a prospective cohort study in primary care," *Journal of general internal medicine*, vol. 30 Suppl 1, no. Suppl 1, pp. S31–S37, Jan 2015, (Last accessed: 05/21/2019), 25480720[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/25480720

[153] "My healthevet," United States Department of Veterans Affairs, 2019, (Last accessed: 05/29/2019). [Online]. Available: https://www.myhealth.va.gov/mhv-portal-web/web/myhealthevet/home

[154] United States National Library of Medicine, "Snomed ct," 2019, (Last accessed: 03/28/2019). [Online]. Available: https://www.nlm.nih.gov/healthit/snomedct/

[155] V. Urovi, A. C. Olivieri, S. Bromuri, N. Fornara, and M. I. Schumacher, "A peer to peer agent coordination framework for ihe based cross-community health record exchange," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: ACM, 2013, (Last accessed: 05/29/2019), pp. 1355–1362. [Online]. Available: http://doi.acm.org/10.1145/2480362.2480617

[156] "Summary of the hipaa privacy rule," US Department of Health and Human Services, 2013, (Last accessed: 09/13/2018). [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html

[157] "Unified medical language system," U.S. National Library of Medicine, 2019, (Last accessed: 03/21/2019). [Online]. Available: https://www.nlm.nih.gov/research/umls/

[158] N. G. Weiskopf, G. Hripcsak, S. Swaminathan, and C. Weng, "Defining and measuring completeness of electronic health records for secondary use," *J Biomed Inform*, vol. 46, no. 5, pp. 830–836, Oct 2013, , (Last accessed: 09/18/2018), 23820016[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/23820016

[159] N. G. Weiskopf and C. Weng, "Methods and dimensions of electronic health record data quality assessment: enabling reuse for clinical research," *J Am Med Inform Assoc*, vol. 20, no. 1, pp. 144–151, 2013, (Last accessed: 09/13/2018), 22733976[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/22733976

[160] P. Wicks, M. Massagli, J. Frost, C. Brownstein, S. Okun, T. Vaughan, R. Bradley, and J. Heywood, "Sharing health data for better outcomes on patientslikeme," *J Med Internet Res*, vol. 12, no. 2, pp. e19–e19, Jun 2010, (Last accessed: 05/07/2019), 20542858[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/20542858

[161] C. Williams, F. Mostashari, K. Mertz, E. Hogin, and P. Atwal, "From the office of the national coordinator: The strategy for advancing the exchange of health information," *Health Affairs*, vol. 31, no. 3, pp. 527–536, 2012, (Last accessed: 05/10/2019), pMID: 22392663. [Online]. Available: https://doi.org/10.1377/hlthaff.2011.1314

[162] "Medi-span generic product identifier (gpi)," Wolters Kluwer, 2019, (Last accessed: 05/07/2019). [Online]. Available: https://www.wolterskluwercdi.com/drug-data/gpi/

[163] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger byzantium," 2019, (Last accessed: 03/10/2019). [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[164] World Health Organization, *International classification of diseases : [9th] ninth revision, basic tabulation list with alphabetic index.* World Health Organization, 1978, , (Last accessed: 05/10/2019). [Online]. Available: https://apps.who.int/iris/handle/10665/39473

[165] World Health Organization, "International statistical classification of diseases and related health problems 10th revision," 2016, (Last accessed: 03/27/2019). [Online]. Available: https://icd.who.int/browse10/2016/en

[166] A. Wright, E. G. Poon, J. Wald, J. Feblowitz, J. E. Pang, J. L. Schnipper, R. W. Grant, T. K. Gandhi, L. A. Volk, A. Bloom, D. H. Williams, K. Gardner, M. Epstein, L. Nelson, A. Businger, Q. Li, D. W. Bates, and B. Middleton, "Randomized controlled trial of health maintenance reminders provided directly to patients through an electronic phr," *Journal of General Internal Medicine*, vol. 27, no. 1, pp. 85–92, Jan 2012 (Last accessed: 05/29/2019). [Online]. Available: https://doi.org/10.1007/s11606-011-1859-6

[167] Q. Xia, E. B. Sifah, A. Smahi, S. Amofa, and X. Zhang, "Bbds: Blockchain-based data sharing for electronic medical records in cloud environments," *Information*, vol. 8, no. 2, 2017, (Last accessed: 01/24/2019). [Online]. Available: http://www.mdpi.com/2078-2489/8/2/44

[168] Y. Xu, Y. Xu, and S. Saria, "A bayesian nonparametic approach for estimating individualized treatment-response curves," *CoRR*, vol. abs/1608.05182, 2016, (Last accessed: 09/18/2018). [Online]. Available: http://arxiv.org/abs/1608.05182

[169] P. Yadav, M. Steinbach, V. Kumar, and G. Simon, "Mining electronic health records (ehrs): A survey," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 85:1–85:40, Jan. 2018, (Last accessed: 09/18/2018). [Online]. Available: http://doi.acm.org/10.1145/3127881

[170] C. L. Yocom, "Health information technology," U.S. Goverment Accountabilty Office, 441 G St., NW Washington, DC 20548 4350 East West Highway, Suite 800Bethesda, MD 20814, Tech. Rep., March 2017, (Last accessed: 06/01/2019). [Online]. Available: https://aspe.hhs.gov/system/files/pdf/177721/litreview.pdf

APPENDIX

APPENDIX A: Hypertension Survey Info

```
Type Heartsurvey Struct {
       Age string `json:"AGELAST" bson:"AGELAST"`
       Sex string `json:"SEX" bson:"SEX"`
       Race string `json:"RACEV1X" bson:"RACEV1X"`
       Highbloodpressure string `json:"HIBPDX" bson:"HIBPDX"`
       Agina string `json:"ANGIDX" bson:"ANGIDX"`
       Coroneyheartdisease string `json:"CHDDX" bson:"CHDDX"`
       Heartattack string `json:"MIDX" bson:"MIDX"`
       Otherheart string `json:"OHRTDX" bson:"OHRTDX"`
       Stroke string `json:"STRKDX" bson:"STRKDX"`
       Emphysema string `json:"EMPHDX" bson:"EMPHDX"`
       Highcholesterol string `json:"CHOLDX" bson:"CHOLDX"`
       Diabetes string `json:"DIABDX" bson:"DIABDX"`
       Smoke string `json:"ADSMOK42" bson:"ADSMOK42"`
       Restrictfood string `json:"NOFAT53" bson:"NOFAT53"`
       Moreexercise string `json:"EXRCIS53" bson:"EXRCIS53"`
       Bmi string `json:"BMINDX53" bson:"BMINDX53"`
       Poverty string `json:"POVCAT" bson:"POVCAT"`
       C410 string `json:"410" bson:"410"`
       C413 string `json:"413" bson:"413"`
       C414 string `json:"414" bson:"414"`
       C424 string `json:"424" bson:"424"`
       C425 string `json:"425" bson:"425"`
       C426 string `json:"426" bson:"426"`
       C427 string `json:"427" bson:"427"`
       C428 string `json:"428" bson:"428"`
       C429 string `json:"429" bson:"429"`
       C436 string `json:"436" bson:"436"`
       C437 string `json:"437" bson:"437"`
       C440 string `json:"440" bson:"440"`
       C441 string `json:"441" bson:"441"`
       C442 string `json:"442" bson:"442"`
       C443 string `json:"443" bson:"443"`
       C444 string `json:"444" bson:"444"`
       C447 string `json:"447" bson:"447"`
       C453 string `json:"453" bson:"453"`
       C454 string `json:"454" bson:"454"`
       C455 string `json:"455" bson:"455"`
       C458 string `json:"458" bson:"458"`
       C459 string `json:"459" bson:"459"`
       C490 string `json:"490" bson:"490"`
       C491 string `json:"491" bson:"491"`
       C492 string `json:"492" bson:"492"`
       C493 string `json:"493" bson:"493"`
       C496 string `json:"496" bson:"496"`
       C514 string `json:"514" bson:"514"`
       C518 string `json:"518" bson:"518"`
       C519 string `json:"519" bson:"519"`
```

```
49 }
```

Algorithm A.1: Hypertension Survey Info

APPENDIX B: Hypertension Survey Info

Table B.1.: Feature Vector

header	**values**
401	0
410	0
413	0
414	0
424	0
425	1
426	0
427	0
428	0
429	0
436	0
437	0
440	0
441	0
442	0
443	0
444	0
447	0
453	0
454	0
455	0

continued on next page

Table B.1.: *continued*

header	**values**
458	0
459	0
490	0
491	0
492	0
493	0
496	0
514	0
518	0
519	0
ADSMOK42	1
AGELAST	2
ANGIDX	2
BMINDX53	2
CHDDX	2
CHOLDX	1
DIABDX	2
DUPERSID	26974101
EMPHDX	2
EXRCIS53	1
HIBPDX	2
l401	0
MIDX	2
NOFAT53	1
OHRTDX	1

continued on next page

Table B.1.: *continued*

header	**values**
POVCAT	1
RACEV1X	1
SEX	2
STRKDX	2

APPENDIX C: Example Client Usage of System

Fig. C.1.: Welcome Page

Fig. C.2.: Patient Registration

Fig. C.3.: Patient Login

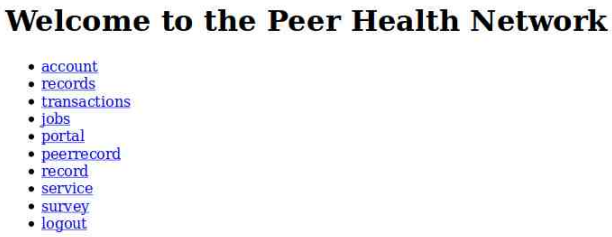


Fig. C.4.: Network Page

Fig. C.5.: Portal Selection

Fig. C.6.: Downloading Records from a Registered Portal

Local Records

Hash	Time	URL
a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
cf319a99daabdf4615dfd8f8c2e7e04ee57a2a92f5707ae91aeb894fc12ca968	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
3deeb9de5e9b5f040f35a825454b06b9277afe156afd337104a0088536254b9f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
bb0c77dca757e9a8d6d5583bd4f1812601f20dc3612a84462d37643b386599644	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
e542218a546fa3947de23e8fbd489d16ca943fc2a072caae3460a38a3b6211d8	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
6cd317c484e611ca05cd7f4d90223caf9ebb0e81d49cadef41d3b267ea6fa374	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
2767b01353f541ef0ea43d41c2dc11e72003966cf2084dcdd21ca436e70064ffe	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
2107f2f2a99bc6593aba7d32b967e793797ce61380a37f151c916cc443b30ff7	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
0e05671518b91926ad65648aa16004f5cd86262e355cd29ac48d3e19ebc8f990	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
5324f88c102c51a20abd12f6676b7c3526c10c1807e88586540972a1dbe39a56	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
2600810ae87a7bcaaeae1fd33f9413caec8b46089cfcfc70363ba77c934ce05f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
a160d4f8d5cd8d3abbddea08bb74194127ff7aff3c453abf334fe8d5a83b2ae1	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
d24b8f61940c3cc7dbbf0430f67c47f1e7cf05bb52cbd00e08cd8ef99f6fe012	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
b15d67f6795b48ac665e4915304c5476ab1ca5d15b5d0c0c67acdf0a38cab377	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
d1919b157963fc21bf5d79f7b11034a68a7320a58e331901ccc55a67d670671c	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
cd2560ed5b6bd8a276c53fcc5004afb3973b871613201d96007509e6e55139e9	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
c7cf369ccaa9c3427e1c7bd4bca74b9fae8b38f50a65588bc430c217b254043d	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
08601f1f8210daba6e598333b248f694a3b141ff2cc9da4c5bb76d9a6ccb4d56	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
9422885c73f355d05b1d3f943de460f284983b8ef6e60305828eb977375be81f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
9751857719367090808a5ab45581923f74007a30dbe9c1e7b262996e050908109	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
d03716b58e577b02e712584e31a8537a91328c17b6b167da396e761c3a87f9ea	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
2b4a76a126d35137a7dea14a2be85108f83696b936409cf2a24d6e82ce96202d	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
0d14ab4001f78cd83ef89deece7f6c03e56b5756797755f87c5af4abf93fbe6f0	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
fb2ed28ecbad34044999c6bd2e0f972c43b1c91908561aa8249496e6dff9c442	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
6961d003d6d1e8a2a950e2a1d44b805c9b6c7cb9ce56794fcea63b872dd7e675	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
6c1aa363ff3f068fe59312a9cdab493da906c0b5939085cb7dab7d9d1e7794c0	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
01603ff26c443d406eb37921789b9feb68387a33e0bdadfceb449f3a25655992	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
e112c8f3713c48844f82697e02132db41b1788fa784a6d76f0199e44c52d5cfb	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
8bf01e3b7725c8030c423c30f6f3f4d43ca4821b215f7241ea805635c40a8b5f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
7658223bff03f4470d80dd3ba69f5a6d6caecaf5699d7b86a4b5bab8ba6e85db	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
69ef6878326f396ae4b9eb9a65899b4d3bd2850e1f925c077d2d9b8789fe70c1	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
ffe4c1ab8a4b5449f33c0349e0768e44b5e1a5be676af1f97760e97e95a7cbf4	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
322b5b95b585b631c80513d03cfab5a5474e8426c36bee4a0104df3941996484	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
60b55d8e3965e93e2782a64b6fb9ae7ad001c78354c3c83a2a842fbb5c833698	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
145cda71bf4f68d359b4754bc0d3f13bb101eb3dcaac685cff2442ab138408e1	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
ef00db62fa0e9ac16df125bce319b58f17c3ef8b151d6dbfa90947842d5a8a8a	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
d7f7b99258675f2f2118d9a44f491df935a0f3bb1704b82ae49a6987ce2f58bd	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
1f76705f0449e3ab05ccaeaaaa62aab5259e929aa75e74f5a6ecf1a75cc95912	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
82d2c4cb96a91bdaf7a560219f7e31e31a078e85a10f2dee9127a7015a5d7c55	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
c9211b53d9a3959e367afe0b1917da74813c3bcf531fcab423bf62d7d0f5c823	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
cb93fa82f8a10f9042bd6c9a84b31e0004bfec80048fb0aa063cf076beaccc57	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
433c30ef6c6dc22946e4cf607edadf578b81e677f4fa6219a54c539c68867053	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
34461c71d34c9e761eded0cdda0a5e8cb2ace073c3085fadd3d965274aa2d5dd	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
5fe6283aaebadf60d0018e5e291472a1ef3e9a0361e64680b53460b8be573730	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
33682e282219d9d62e0741851bcaa1ce9cd63670cc027f3c868db772e7695878	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
e320ca08f0111050cc237771e71f72f216766babeac5f5644112e44a3c3c15d5	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
cafec2653608e2419e8aef7c1414bd8ae64c334831319b8f9d28e50fe12c5d3f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
6bf7497cfe22e39d3f30d2d6db1e667bb498992e262328cf9f2937f6c009a473	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
90fbd9118bfaf0319b6424e1bfff2c696ecf0e92c8b78256488860276 7b6a0c1	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
9c9bff86b8de206f8b2758f8158ab21695380b08fcb3e874fb8142a2853b02eb3	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
b7f74e44931e10e52721c0fef2f9e4b611ad6dc0ba36f09b5575c65d162f2118	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
4f6619e5996578f602e6f95da6e3c3233b41758907fe83b8c522681bea76f0b9	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
e91246f066d10d93dd82491de0a767decc118f7f9da002e709400627d4017bfe	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
eade277adfe71caa591c9d48d3bec6892985c8922e4063a5529c35e4a2b81130	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
8c6dc19743d09de01cef3d46afe770572b2c9df5a8fe98de41402119aee4e63c	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
3f5630efa9478c951bdae622a5da690b9f990fb74b514cf146fe2127c820eedd	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
869cb76c5a29de806e12c70d8a204317dfcb79248a8d13590b1266fbecf2469c	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
fd01dc705960ce7fda41fbc7c24245ddfca5a920a5328b56ce7953255659c63d	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
96c7b6b47bb4df61d74036cd07e7a9fcf145ad10f803992bdc75b0a348310d7e	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
b77d2f3bf79c5b219a32fa25bd8938012b983f4c9f67d51da9e750372f4b037a	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
55ca26cfb8068fcd5da007a9f659b3452ee0ae502bb1dd384f404b9e2d9bb893	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
480a056e5c1055f68ebf1d7e25aac1a9d9ec7e7f45376de5cb2b7e0e9da8f33b	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
2729cbbe71d42918a76e30c09da886199d5a78f5217e2b8abb3d046c73dfb7f8	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
176bbe1975b3095d9fae1f7c116b2628f082cf2c8b4620e1fe4d4aade46e4024	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
b0155422e187d61260db5e67d543b2225be5fcb283a2d2ecc6009cfb16ef7716	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
df0aa328486a9bce607dc1c912df5e9be373a08511a64d6b57252d6f7bee4ec6	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
b34b9d52a0ced02337dac10bf4050d8ead0d794f4e6df079e0049e921df5d335	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
e600fc1a95c24a8ed3aabd2b377fe2a11872a94f2a801c3aa65970f742de8b97	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
60b5cdb5785ef7a294eb9abdeafd3e65a341ae0467fdf9da863287db947e92d1	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
2cd0e5740561b2078a7c49bbca2aab622b9f48ee785b9c01b6af966ec1224ca2	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
68f8183451301e262c8bd34c340e902744e8fff278a611015cd4ce7c1d033bc3	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
da83424d5320806e3849064eddc8029b7cee6d107be7600cb6c0b1b0be5fc8ee	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
b72395940c0119e00412c8da5dee68ac4cbcdb046947057e22a07a6e030a4dbc	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
5033c29be6d4840b8616e82e13481903190f367d762f8ced357770c6e5ff3bd3	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
1f4e734cdafcb7617082f46c54dee94204a7297bcd943471ac7fdea602c21fed	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
3e5991c8dab04cb26763e9c6f9c9dab0fb57b2666bb4218726812244d4c0424f3	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
1db585a6f02c5fceeabb4ec85b4c8d544bbcd4151461a8e86d2bf669308cf702	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
c773fe189af5c5bff28040a99b40389d4f1075e9ab7601aef34d8498372c117	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
44008046dec6bc89e277722c2246466631397b6af2e6014d7643c86b0a9d7fb73	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
07f8b047a1a329ba5a2d9fc0b3080383cafd4468324ca172e3623b318dee1e74	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3
046b0530b2ebfa27cc484586f8da7182fd024254f4dfd583c294c56454d57f36	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3

Fig. C.7.: Viewing Records

Fig. C.8.: FHIR Data for a Single Record and Posting to Network

C.1 Provider Requesting a Patient Health Record

Fig. C.9.: Provider: Network Records

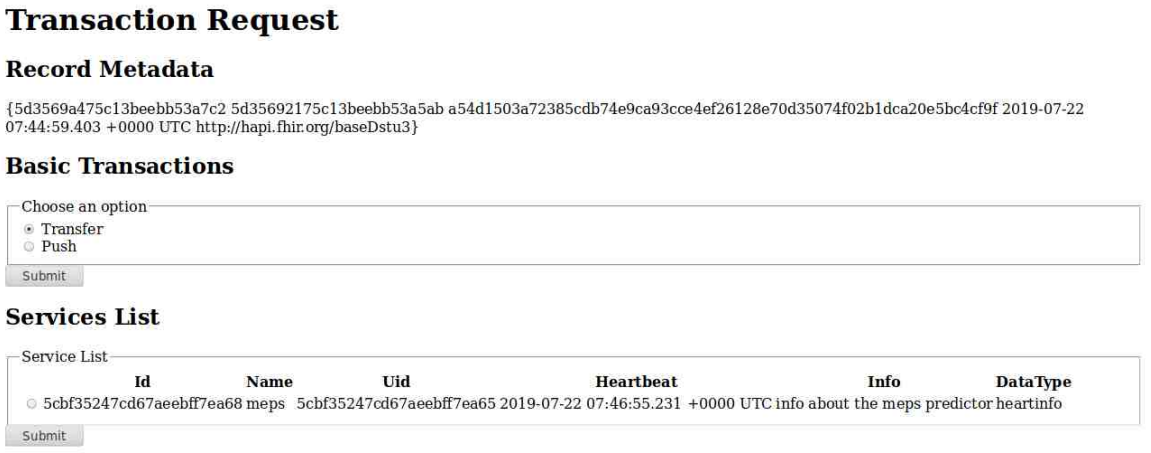


Fig. C.10.: Provider: Selecting Patient Record for a Transaction

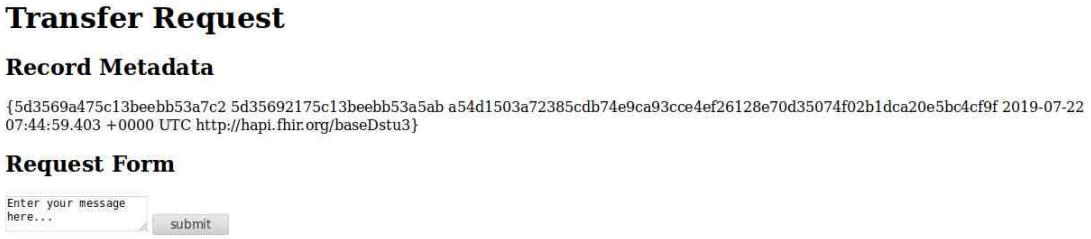


Fig. C.11.: Provider: Sending Request for Get Transaction

Fig. C.12.: Patient/Provider: Transaction Waiting for Approval

List of Network Transactions

Id	DocHash	SenderId	ReceiverId	Action	State	Time	Authorize
5d356a1844e6be19d2a9a5c6	a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f	5d3569d375c13beebb53a84b	5d35692175c13beebb53a5ab	transfer	transfer	2019-07-22 07:47:36.465 +0000 UTC	

Fig. C.13.: Patient: Approving Transaction

List of Network Transactions

Id	DocHash	SenderId	ReceiverId	Action	State	Time	Authorize
5d356a1844e6be19d2a9a5c6	a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f	5d3569d375c13beebb53a84b	5d35692175c13beebb53a5ab	transfer	final	2019-07-22 07:47:36.465 +0000 UTC	

Fig. C.14.: Provider/Patient: Transaction Complete

Records from other Peers

Uid	Tid	Hash	Time	Origin
5d35692175c13beebb53a5ab	5d356a1844e6be19d2a9a5c6	a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3

Fig. C.15.: Provider: Viewing Peer Records

Record: a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f

Metadata

5d35692175c13beebb53a5ab 5d356a1844e6be19d2a9a5c6 a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f 2019-07-22 07:44:59.403 +0000 UTC http://hapi.fhir.org/baseDstu3

Data

{ "_id": "914869", "birthDate": { "precision": "date", "time": "2018-12-16T05:00:00Z" }, "gender": "male", "meta": { "lastUpdated": { "precision": "timestamp", "time": "2018-12-16T19:41:36.807Z" }, "versionId": "1" }, "name": [{ "family": "Walker", "given": ["Austin"], "text": "Austin Walker", "use": "official" }], "resourceType": "Patient", "text": { "div": "\u003cdiv xmlns=\"http://www.w3.org/1999/xhtml\"\u003eAustin Walker\u003c/div\u003e", "status": "generated" } }

Fig. C.16.: Provider: Transferred Peer Record

C.2 Patient Requesting the Hypertension Service

Fig. C.17.: Patient Filling out Survey for Service

Local Record: 8a7dfd023ce520c27b5fcb8bff55dcaf1aad1562db8f161ab31e0a846a6422cc

Metadata

8a7dfd023ce520c27b5fcb8bff55dcaf1aad1562db8f161ab31e0a846a6422cc 2019-07-22 07:50:39.868 +0000 UTC local

Data

{ "410": "0.0", "413": "0.0", "414": "0.0", "424": "1.0", "425": "0.0", "426": "0.0", "427": "0.0", "428": "0.0", "429": "1.0", "436": "0.0", "437": "0.0", "440": "0.0", "441": "0.0", "442": "1.0", "443": "0.0", "444": "1.0", "447": "0.0", "453": "0.0", "454": "0.0", "455": "0.0", "458": "0.0", "459": "1.0", "490": "0.0", "491": "0.0", "492": "0.0", "493": "0.0", "496": "0.0", "514": "1.0", "518": "0.0", "519": "0.0", "ADSMOK42": "2.0", "AGELAST": "3.0", "ANGIDX": "2.0", "BMINDX53": "1.0", "CHDDX": "1.0", "CHOLDX": "", "DIABDX": "2.0", "EMPHDX": "2.0", "EXRCIS53": "1.0", "HIBPDX": "2.0", "MIDX": "2.0", "NOFAT53": "1.0", "OHRTDX": "1.0", "POVCAT": "3.0", "RACEV1X": "6.0", "SEX": "1.0", "STRKDX": "2.0" }

Network Options

Post to Server success

Fig. C.18.: Survey as a Record

List of Network Records

Id	Uid	Hash	Time	Origin	
5d3569a475c13beebb53a7c2	5d35692175c13beebb53a5ab	a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f	2019-07-22 07:44:59.403 +0000 UTC	http://hapi.fhir.org/baseDstu3	Request Transaction
5d356ae675c13beebb53acc2	5d35692175c13beebb53a5ab	8a7dfd023ce520c27b5fcb8bff55dcaf1aad1562db8f161ab31e0a846a6422cc	2019-07-22 07:50:39.868 +0000 UTC	local	Request Transaction

Fig. C.19.: Survey Posted to Network

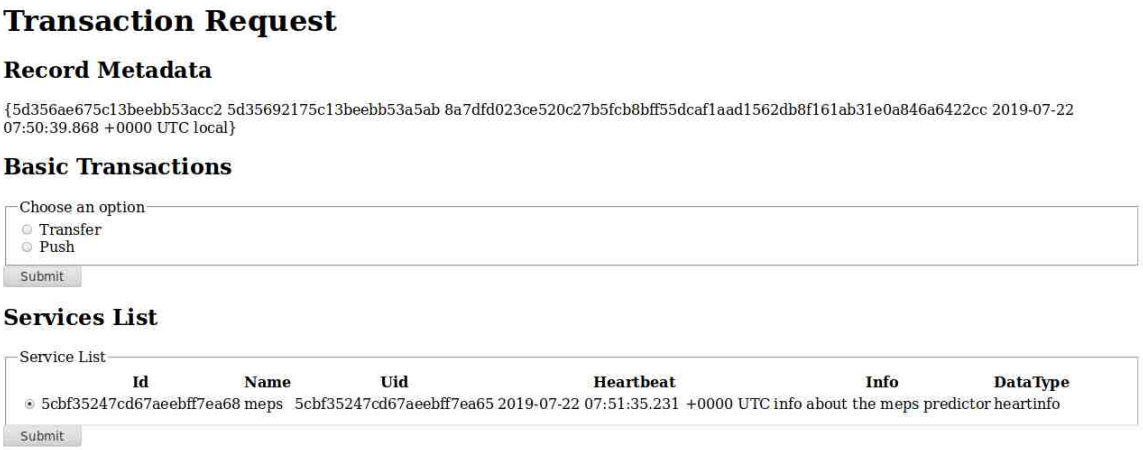


Fig. C.20.: Selecting Service Transaction

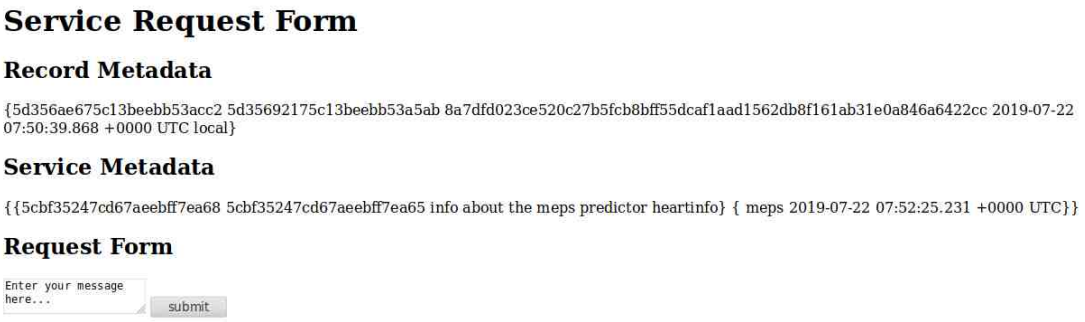


Fig. C.21.: Requesting Service Transaction

List of Network Transactions

Id	DocHash	SenderId	ReceiverId	Action	State	Time	Authorize
5d356a1844e6be19d2a9a5c6	a54d1503a72385cdb74e9ca93cce4ef26128e70d35074f02b1dca20e5bc4cf9f	5d3569d375c13beebb53a84b	5d35692175c13beebb53a5ab	transfer	final	2019-07-22 07:47:36.465 +0000 UTC	
5d356b4444e6be19d2a9a5c7	8a7dfd023ce520c27b5fcb8bff55dcaf1aad1562db8f161ab31e0a846a6422cc	5d35692175c13beebb53a5ab	5cbf35247cd67aeebff7ea65	service	final	2019-07-22 07:52:36.728 +0000 UTC	

Fig. C.22.: Service Transaction Finished

Local Record:
8dc8f3631ae6036ec5b777070e72545c01c83267ecec7a93b8f04736a44a4885

Metadata

8dc8f3631ae6036ec5b777070e72545c01c83267ecec7a93b8f04736a44a4885 2019-07-22 07:52:46.18 +0000 UTC 5d356b4444e6be19d2a9a5c7

Data

{ "result": "true" }

Network Options

Post to Server

Fig. C.23.: Hypertension Service Result