Rutgers University
Software Engineering
Health Analytics Engine

Group 5
Rizwan Chowdhury, Nathaniel Arussy, Dang Khoa Dinh, Smeet Kathiria, Eric Rivera, Hersh Shrivastava, Suva Shahria, Khalid akash

0. Contributions Breakdown **3**

1. Customer Statement of Requirement **7**

2. Glossary Terms **9**

3. System Requirements **11**
3.a Enumerated Functional Requirements 11
3.b Enumerated Non-functional Requirements 12
3.c User Interface Requirements 13

4. Functional Requirements Specification **16**
4.a Stakeholders 16
4.b Actors and Goals 16
4.c Use Cases 17
i. Casual Description 17
ii. Use Case Diagram 17
iii. Traceability Matrix 19
iv. Fully-Dressed Description 20
4.d System Sequence Diagrams 23

5. Effort Estimation using Use Case Points **25**

6. Domain Analysis **31**
6.a Domain Model 31
i. Concept definitions 32
ii. Association definitions 33
iii. Attribute definitions 35
Iv: Traceability Matrix 36
6.b System Operation Contracts 37
6.c Mathematical Model 38

7. Interaction Diagram **41**

8. Class Diagrams and Interface Specification **49**
Class diagram 49

9. System Architecture and Design **53**
Architectural Styles 53
Subsystem Package Diagram 54
Mapping Subsystems to Hardware 54
Persistent Data Storage 55
Network Protocol 56
Global Control Flow 57
Hardware Requirements 58

10. Algorithms and Data Structures **59**
Algorithms 59
Data Structures 60

11. User Interface Design and Implementation **61**
11.a Design 61
11.b User Effort Estimation 64
11.c Changes That We Have Made 64
11.d Ease-of-Use Employed 65

12. Design of Tests **66**
12.a Test Cases 67
12.b Test Coverage 68
12.c Integration Testing 67
12.d Plans for More Tests 68

13. History of Work, Current Status, and Future Work **69**

14. References **74**

0. Contributions Breakdown

Name	Percentage
Nathaniel Arussy	14.29%
Rizwan Chowdhury	14.29%
Dang Khoa Dinh	14.29%
Smeet Kathiria	14.29%
Eric Rivera	14.29%
Suva Shahria	14.29%
Hersh Shrivastava	0%
Khalid Akash	14.29%

Summary of changes

With relation to not predicting disease anymore:

1) Adapted and edited language of Customer Problem Statement.
2) Adapted and removed System Requirements relating to this
3) Adapted and removed Use Cases relating to this

With relation to third party access to public data:
2 and 3 are repeated from before

With relation to login procedure changing from email and password to metamask
2 and 3 are repeated from before
4) Incorporated into 11.a and 11.c and 11.d
5) Glossary Terms Updated

Design of tests:
Test case 3 added
Test cases 1 and 2 made to integrate more into third test

Section 4:
Added Use Case 10

Section 11:
Updated UI Designs
Updated Changes we made to disclude user created units, and include Login

Discussions and Notes:

This section is not apart of the report 2 standard; rather it is meant to discuss questions, concerns and provide additional information to the reader about our choices of the technologies that we use in our project.

Ethereum Costs and Feasibility

The core of our system is using the Ethereum Blockchain to store data. The primary concern with using this existing blockchain infrastructure is the cost of using it. To store data into the blockchain and to run functions that require going through that data require currency called Ether. Ether is valued at $189.20 per Ether. For our system we have to consider the price of Ether and the data we store as well as whether or not actually using the blockchain for our problem is a feasible idea.

For our population descriptor service, we will analyze how much data we utilize and its price in the Ethereum Blockchain. Within the Ethereum blockchain there are costs to uploading/updating data, they are done through transactions in the blockchain which. Transactions, since they change the state of the blockchain, require Ether(money). Reading data from the blockchain is free of charge, unless special functions are employed. For our system, the main costs (for the blockchain) arise from uploading/updating user population descriptors. We will have many users who will regularly update/upload their data. The costs, however, depend on how much ether is actually charged for data put into the blockchain. The price fluctuates but from our testing so far we have seen that uploading 1000 samples of data requires around 3 Ether. Which, if we are to have many customers with many updates/uploads, will become a significant amount of money. Our estimates, with the *current* price of Ethereum is that 1 mb of data will take around $600 to store. This is currently a lower price than that of approximately 1 year ago when Ethereum hit its peak price, making the price of 1mb upwards of a few thousand dollars.

Ultimately, the price of using the Ethereum blockchain will vary based on how much data we store within the blockchain. If we store heavy data, magnitudes in gigabytes or higher, then the costs are far too great. In that case, this particular blockchain is not feasible for our system and should not be used. If, however, at the end it ends up the data we store is in the magnitudes of kilo-bytes to megabytes, then costs can be more manageable for the system.

Why use Ethereum Blockchain?

With the concerns about costs and etc., a question that we considered was why use the Ethereum Blockchain at all. Why not implement our own? We decided to use Ethereum due to the already existing infrastructure it provides and the benefits that come from it. Having an already existing infrastructure provides greater persistence and allows for focus on other features. We will discuss two particular important reasons for our use of the Ethereum blockchain.

The original blockchain paper for bitcoin invented a novel designed called the Blockchain that was a way to persist data and keep consistency among all participating nodes. The specific implementation of this concept was called the Bitcoin which was a way to transfer digital currencies to participants. However, other blockchain projects also came up with their own implementation of the blockchain concept, such as Name coin which persists domain names over the network. Ethereum was novel because it was able to use the blockchain concept to **generalize** the specific implementation of the use of blockchain by allowing users to upload code to customize the behavior of the blockchain. Concepts like Bitcoin and Name coin can easily be implemented on the ethereum network without having to start a new blockchain from scratch. This leads to the next important point.

The integral focus of the block-chain is data persistence and redundancy. For a blockchain persistence is based on how many nodes exist within the network to hold data/ledgers. If there are not many nodes (computers connected to the network) then the there is a greater impact whenever a node stops participating within the network, valuable redundancy is lost. Furthermore, when there are less nodes there is less incentive for new nodes to join or old nodes to continue participating in the network. These events cause the blockchain to lose places to store data/ledgers thus making data persistence an issue. With Ethereum, we do not have this issue since it is an established blockchain network, currently the **2nd most popular blockchain implementation.** The Ethereum blockchain network has many nodes which guarantees data persistence. Working with a blockchain that does not have the participants that ethereum already has makes it so that participants are less incentivized to pursue in mining.

Furthermore, since Ethereum is an established network, there exist many tools and apis that can be used to make our system better. For example, we utilize the web3 library built specifically for the Ethereum blockchain to carry out many functions relating to Logins and carrying out function on data in the blockchain. Also we are planning on utilizing a tool called MetaMask to make logging in simpler and more secure for our users.

Another reason for using the Ethereum blockchain is to save effort on constructing a new blockchain infrastructure and instead focus on features. We get

additional time to work on better graphics and other services that we would not be able to offer otherwise.

1. Customer Statement of Requirement

World health in perspective: chronic, noncommunicable diseases (NCDs), like cardiovascular disease or obesity or diabetes, are steadily increasing around the world. There are 422 million people with diabetes in 2014. This means roughly 1 in every 16 people on the planet has this disease. The WHO estimates that diabetes was the seventh leading causes of death in 2016. Diabetes can be treated and its consequences avoided or delayed with diet, physical activity, and checking blood glucose regularly. The majority of the burden is shouldered by low, and middle income countries. Population growth and aging are the largest contributors. In the US, the number of people over 65 is expected to triple in 2030, while there is a shortage of healthcare professionals. Furthermore, the market for biosensor is expected to surpass 29 billion dollars in 2024. Combating non-communicable disease has been outlined in a number of reports by the WHO and US government which included reducing deaths from NCDs by 25 percent by 2025.

World Bank study estimates NCDs will cost the global economy about $35 trillion from 2005 to 2030. Data-driven diagnostic and analysis are the way of the future. There are 3 reasons for that: increased older population and rising awareness among people, shifting

interest toward home diagnosis, and increased adoption of personalized and technological products.

Our goal shall be to offer a meaningful service to our clients by helping them anticipate health problems and enabling the customers to live a healthier life. When medical data of theirs stands out of the norm too much, we will offer inform the user, and offer some following steps.. Now, users can see their health status on demand. The product will use the large portion of population data and data visualization to put the information into perspective. The product fits well with the trend of personalized medicine and early diagnosis. We will design our product with innovation in mind. So, the system can be adjusted to customer's preferences, medical studies, and advances in biosensor.

The users will need to input medical (e.g. weight, heart rate) to our web-based service and then receive the result. We are working hard to make the service up to date with diverse features to give an in-depth look into the current state of population health. We plan for the product to be an evolution, by bringing data analysis to health-conscious users and health experts alike. Currently, our service focuses on physiological parameter. We will allow the blockchain to persist all user data and attempt to filter out any malicious requests to alter the overall descriptors of the population.

The product will strive to effectively communicate the data and analysis to customers through visual and non-visual means. We will provide visual and non-visual data to help customers compare their own data to others as well as see the bigger picture. The system will contain UI to show graphs that map the aggregate data with respect to each health parameter(or a set of parameters) for groups of people. The graphs will offer to pin-point the location of the users' data so that customers can see where they are relative to other users in a population group. For data that cannot be represented visually there will also be non-visual and text-based UI to show users their data and analysis of that data.The visual data presentation will be a key part in this product. Visual aids, like graphs, will bridge the gap between everyday customers and complex domain technicalities. If the data is mostly numbers then the user may have problems coming to proper conclusions. It would also make the results boring and stale, which would cause loss of attention/interest from the customers. Visual graphs can retain attention and effectively communicate the data and analysis. User will be able to see their place amongst population groups regarding different parameters or conditions and be able to better analyze their own circumstances. They will see connections between different health parameters as well. This would ultimately lead to better decision making on the customer's end.

Our product uses the cutting edge technology of blockchain. The essence of blockchain is decentralized. Since, there is no central server to hack into, a number of users with strong computing power can act as super node to process the data. With a blockchain, anyone with network access can readily access a growing list of user data persisted permanently as long as the blockchain network stays up. However, the cost of performing computation and storing data in any decentralized blockchain comes with its own costs.

Blockchain data can be persisted by having participating nodes mining (submitting proof of work) and offering to perform computations. Replicating all transactions that have been made and constantly competing with other nodes to submit a block is expensive. Estimates show that storing 1 gb of data in a decentralized blockchain like Ethereum costs $186,700 while a typical database costs $00.02 for an equivalent amount. Storage is only part of the costs as the blockchain we would like to utilize is Ethereum; a blockchain that can also perform arbitrary computation on nodes which brings additional costs. Along with this, blockchains are not optimized for mass data retrieval such as other SQL or NoSQL databases.

Ultimately, we don't believe blockchain to be a one stop solution for all purposes; rather, it is a powerful, customizable (in the context of Ethereum), reliable and persistent solution for public data. Using blockchain technology and a group of micro-services (centralized servers and databases) to augment the blockchain, we would like to create a tool to allow customers to truly gauge their health in the context of their peer population. We will ensure that public data is always persisted, not only for our use, but for the use of any other services that would like to use it.

2. Glossary Terms

Smart Contracts- A smart contract is an immutable computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts allow the performance of credible transactions without third parties. Smart contracts can be made by uploading Ethereum Virtual Machine code to the Blockchain and can be invoked by any outside party via transactions.

Storage- Place where all the contract state variables reside. Every contract has its own storage and it is persistent between function calls.

Memory - This is used to hold temporary values. It is erased between (external) function calls.

Ethereum Virtual Machine - The Ethereum Virtual Machine (EVM) is a Turing complete virtual machine that allows anyone to execute arbitrary EVM Byte Code. Every Ethereum node runs on the EVM to maintain consensus across the blockchain.

Truffle Framework - Truffle is a development environment, a testing framework and a crypto asset pipeline in one for development of smart contracts in Solidity programming language.

Gas(Ethereum) is a unit that measures the amount of computational effort that it will take to execute certain operations.

Blockchain - A blockchain is a growing list of records, called blocks, that are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data.

Block - A block records some or all of the most recent transactions that have not yet entered at any prior blocks. In other words, it is a group of transaction at a particular time.

Block time- The block time is the time needed to generate the next block in the chain. It is essentially the time the blockchain miners need to find a solution to the block hash.

Hash - A hash is a function that converts an input of letters and numbers into an encrypted output of a fixed length.

Keccak256 computes the Ethereum-SHA-3 (Keccak-256) hash of the arguments passed into the function. This is how String comparison is done in Solidity. Input values are hashed to a uniform 32 bytes, and the resulting hash code of the strings are compared. It can also be used for pseudo-random number generation.
Solidity is known as a contract-based, high-level programming language. This platform has similar syntax to the scripting language of JavaScript. Solidity as a programming language is made to enhance the Ethereum Virtual Machine. Solidity is statically typed scripting language which does the process of verifying and enforcing the constraints at compile-time as opposed to run-time.

Nonce - A nonce is an abbreviation for "number only used once," which is a number added to an encrypted block in a blockchain that, when rehashed, meets the difficulty level restrictions. The nonce is the number that blockchain miners are solving for. When the solution is found, the blockchain miners are offered cryptocurrency in exchange.

Ethereum network- A collection or a group consisting of several different nodes running an Ethereum client. Otherwise referred to as a super-node.

Node - A node is a participating computing system on a blockchain network. A powerful machine that is able to run blockchain software. The role of a node is to support the network by maintaining a copy of a blockchain and, in some cases, to process transactions.

MetaMask- Allows user to run Ethereum in browser without running a full Ethereum node. This includes a secure identity vault, which offers a user interface to manage identities on different sites and sign blockchain transactions.

3. System Requirements

3.a Enumerated Functional Requirements

Enumeration	Priority	Description
REQ-1	10	The system shall keep immutable data submitted by participants in a population and persist it forever.
REQ-2	9	The system shall keep user-data anonymous to anyone other than the user.
REQ-3	5	The system shall allow the user to use their own data as payment for descriptors of the entire population in general for the given data-type.
REQ-4	2	The system shall parse the data time-series data on behalf of the user and present it.
REQ-5	5	The system shall allow the user to upload user-data from any device with a web browser available.
REQ-6	2	The system shall allow the user to request for updated data. The system should periodically update data presented to the user by-itself without manual intervention.
REQ-7	1	The system shall allow participants to retrieve only their data via a popular transport format (JPEG).
REQ-8	4	The system shall parse the data and present it in a useful visual format for the user to see where they stand in the population and detect trends (different forms of graphs).
REQ-9	8	The system shall provide ways to store identity information and persist it permanently.
REQ-10	6	The system shall provide ways to authenticate a given user for its population descriptors.
REQ-11	5	The system shall process the blockchain data and give useful statistical information to the user in addition to data visualization.
REQ-12	3	The system should try to rate the user’s standing within the population positively or negatively and give users web links to more information on how to improve such factors.

3.b Enumerated Non-functional Requirements

Enumeration	Priority	Description
REQ-13	3	The system will log out users on the web client after prolonged inactivity
REQ-14	2	The amount of transactions by system required in a certain period of time (hours)
REQ-15	6	The amount of data the system needs to store must be malleable. The amount of storage needs to scale with new users.
REQ-16	7	A user can't log into another user's account through malicious activity.
REQ-17	4	User experience is enhanced by quick response from the system, gathering and uploading data to the database is fast, ui runs smoothly.
REQ-18	7	Should have an easy and intuitive UI for existing and new users.
REQ-19	3	User should be notified of failed login attempts.
REQ-20	5	Users can access the web client through all popular web browsers on computers and smartphones.

3.c User Interface Requirements

UI requirements	Priority	Description
REQ-21	10	**Graphical Visualization of Aggregate Population Data**. Users require the ability to view their standings within populations. Line graphs, pie graphs, and maps are a good means by which to provide that insight. Users will be able to view graphs of aggregate data, and view their position within graphs by hovering over graphs and will have their position highlighted with the information at that point. Elements include: -Line Graphs -Display Bar: displays values being hovered over in graph. -Pie Graphs -Map: Where some of these results are coming from
REQ-22	8	**Non-graphical data display**. Some data or data summaries, such as means or medians, are better displayed as text. Users will need ui to view data they wish to see in an organized manner. Data will be shown user personal progress, comparisons to average, and suggestions on how to proceed.
REQ-23	7	The system will contain **DATA ENTRY FORMS** to obtain personal population parameters from users. Will contain various input methods to obtain required data based on type of question. User's weight will require text entry while whether or not a user has a certain condition will require a selection.. Input methods include: -Text Entry Fields(to enter user data like weight or height) -Radio Buttons (for single choice question) -Multi-Select Buttons

REQ-24	7	**User Login Page.** Users will require input means to enter their login information in order to access services and create an account with the system. Elements include: -Sign up button -Login Button -Ethereum Account authorization
REQ-25	7	**User Log-Out Button** -Showing log-out button on certain screens once the user is logged into the system. -On user interaction with logged out button, system should go back to the login page.

Figure 2.1 - Data Entry Forms

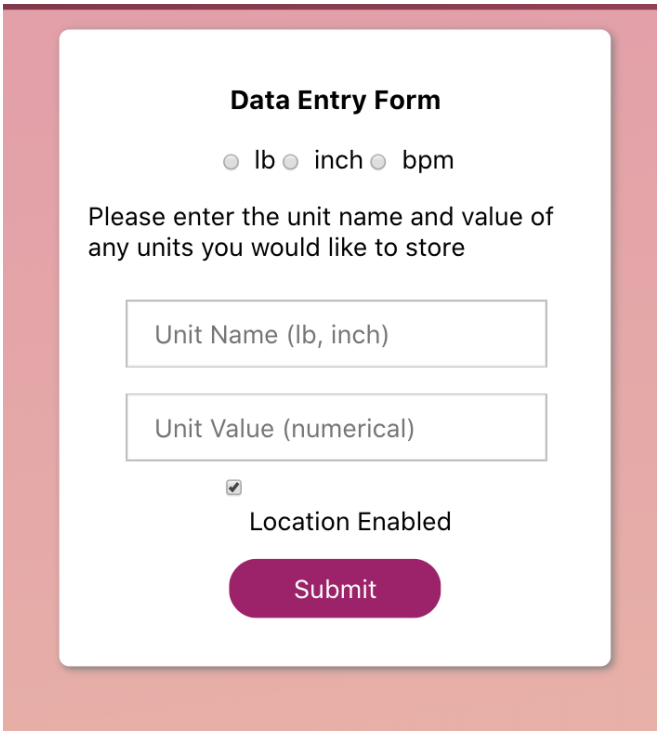


Figure 2.3 - Visual Data Presentation side by side with non-graphical data

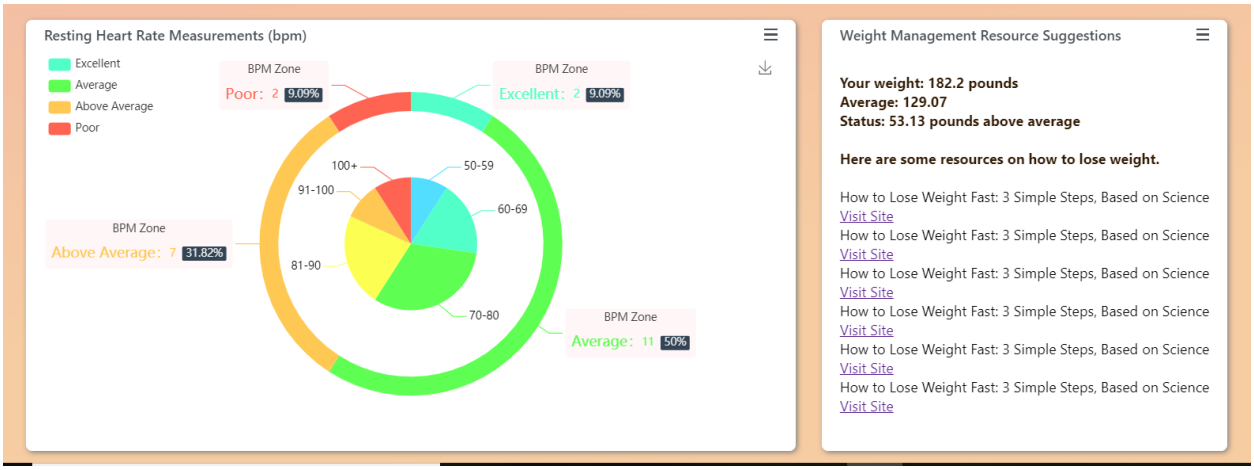
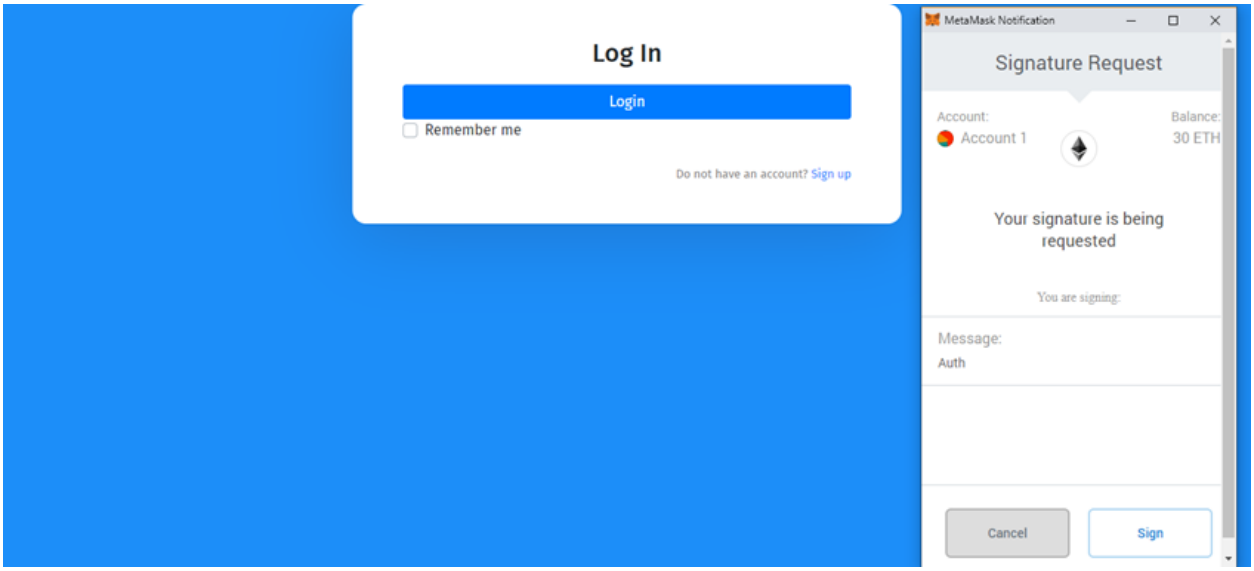


Figure 2.4 - User-Login Page



4. Functional Requirements Specification

4.a Stakeholders

I.Health App User - Has an interest in the system, because the user wants to monitor their own health status.
II.Blockchain - Is a ledger that records transactions, and, in this project, health data. The cryptography of the blockchain has to keep this data secure.
III.The medical service provider – Access to mass data analytics on secure data
IV.Health Patients – Benefit from their caretaker giving contextualized diagnoses and information relative to others

4.b Actors and Goals

Actors	**Goals**
User(Initiating)	To be able to access the webpage and be able to login into the system.
User(Initiating)	To be able to access information of their own profile and health data.
User(Initiating)	To be able to see and compare their current health standings relative to other users on a graph that maps users health data.
User(Initiating)	To be able to input their health information.
Non-User(Initiating)	Able to create an account with the system.
Site Administrator(Initiating)	Perform administrative work for the website, manage database,smart contracts,server.
SmartContracts (Participating)	computes statistics and health data based on user input.

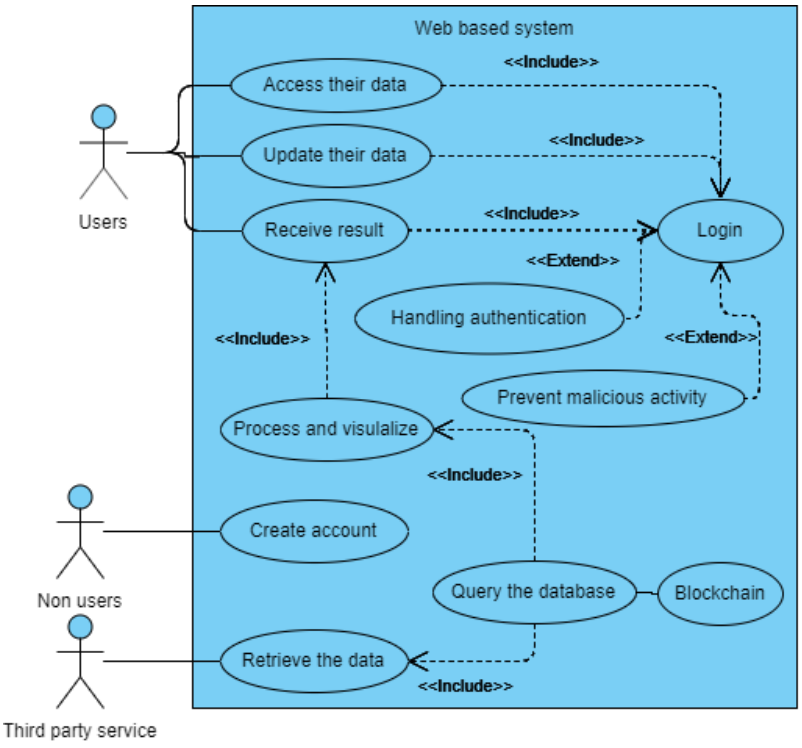
4.c Use Cases

i. Casual Description

Use Case	Name	Description	Requirements
UC-1	login	The ability for one to access their account after authorization.	REQ-24,REQ-16
UC-2	InputData	Allows the user to input their health related information into the system	REQ3, REQ-5,REQ-23, REQ-9,
UC-3	ReceiveDataForUser	The user will be able to see all their data at any given time.	REQ6, REQ7,REQ22, REQ21
UC-4	CompareData	The system will compare data from the user to the population through various visual means. E.g. graphs, tables, etcetera	REQ-22, REQ-21, REQ-12, REQ-11, REQ-8, REQ-4
UC-5	DisplayVisualAnalytics	The system shall display all data in a graphical/visual format.	REQ4, REQ8, REQ-22, REQ-21, REQ-17
UC-6	LogoutUser	The system logs out the user after prolonged inactivity or if the user requests to be logged off,	REQ25, REQ-13
UC- 7	Register(Account Creation)	Allows non-user to create an account with the system to use the services.	REQ-23

UC-8	Data Administration	Allows site administrators to manage user data stored on blockchain like data request,persisting,output and write efficient smart contracts.	REQ-14,REQ-1 REQ-4
UC-9	User Notification	Notifies user on incorrect data input	REQ-23
UC-10	Resource Suggestion	Offer user resources based on compared data	REQ-12, REQ-22

ii. Use Case Diagram



Figure

iii. Traceability Matrix

Reqs	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10
REQ1	10	X		X							
REQ2	9	X									
REQ3	5			X							
REQ4	2					X		X			
REQ5	5	X		X							
REQ6	2				X						
REQ7	1				X						
REQ8	4					X		X			
REQ9	8	X		X							
REQ10	6										
REQ11	2		X								
REQ12	1		X								X
REQ13	7						X				
REQ14	5					X					
REQ15	3					X					
REQ16	3								X		
REQ17	2										
REQ18	6			X						X	
REQ19	7	X									
REQ20	4	X	X	X	X	X	X	X	X	X	
REQ21	2	X		X	X	X				X	
REQ22	3	X									X
REQ23	5						X				
REQ24	10					X		X			
REQ25	8				X	X		X			
MaxPW		10	4	10	8	10	7	10	4	6	3
TotalPW		48	7	47	17	38	16	28	7	12	4

iv. Fully-Dressed Description

Use Case: UC-2	InputData
Requirements	REQ3, REQ5,REQ-22, REQ-9,
Initiating Actor:	User
Actor's Goals:	To access the system and put in health information, log in information.
Preconditions:	The user has access to the internet to interact with the system. The system asks for and has ways to fill out all the necessary information
Postconditions:	The information will be stored in the system to be used by the blockchain. The user can view their information.

Flow of Events:
1.<- System asks for identification in the form of a login
2.-> User supplies the right identification information.
3.<- The system prompts for information to be filled out.
4.->User fills in the necessary information they want to fill out.

Extensions

1a User chooses to create an account because they're a new user.
-> User click on the create new account button
<- System takes the user to the page to create new account

1b Login failed
<- System notifies user of failed attempt
<- System allows user to try to login again.

Use Case: UC-4	CompareData
Requirements	REQ-20, REQ-21, REQ-12, REQ-11, REQ-8, REQ-4
Initiating Actor:	User
Actor's Goals:	To parse through the block chain data and give back useful information comparing the user's data to the population data.

Preconditions:	The user is logged in to the system.
Postconditions:	The system will display the users data vs the populations data to the user in various visual medium; only the *type*

Flow of Events:

1.<- The user asks system to show population data
2.<- The system parses and analyzes the population's data.
3.<- The system creates useful visual mediums to display
4.<- The system makes comparisons and analyzes the user's data vs the population's data.
5.<-The system display the user's standing compared to the population
6.<-The system provides web links based off their standings to further improve the user's health

Use Case: UC-3	ReceiveDataForUser
Requirements	REQ6, REQ7,REQ20, REQ21
Initiating Actor	User
Actor's Goals	User will initiate an action to receive his or her personal data from the blockchain and be presented with visual analytics of his/her data.
Preconditions:	The user is logged in to the system
Postconditions:	The user will receive processed data from the blockchain and will have graphical representations of his or her personal data.

Flow of Events:

1.<- The user asks to show user data
2.<- The frontend asks a proxy service to gather data from the blockchain using a user identifier
3.<- Blockchain smart contract is invoked to retrieve paginated data
4.-> Data is returned to proxy service to process and format for different visualization data charts.
5.-> Data is returned to the frontend to be presented to the user

Use Case: UC-1	Login
Requirements	REQ 23, REQ 15
Initiating Actor	User
Actor's Goals	User will access his or her account.
Preconditions:	The user has been authorized to access the account entered.
Postconditions:	The user will now be able to modify and view his or her data.

Flow of Events:

1.-> System prompts user for the username and password.
2.<- User enters username and password.
3.-> System looks through database to find the account.
4.-> If account exists, system authorizes the user.

Alternate Flow of Events

1.-> System prompts user for the username and password.
2.<- User enters username and password.
3.-> System looks through database to find the account.
4.-> If account does not exist, system prompts for the correct information.
5.<- User enters information again.

4.d System Sequence Diagrams

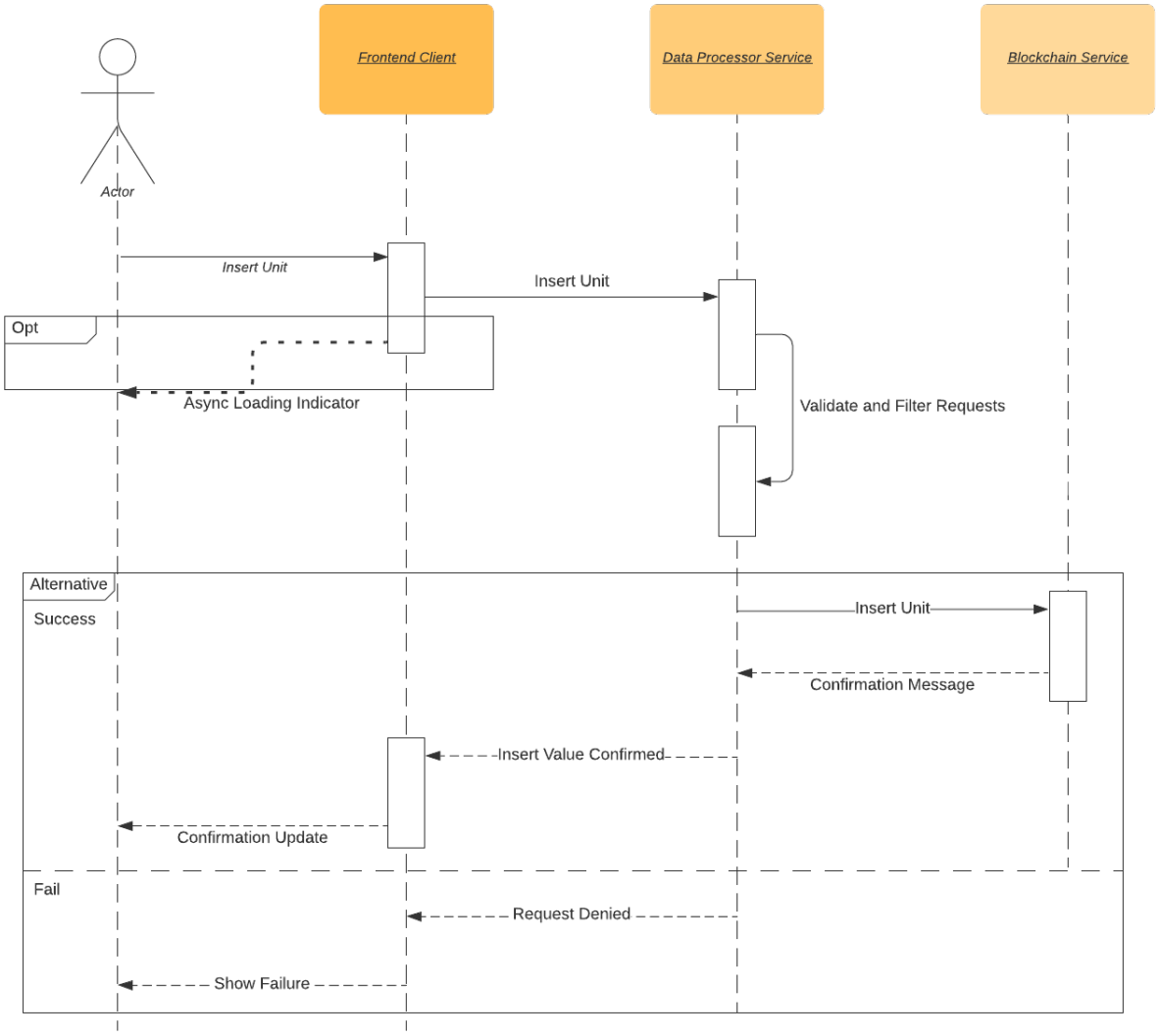


Figure 3.c.1 UC-2

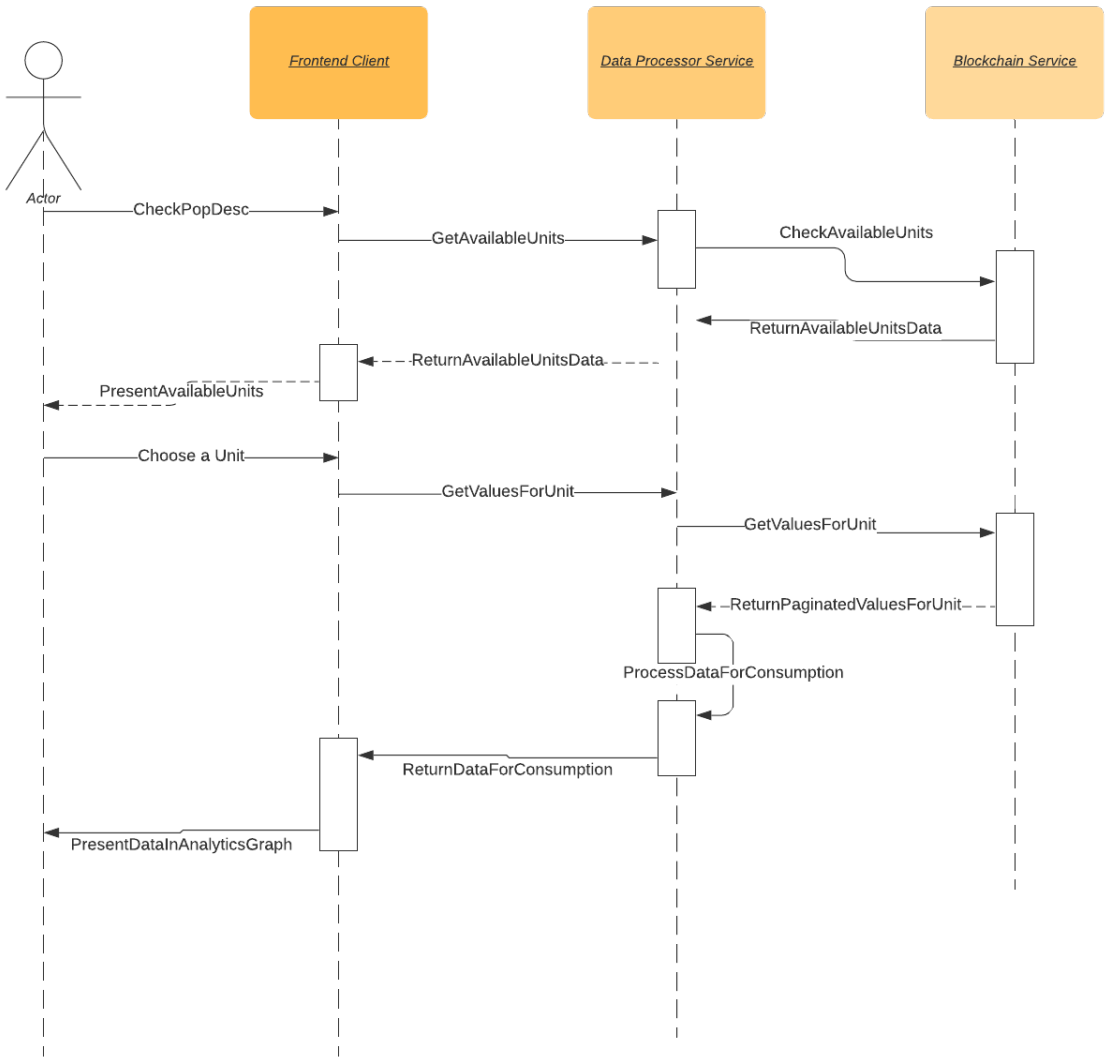


Figure 3.c.2 - UC-5

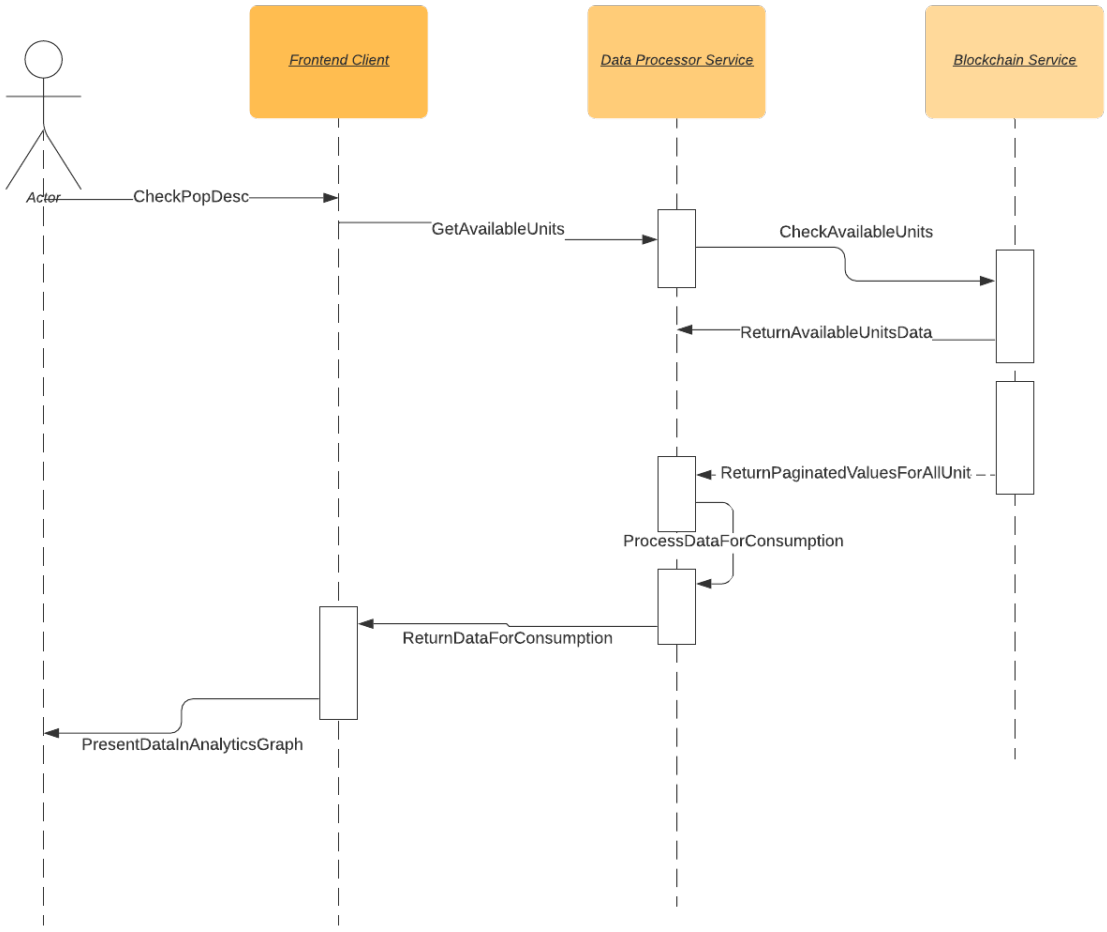


Figure 3.c.3 UC-3

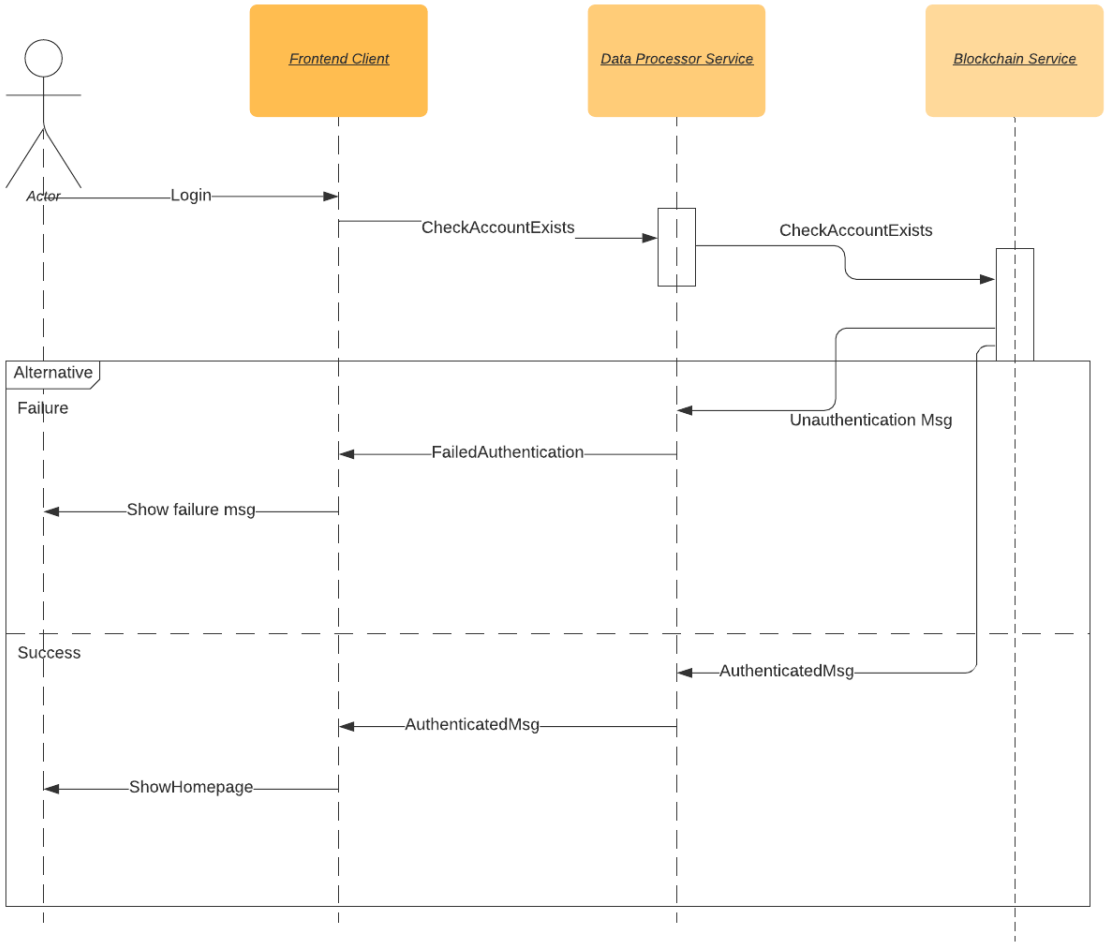


Figure 3.c.4 UC-1

5. Effort Estimation using Use Case Points

Actor Classification			
Actor Name	Description	Complexity	Weight
User/Frontend(FE)	The user is interacting with a graphical user interface during account log in, account creation, data entry, and viewing their results (visual and text-based)	Hyper-Complex	4
Blockchain	Database system interacting with the server via smart contracts	Average	2

Server	Data processing system interacting with the blockchain via smart contracts and with the front-end/user via GraphQL API	Average	2
Smart Contracts	A smart contract is an immutable computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract.	Average	2
UAW(Health Engine) = 0x Simple + 3x Average + 1x Hyper-Complex = 3x2 + 1x4 = 10			

Use Case Classification			
User Case	Description	Category	Weight
Login (UC-1)	Moderate user interface. 6 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Average	10
InputData (UC-2)	Moderate user interface. 7 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Complex	15
ReceiveDataForUser(UC-3)	Complex user interface. 8 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Complex	15
CompareData (UC-4)	Complex user interface. 13 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Complex	15
DisplayVisual Analytics (UC-5)	Complex user interface. 13 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Complex	15

LogoutUser (UC-6)	Moderate user interface. 6 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Average	10
Register (UC- 7)	Moderate user interface. 8 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Complex	15
Data Administration (UC-8)	Moderate user interface. 8 steps for success scenario. 3 participating actors (Server, Smart Contracts, Blockchain)	Complex	15
User Notification (UC-9)	Simple user interface. 3 steps for success scenario. 1 participating actor (FE)	Simple	5
Resource Suggestion (UC-10)	Complex user interface. 13 steps for success scenario. 4 participating actors (FE, Server, Smart Contracts, Blockchain)	Complex	15
UUCW(Health Engine) = 1x Simple + 2x Average + 7x Complex = 1x5 + 2x10 +7x15 = 130			

Technical Complexity Factors				
Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	*Distributed System*: Distributed web-based system by blockchain nature	2	3	2x3=6
T2	*Response time/performance objectives*: Minimizing latency for data fetches to the blockchain is important.	1	3	1x3=1
T3	*End-user efficiency:* User expects good performance.	1	3	1x3=1

T4	*Internal processing complexity:* Complex processing through smart contracts and GraphQL API	1	5	1x5=5
T5	*Code reusability:* Necessary for GraphQL calls that feed data visualisations and other user data interactions	1	3	1x3=3
T6	*Easy to install:* Web based system. Installation not applicable.	0.5	0	0
T7	*Easy to use:* User ease of use is very important. Minimal learning curve for system.	0.5	3	0.5x3=1.5
T8	*Portability to other platforms:* Portable across many modern browsers.	2	3	2x3=6
T9	*System maintenance:* Compartmentalized system design is crucial to feature addition/enhancements/ modifications. Moderate effort for change required.	1	2	1x2=2
T10	*Concurrent/parallel processing:* Multiple users access and use the service at any given time. This is a requirement.	1	4	1x4=4
T11	*Security features:* Blockchain technology is very important to protect sensitive health data.	1	5	1x5=5
T12	*Access for third parties:* Not applicable.	1	0	0
T13	End user training: No training required.	1	0	0
Technical Factor Total:	34.5			
TCF(Health Engine) = Constant-1 + Constant-2 x Technical Factor Total = 0.6 + (0.01 x 34.5) = 0.945				

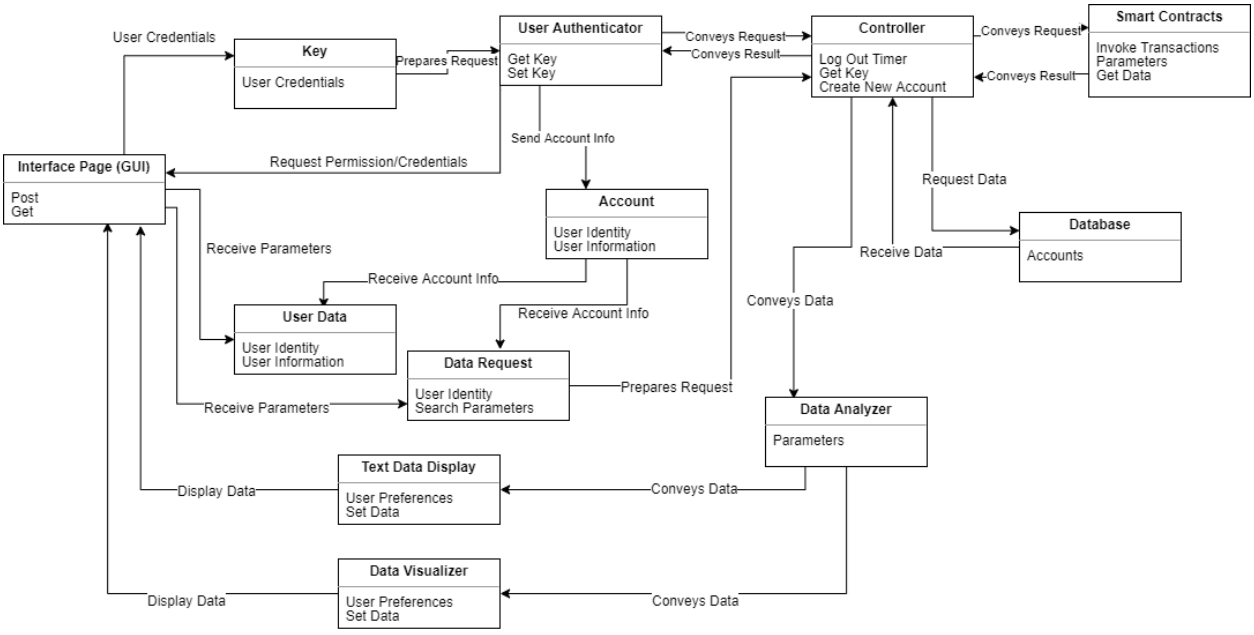
Environmental Complexity Factors				
Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor
E1	*Familiarity with development process:* UML-based approach	1.5	3	1.5x3=4.5
E2	*Application problem experience:* some experience with application problem	0.5	2	0.5x2=1
E3	*Experience of team:* Distributed experience among the team	1	3	1x3=3
E4	*Lead analyst capability:* Team leader has good capability	0.5	4	0.5x4=2
E5	*Motivation*: Motivation distributed among the team	1	3	1x3=3
E6	*Stability of requirements:* moderately stable requirements expected	2	4	2x4=8
E7	*Part-time staff:* All team members are part-time members of the course/project development	-1	5	-1x5=-5
E8	*Difficult programming language:* Programming language of average difficulty (Javascript,Typescript)	-1	3	-1x3=-3
Environmental Factor Total:	13.5			

ECF(Health Engine) = Constant-1 + Constant-2 x Environmental Factor Total = 1.4 + (-0.03 x 13.5) = 0.995				

Use Case Points and Project Duration
Use Case Points Calculation: UCP = UUCP x TCF x ECF = (10 + 130) x (0.945) x (0.995) = 131 Use Case Points *Project Duration:* Using a productivity factor of 28 hours per use case point, Duration = 28 * 131 = 3668

6. Domain Analysis

6.a Domain Model



Domain Model Diagram

i. Concept definitions

Responsibility Description	Type	Concept Name
Website with React pages for account log in, account creation, and viewing outcomes and results in UI made from Scalable vector graphics.	K	User Interface (GUI)
Form specifying the parameters for data retrieval from the blockchain, as well as parameters for desired data analysis.	K	Data Request
Data input form where the user enters their health information	K	User Data
Render non-graphical data and summaries in an organized way for user-requested data	D	Text Data Display
Create data visualizations for the user-requested data	D	Data Visualizer
Establishes a connection to the Ethereum Blockchain. Accepts data requests and user data, and returns the raw data	D	Smart Contract
Analyses the raw data for the requested measurements	D	Data Analyzer
Container for user's authentication data (individual and third-party users)	K	Key
Verifies that a user with appropriate credentials exists. If not, inform the user and proceeds accordingly. Obtain permission for third-party login.	D	User Authenticator
Coordinate actions of system concepts and user requests. Responsible for data retrieval and transfer to concepts. Refresh data periodically and log out users after prolonged time.	D	Controller
Holds account information of a specific user and provides complete flexibility in managing users own data.	K	Account
Stores account data, user data and collaborates in all activities related to data visualization, administration and storage.	K,D	Database

ii. Association definitions

Concept Pair	Association Description	Association Name
User-Interface (UI) <-> Key	User enters their login information or new user information on the UI	User Credentials
Key <-> Authenticator	The authenticator takes the user's information and prepares verification request, which is sent to the controller.	Prepares Request
Authenticator <-> Controller	(1) Controller receives verification requests, used to invoke the appropriate smart contract. (2) Controller informs authenticator of successful login	(1) Conveys Request (2) Conveys result
Controller <-> Data Request	Controller receives a request for data. It prepares a formal data request	Prepares Request
Controller <-> User Data	Controller receives user data. It prepares a formal data upload request	Prepares Request
Controller <-> Smart Contracts	(1) Controller generates a request to invoke the appropriate smart contract for data retrieval (2) Controller receives raw data from the blockchain	(1) Generates Request (2) Receive Data
Controller <-> Data Analyzer	Controller passes raw data to the data analyzer	Conveys Data
Analyzer <-> Data Visualizer	Analyzer passed processed data to be visualized	Conveys Data

Authenticator <-> User-Interface	(1) Authenticator requests permission for third party log in (2) Authenticator requests valid user credentials	(1) Request Permission (2) Request Credentials
Analyzer <-> Text Data Display	Analyzer passed processed text data to be displayed	Conveys Data
Data Request <-> User-Interface	User enters parameters for data request, which is then contained in a form	Receive Parameters
User Data <-> User-Interface	User enters request for personal data, given the parameters (e.g. user id)	Receive Parameters
Text Data Display <-> User-Interface	Display the non-graphical data in an organized way	Display Data
Data Visualizer <-> User-Interface	Display graphical data	Display Data
Controller <-> Database	(1) Controller generates a data retrieval/upload request (2) Controller receives data from the database	(1) Request Data (2) Receive Data
Authenticator <-> Account	Once authenticated, user account data is stored in Account concept (from database)	Send Account Info
Account <-> Data Request	Data request receives account information to be passed along with the request	Receive Account Info
Account <-> User Data	User Data form receives account information to be passed along with the user data	Receive Account Info

iii. Attribute definitions

Concept	Attributes	Attribute Description
Data Visualizer	Set data	This is setter for the system where the data is converted into visual form to fetch to GUI
	Set user's preference	This allows the data visualizer to be customized to each user's taste
Smart Contracts (Ethereum network)	Invoke a transaction	The server will the blockchain to add a new node in the common chain which typical occur when there is a new user or old user update their data
	Add parameters	This adds flexibility to the system as the administrator adjust the system based on the user's preference
	Get Data	An entry or interface between the server and the blockchain to extract the data for the user
Authenticators	Get key	This is for the getter and setter for the front and back to authenticate and distinguish between the user and third party
	Set key	
Controller (server)	Create new account	This shall allow the new users to register to the service
	Log out timer	Used to trigger automatic logout after idle time
	Get key	Together with the Authenticator forms the login mechanism
GUI (or web page)	Post	This is the primary way that the front interacts with the server and the Ethereum network
	Get	
Key	Credentials	User's identification information, such as user ID and password
User Data	User Identity	User's identification information
	User Information	User's health information to be uploaded

Data Request	User Identity	User's identification information
	Search Parameters	Parameters indicate the wanted data/ measurements
Text Data Display	User Preferences	Customization parameters for results
	Set Data	This is setter for the system where the data is converted into organized text form to fetch to GUI
Data Analyzer	Parameters	Parameters indicate the wanted data/ measurements
Database	Accounts	Record of existing accounts
Account	User Identity	User's identification information
	User Information	User's health information

Iv: Traceability Matrix

Traceability Matrix

	Use Cases									
Domain Concepts	Login	Logout	Display	Compare Data	Authorization	Register	Public Access	Input	Data Administration	Notify
Controller	x	x	x	x	x	x	x	x	x	x
GUI			x					x		
Key	x					x				
Smart Contract				x					x	
Authenticator	x		x							x
Website	x	x	x		x	x	x	x		x
Server									x	
Data Request				x				x	x	
Web Framework			x						x	
Account	x	x			x	x		x		
Credentials					x				x	
Database	x	x	x	x	x	x	x	x	x	x
Data Visualization			x	x					x	
Data Analyzer				x					x	
Text Data Display			x						x	

6.b System Operation Contracts

Name:	Login
Responsibilities:	To have the user access his or her account.
Use Case:	1
Exception:	Password is wrong, or user does not have an account.
Precondition:	User has an account. System prompts user for both username and password.
Postcondition:	User is now logged in and able to use and modify the account.

Name:	InputData
Responsibilities:	To have the user put in health information.
Use Case:	UC-3
Exception:	User does not have an account. User does not put in their correct information.
Precondition:	The user has access to the internet to interact with the system. The system asks for and has ways to fill out all the necessary information
Postcondition:	The information will be stored in the system to be used by the blockchain. The user can view their information.

Name:	ReceiveDataForUser
Responsibilities:	System will visually display to the user his or her data, when the user prompts the system for data.
Use Case:	4
Exception:	None

Precondition:	User has logged in/been authorized.
Postcondition:	User's health data will be displayed graphical representations.

Name:	CompareData
Responsibilities:	System will retrieve health data from the blockchain and compare this to that of the user.
Use Case:	5
Exception:	None
Precondition:	User has logged in.
Postcondition:	The app/system will display the user's health data compared to the overall population's health data.

6.c Mathematical Model

A large amount of population data will be collected on our blockchain of public data. Our job will be to present this data to the user via basic statistical methods.
For example, we will present different variables of data over a period of time via line graphs and show changes in slope over periods of time. For personal data, we can show change of physical characteristics over a period of time. The following graphs are samples given by the open source Apache ECharts library.

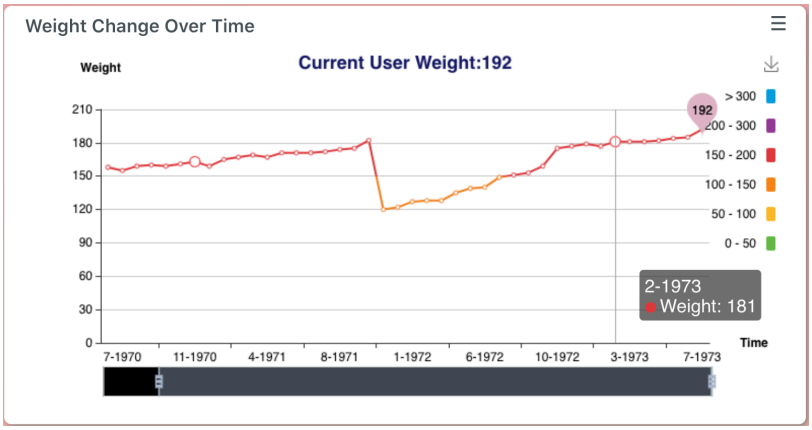


Figure 5.c.1

We can also present bar graphs whilst displaying standard deviations, averages, modes, and medians on those graphs.

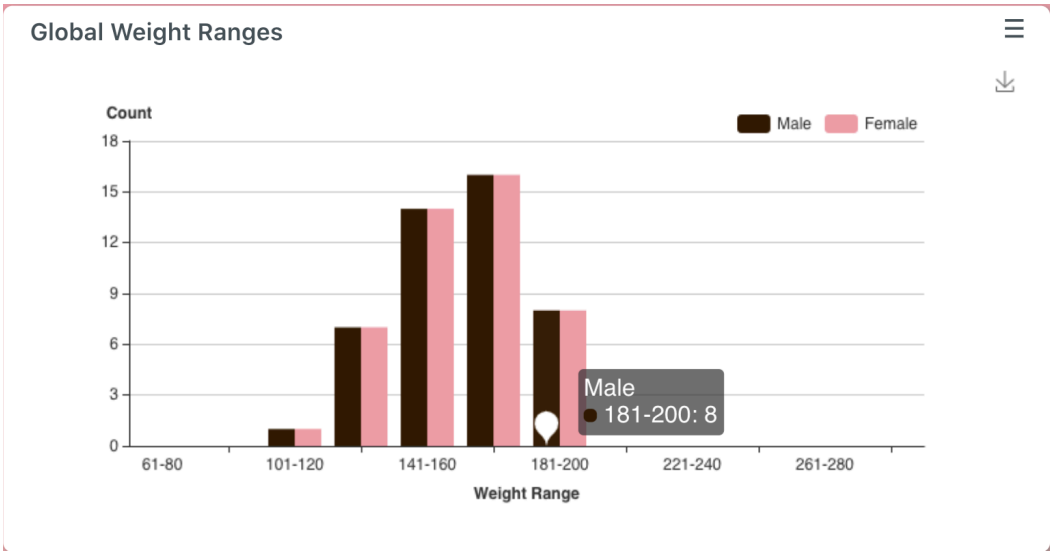


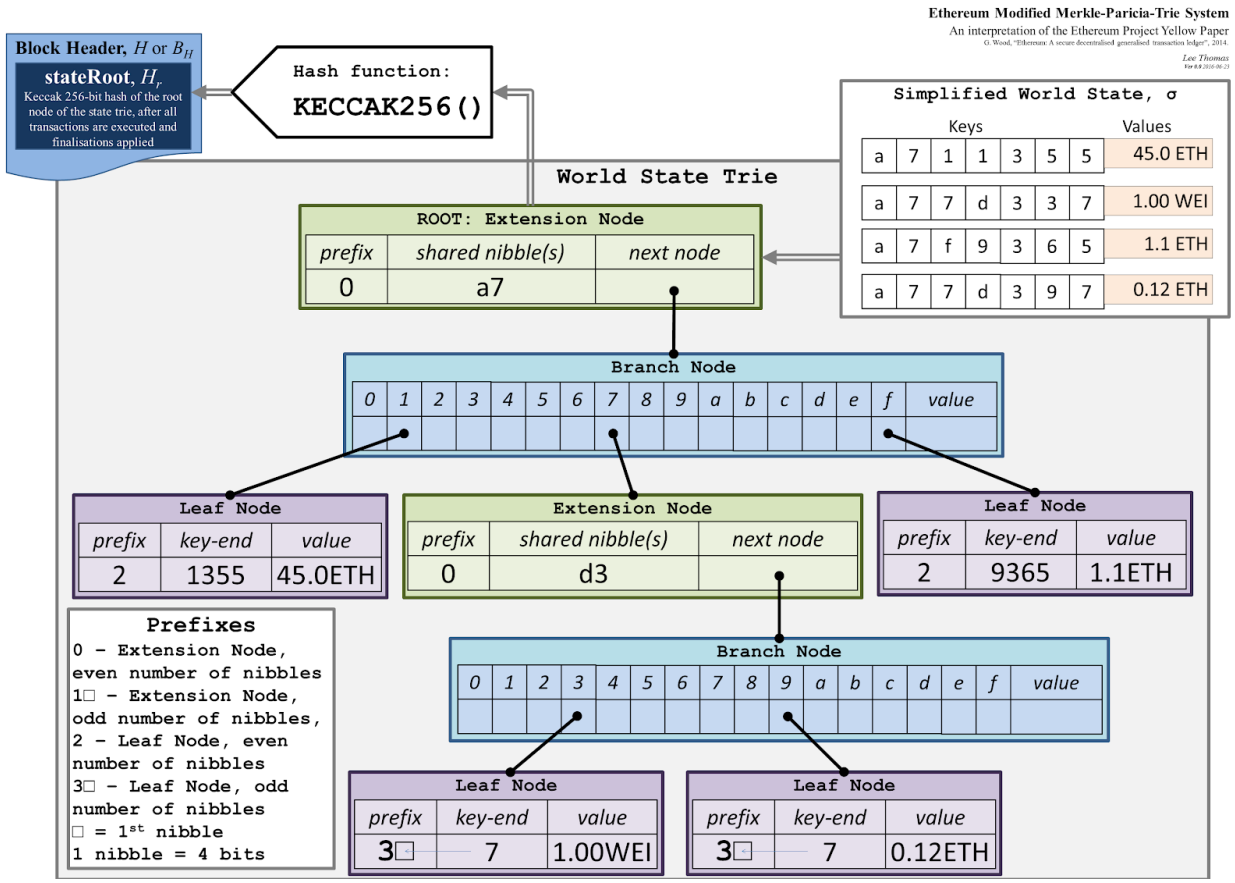
Figure 5.c.2

We may also choose to record the location of the user to show frequency of a certain characteristic in a geographic location.

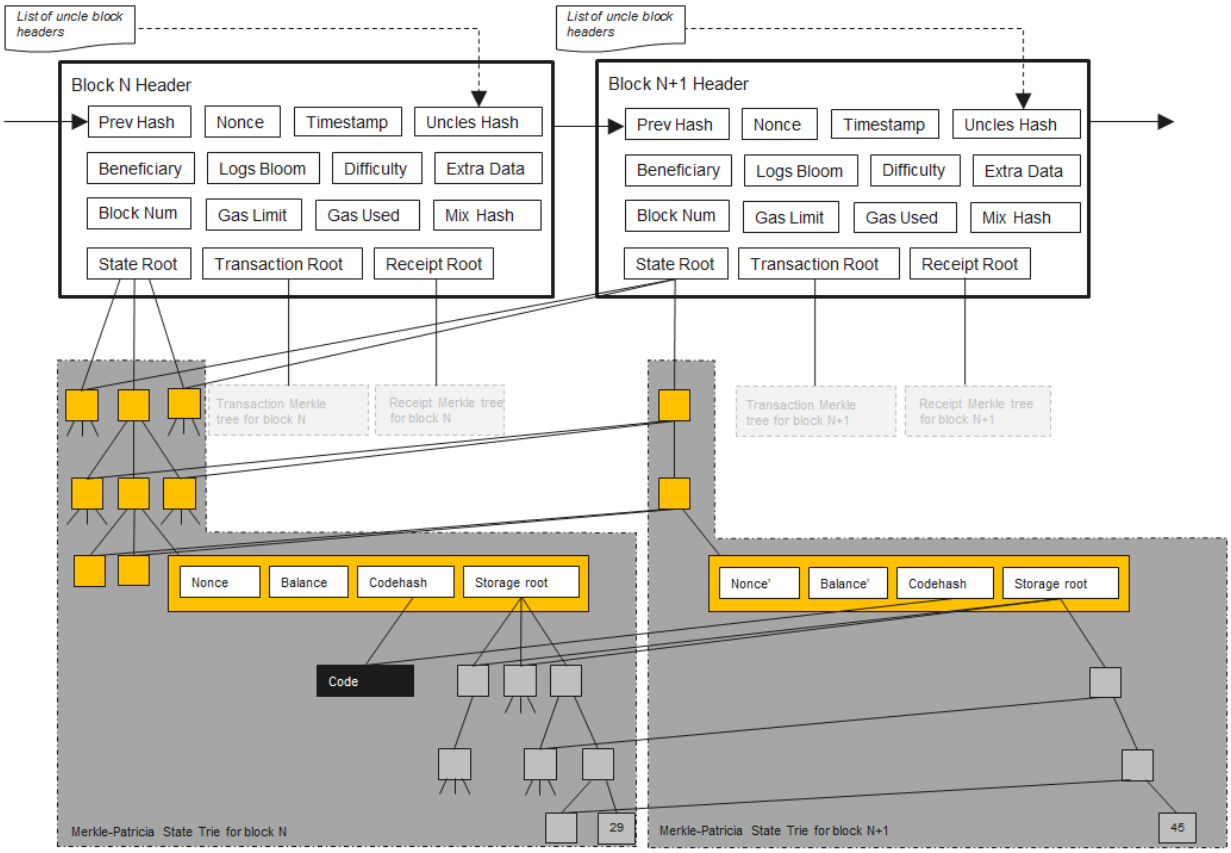
Figure 5.c.3

To support user data privacy, we will implement a cryptographic encryption algorithm to all data. For now, we are planning on using the asymmetric algorithm, RSA, to secure user data. As a private key, we can use the same private key that is used for access to the Blockchain. Thus only users that inputted the data can read from that data. For global data that is visible to all users, we will store it in plain text in the blockchain.

The server and blockchain interaction is dependent on the implementation of Ethereum network which is defined in the Ethereum yellow paper. Ethereum is a project which attempts to build the generalized technology; technology on which all transaction-based state machine concepts may be built. The execution model is specified through a formal model of a virtual state machine, known as Ethereum Virtual Machine. The machine has a simple stack-based architecture. The word size of the system is 256-bit.
Furthermore, Ethereum implementation relies on Merker trees . Although it is definitely theoretically possible to make a blockchain without Merkle trees, simply by creating giant block headers that directly contain every transaction, doing so poses large scalability challenges that arguably puts the ability to trustlessly use blockchains out of the reach of all but the most powerful computers in the long term. Thanks to Merkle trees, it is possible to build Ethereum nodes that run on all computers and laptops large and small, smart phones, and even internet of things.
The data structure and implementation of ethereum



Merkel tree in Ethereum



7. Interaction Diagram

The following are system sequence diagrams for our most important use cases.

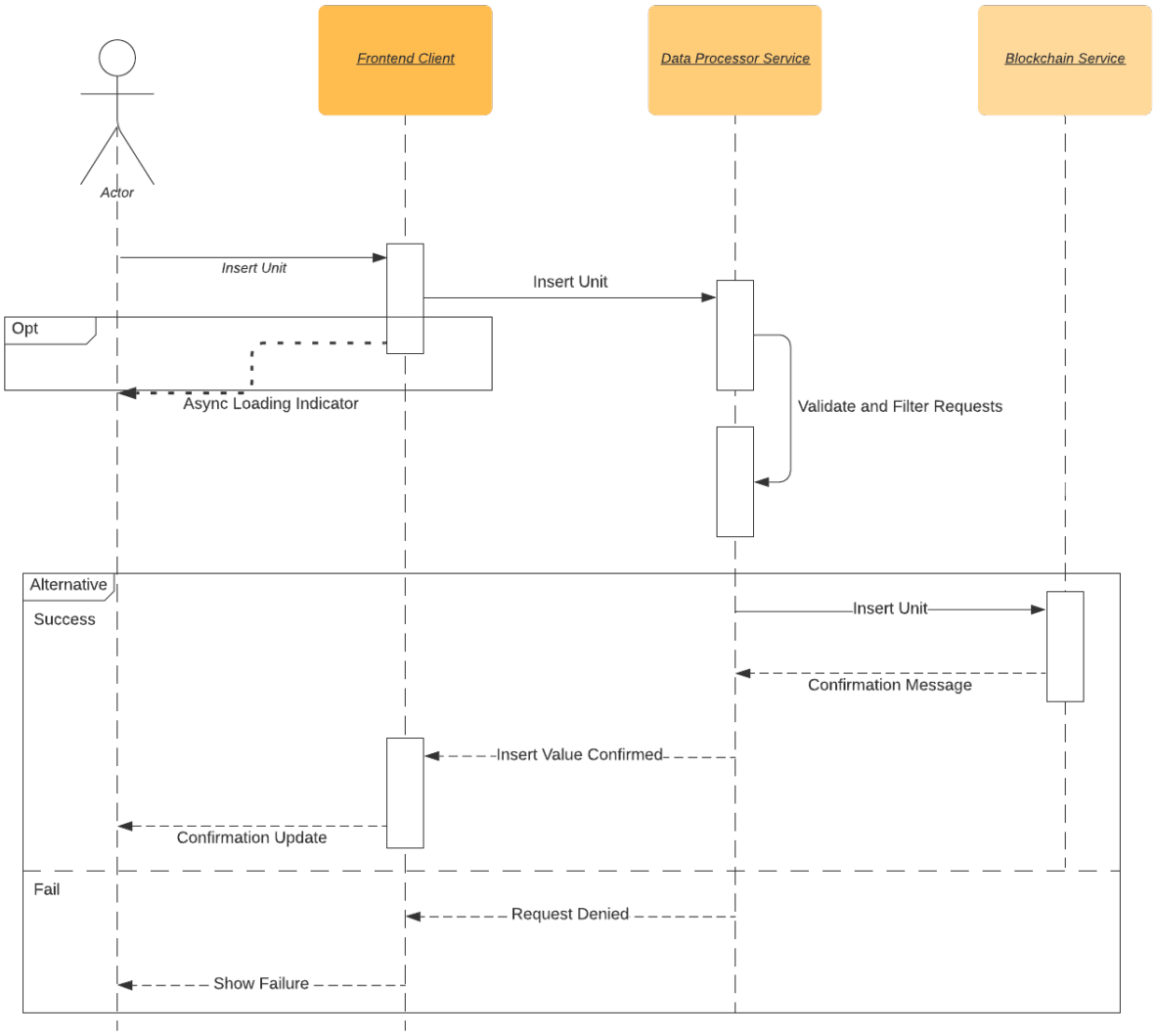


Figure 3.c.1.1 UC-2

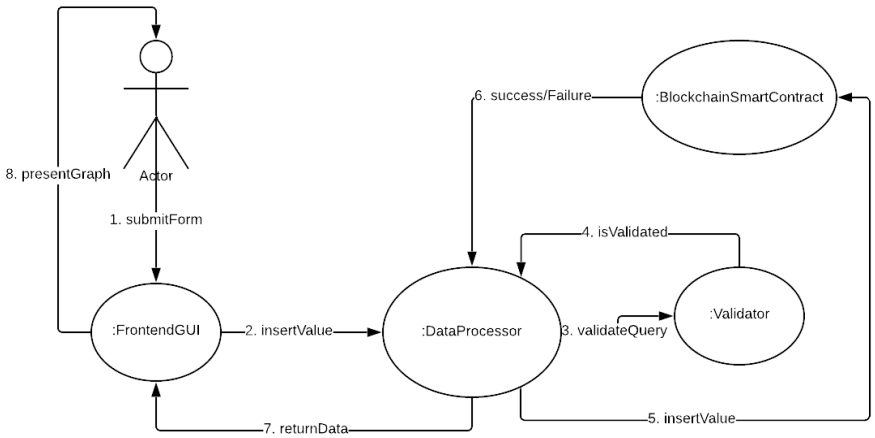


Figure 3.c.1.2 UC-2

This first use case shows the sequence of the insertValue use case where users will need to insert some sort of value to the blockchain network for personal and global use. The three participating systems are the Frontend Client (GUI), the centralized backend server (Data Processor Service), and the Blockchain (Blockchain Service). First the user (Actor) interacts with the Frontend Client to insert some value via a form. A loading indicator is optionally shown depending on implementation. The client sends a request to the backend service which validates/filters any requests that are out of the norm (outliers in the data). Then, if data insertion in the blockchain is successful, the chain will bubble back to the actor, alternatively failure will flow back to the user as well. In terms of design principles, we tried to make the most of the system as stateless as possible; only keeping state in the blockchain. In addition to this, we used a three tiered architecture pattern to separate the concerns of the frontend, the processor, and the storage.

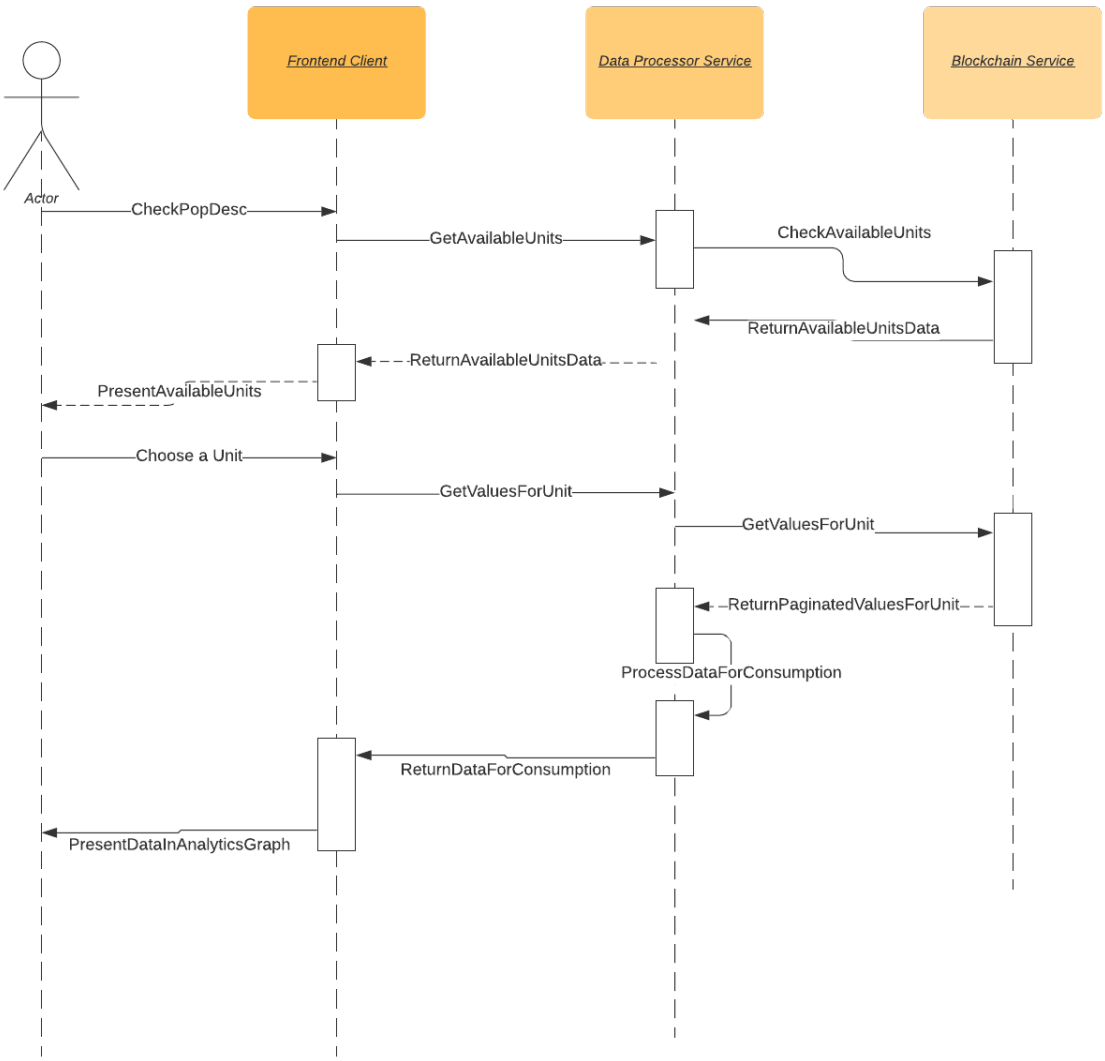


Figure 3.c.2.1 - UC-4

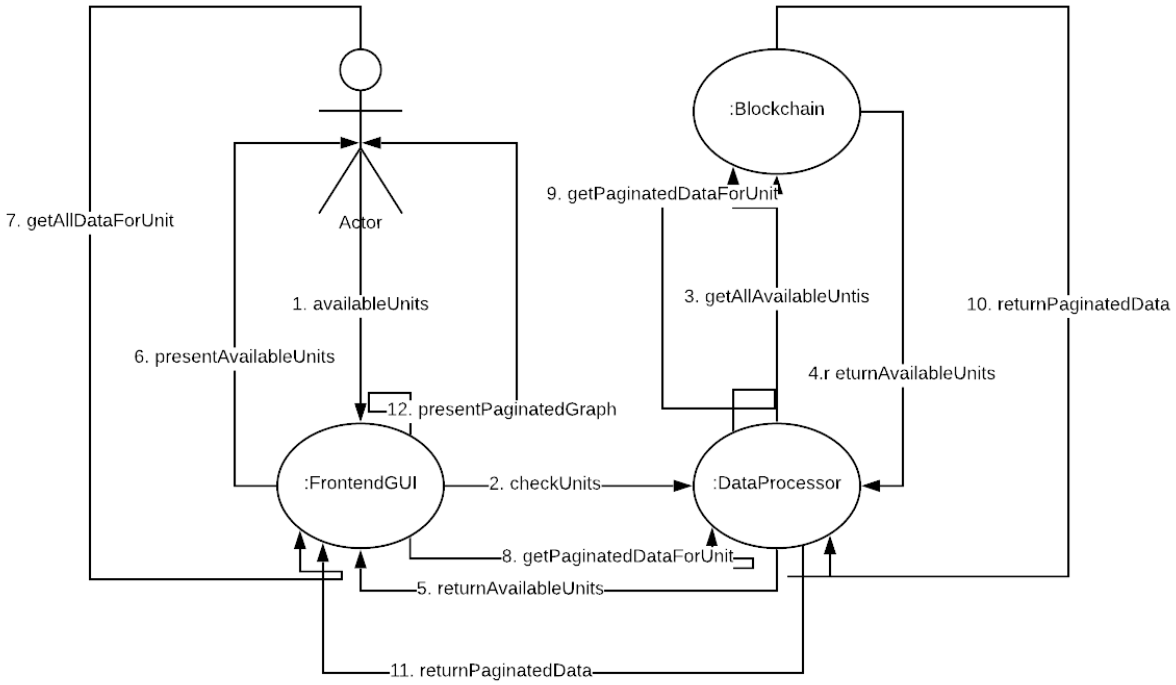


Figure 3.c.2.2 - UC-4

This use case shows how data of population descriptors are retrieved from the blockchain. The three participating systems are the Frontend Client (GUI), the centralized backend server (Data Processor Service), and the Blockchain (Blockchain Service). The user (Actor) first asks what units of descriptors are available to the client, which traverses through the centralized backend, to the blockchain. This returns all available units which the user chooses from. Then the user asks the user to gather data for a specific unit and this traverses through to the blockchain into a paginated data request. Depending on the graph that the page wants to render, different pagination counts may be chosen. Then the backend formats the data in a format that the frontend wants to consume, at which point the data traverses back to the frontend client to render to the user. For brevity, UC-4 also has a very similar sequence but only involves user specific data (in which case a user context will be passed to retrieve the data).

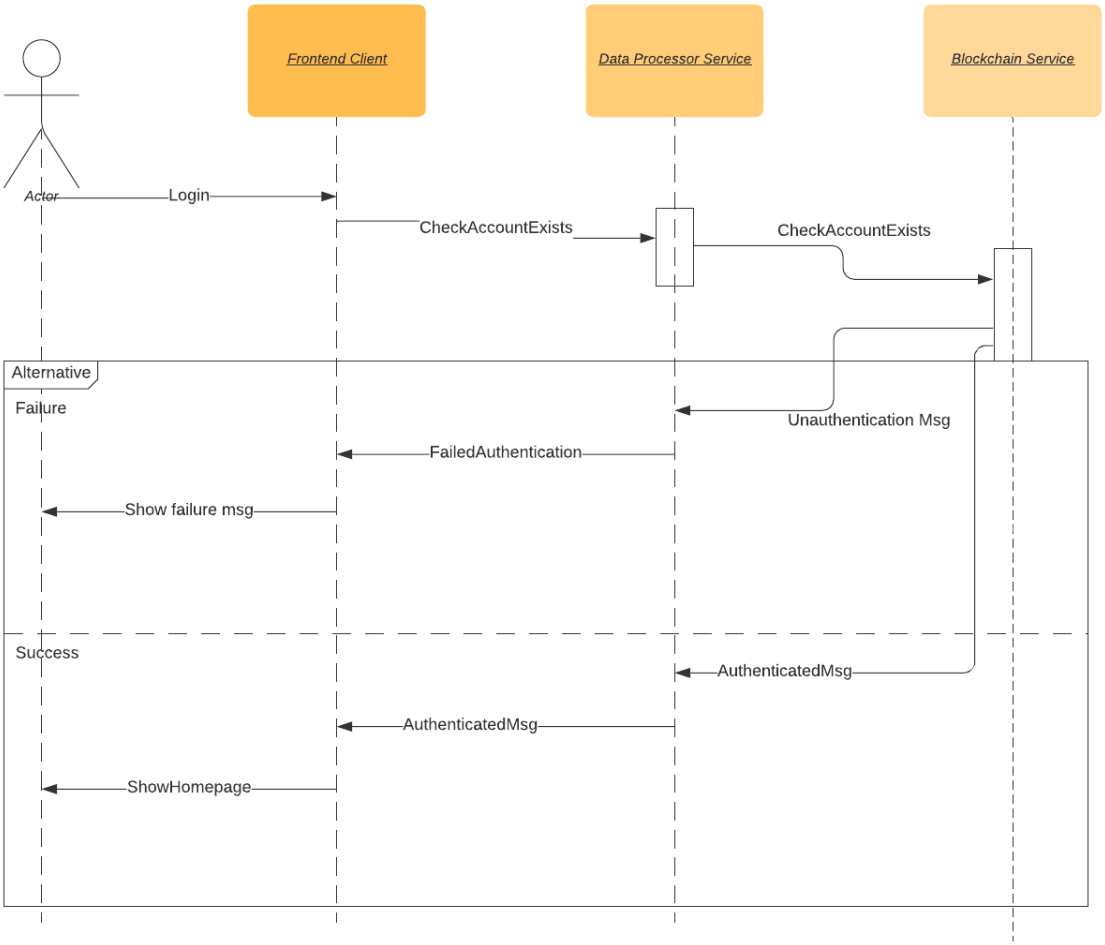


Figure 3.c.4.2 UC-1

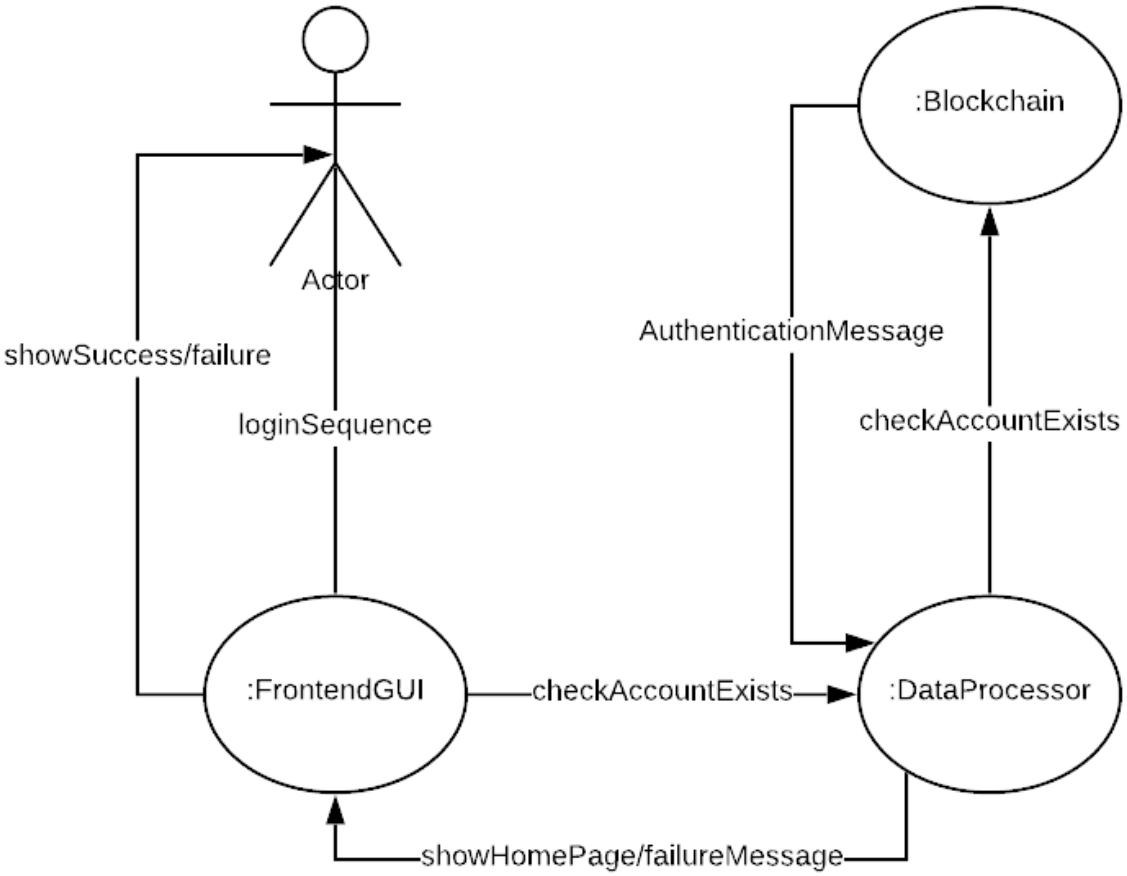
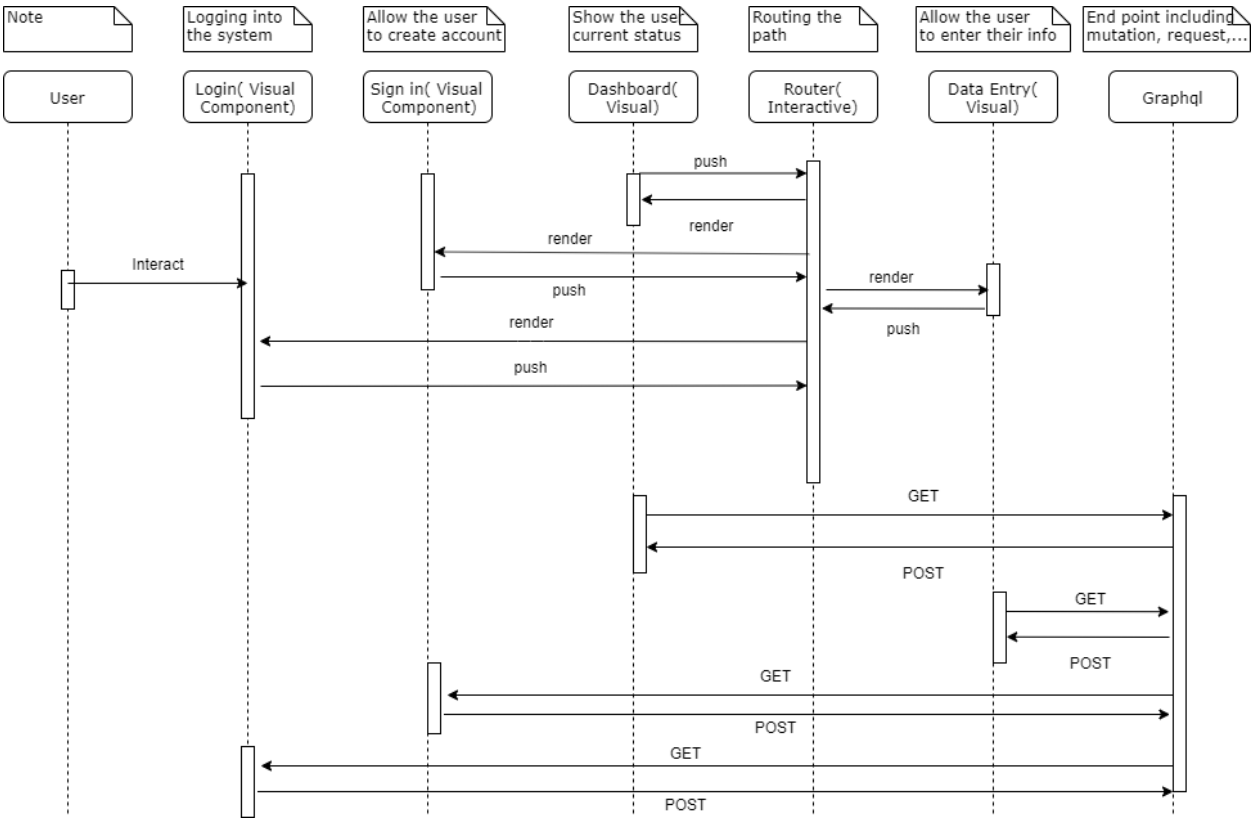


Figure 3.c.4.2 UC-1

This use case pertains to logging into the system and identifying the user for Authentication/Authorization. This is an estimated sequence as the full implementation of the authentication must be actively researched. However, the only place we expect things to change is in the CheckAccountExists method in DataProcessorService and BlockchainService. Failure and success produces sequences that bubbles up to the user.

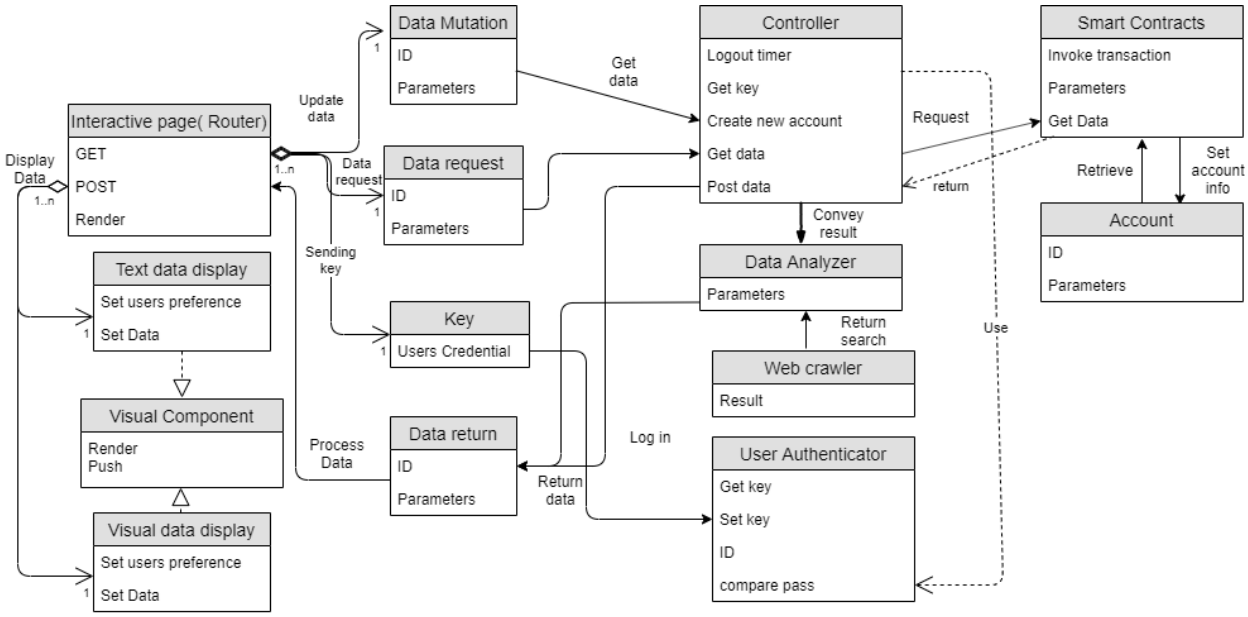
Model View Controller



This design pattern is popular in UI because it decouples responsibilities between the business logic(model), the UI(view), and the user interaction(controller). The React frontend is no exception. Thus, the design frontend is centered around this design pattern.

8. Class Diagrams and Interface Specification

Class diagram



Compare to the domain analysis in report 1. We have rearranged the class into 4 stages: frontend, transport layer (graphql), the backend, Ethereum network. Most of the concepts remain the same as in report 1. After an iteration, we decided to add in some data objects to represent the transport layer in practice. After report 2, we implement more feature leading to more changes in the class diagram

Class	Operation and Attribute	Description to in the class diagram
Text Data Display	Set user preference	Customize text display to the users' preference then send it to the controller
	Set Data	The server will pass the data into the method
Visual Data Display	Set user preference	Customize the visual display to the users' preference then send it to the controller
	Set Data	The server will pass the data into the method

Interactive page	Get	The primary getter and setter for the front end to connect to the backend
	Post	
Visual Components	Push	Notify the controller to change page
	Render	Render each independence page
User Authenticator	Compare pass	Comparing the password to verify the user's id for the controller
	Set key	Receiving the user credential from the key
	Get key	Get the hashed key to compare with the password
Data Analyzer(Data processor)	Parameters	Analyze the data to determine the mode, median or other important indicators and sending it back to the server
Data Return/ Mutation/ Return	ID	The user's id
	Parameters	The request and return data
Smart Contracts	Invoke transaction	A transaction usually occurs when a new user signup or the old user update their data
	Add parameters	Add flexibility to the system as the administration can add new parameters
	Get Data	The interface for the blockchain to connect to the blockchain
Key	User Credential	User's identification information, such as user ID and password and passing to the controller
Account	ID	The user's id
	Parameters	The user's health information
Controller	Logout timer	After a certain amount of time, the system will log the user out.

	Create new account	Set up the new account when request on coming in for the page and invoke a transaction
	Listen	Receive request and send the data from the interactive page via the transport layer
	Set data	

Design patterns

Model View Controller speed up the development process by providing tested, proven development paradigms. This allows our team to focus on other features and reduce communication overhead instead of reinventing the wheels. Freshly written code can often hide subtle issues and require communication overhead. Reusing design patterns helps to prevent such subtle issues, and it also improves code readability for coders and architects who are familiar with the patterns.

Object Constraint Language(OCL) Contracts

Display	
Invariant	Login credentials and data input that the user inserted.
Pre-Condition	If the valid credentials are inserted request is made to the blockchain service and data is retrieved.
Post-Condition	The user will receive processed data from the blockchain and will have graphical representations of his or her personal data

Registration	
Invariant	User must be an unregistered user with the system.
Pre-Condition	All the mandatory personal details asked should be entered by the user.
Post-Condition	User gets registered with the system and is ready to proceed with his/her health monitoring.

Data Processor	
Invariant	Analyze the data to determine the mode, median or other important indicators and sending it back to the server
Pre-Condition	Data retrieved from user input.
Post-Condition	Heart rate averages and weight ranges of the population is determined.

Session Controller	
Invariant	Should receive a request for logoff, must track the time user is logged into the system and request for log- off generated after specified period of inactivity.
Pre-Condition	User is logged into the system.
Post-Condition	User Session should be logged out after inactivity for a considerate amount of time and maintained otherwise until requested log-off

9. System Architecture and Design

9.a Architectural Styles

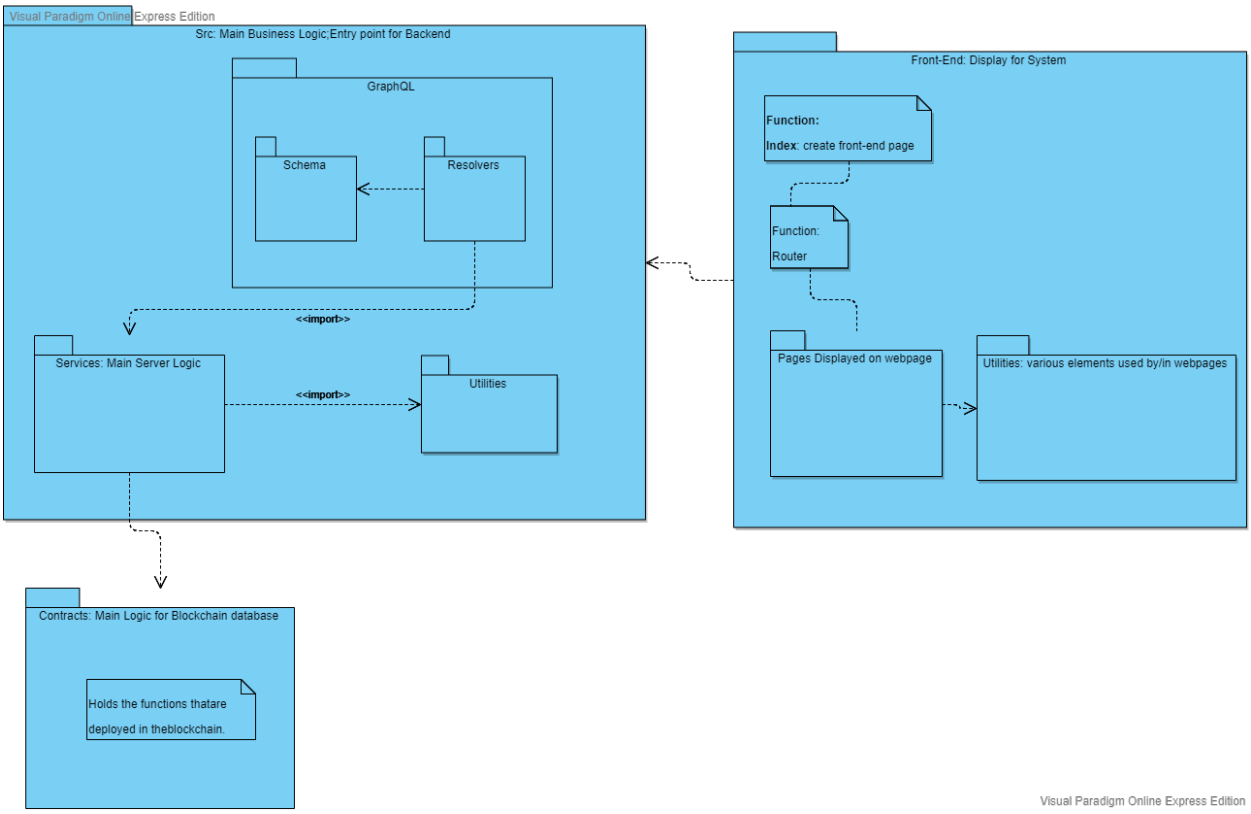
Our system employs different types of architectural styles across different scope. From the scope of the entire system a Three-Tiered architecture is implemented. The client-side user interface acts as the presentation layer where a user can make requests and view the results. User requests are handled by the application layer, a Node JS Express server. The model in this architecture is composed of the various files and functionalities of the GraphQL api as well as the Smart-Contracts for the Ethereum Blockchain. The GraphQL api interacts with the database (the blockchain in our case) and handles all queries. Furthermore, GraphQL provides the capacity to apply business logic to the data from the database through resolvers. Resolvers are functions that can either simply retrieve data from storage or retrieve data and manipulate them as appropriate. The Smart-Contracts are called through the resolvers in order to obtain the data from the blockchain, they are a tool necessary specifically for the blockchain. Through the user-interface, Express server, and GraphQL api a MVC architecture is obtained.

While from a larger scope, our application is Three-Tiered, inside each of the tiers (particularly the client and server), we employ a Model-View-Controller architecture. The Express server file resides within the same package as GraphQL schemas and functions. This may be viewed as the controller and model being one large backend server. Since the controller and model share a package they can be treated as a single entity in which case this becomes a client-server architecture. However, the controller and model functionalities are kept separate and one does not depend on the other for functionality.

The model as a collection of several modules and files itself has a layered architecture. The model is composed of several functions and schemas relating to the GraphQL api and Smart-Contracts for the blockchain. The highest layer is comprised of the “serveGraphQLRequest” contained within the graphql index file. This function receives the query request and then passes it onto the different resolvers. The GraphQL resolvers are a layer below and they process the query to determine which data is requested. The resolvers then call on the Smart-Contracts through the services layer. The services layer receives function calls from the resolvers then sends back results, sometimes by calling on Smart-Contracts. The Smart-Contracts interact directly with the blockchain and obtain the information requested by the resolver and returns. The resolver can then carry out logic with the data and then pass it back to

“serveGraphQLRequest” which will send the results to the controller to be displayed. Each of these layers interact only with the adjacent layer, the Smart-Contracts can only be accessed through the services layer which is accessed only through the resolvers and so on. There is a clear hierarchy which establishes a layered architecture.

9.b Subsystem Package Diagram



9.c Mapping Subsystems to Hardware

Our system breaks down to three subsystems that will run in three different sets of hardware. The client side front end, the web server where the controller and several services for the database reside, and finally the blockchain.

Client-Side Frontend: This is a collection of webpages and user-interfaces that the user will interact with. This subsystem will reside in the user’s device through a web browser, and will use whichever device the user is accessing the website with.

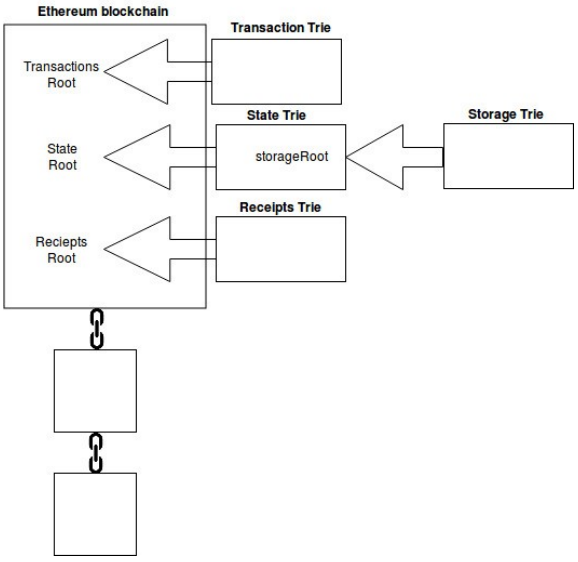
Web-Server: The web-server will contain the Express Server and some other services that will be used to process the data. The webserver will also act as an intermediate

between user interface and the data in the blockchain. This will have to be hosted on servers specifically for web-servers. We do not yet have anything specific in our plans but services such as Amazon Web Services or Github can provide the necessary hardware and services to host the web-server.

Blockchain: Essentially the database of the system, will house the data collected from users. Blockchain is a decentralized peer-to-peer system that exists with many devices. Each device in a node contains data. Since blockchain as a concept requires many devices it cannot be mapped to a single piece of hardware, rather the requirement for a blockchain network would be as many computers as possible. More computers mean more devices to store data and more security. Blockchain does not require any special hardware, regular computers are adequate, the requirement is to have several to many devices. For an Ethereum blockchain specifically, devices that are already part of the Ethereum Blockchain network will be necessary.

9.d Persistent Data Storage

Users in our system will be interacting with the ethereum blockchain to manage their data. Ethereum uses a tree data structure as illustrated in the figure below.



Underlying the structure:

State trie

There is only one state trie. The state trie contains a key and value pair for every account that exists on the Ethereum network. It constantly gets updated.

Storage trie

Each Ethereum account has a storage trie. All the contract data lives here.

Transaction trie

Each block in the ethereum network has its own Transaction trie . Transaction data is stored here.

Merkle Tree:

Ethereum stores two types of data permanent data and ephemeral(temporary) data. Transactions are considered permanent data and stored in the transaction trie and is never altered. Ethereum account address is considered temporary data and is stored state trie.

Blocks are stored on a multi-level data structure. The hashed address of a block points only to the block header which contain the timestamps, previous block hash, and the root hash of the merkle tree data structure that has all the transactions in the block. Each node of the tree is the hash of its children. A “full node” following the merkle tree protocol takes up about 15+ GB of disk space.

Ethereum also supports a protocol called "simplified payment verification" (SPV). This allows “light nodes to exist” which download the block headers and only download the branches with transactions that are relevant to them. Proof of work is verified on the block headers.

9.e Network Protocol

We will be using a backend proxy server that retrieves data from the blockchain. Reading and processing data from the blockchain is expensive, and results in high latency, as a result we will use the proxy server for caching and processing. The server will make JSON-RPC (Remote Procedure Calls) to communicate with the blockchain.
The frontend will use the query language Graphql to query from the backend proxy server.

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol

To talk to an ethereum node from inside a JavaScript application use the web3.js library JSON can represent four primitive types (Strings, Numbers, Booleans, and Null) and two structured types (Objects and Arrays).

Ethereum runs on a decentralized network. There is no central server. It's a peer-to-peer system. Each node can read and send data to other node.
All nodes can request from another node information about Ethereum's current state such as smart contract, account balance. To facilitate this form of communication,
Ethereum uses a Kademlia-like protocol for node discovery. Each node has an id which is hashed. Each node stores a table of known nodes. To look for a node
the node can asks other known nodes to look for a node.

The Design Philosophy behind Ethereum.

Simplicity: The ethereum protocol is designed to be as simple as possible at the cost of time efficiency and storage so that anyone can implement their own blockchain.

Modularity: Ethereum is designed to be modular and as separate as possible. If you made a small protocol modification in one location, the application stack will continue to run.

*Non-discrimination and non-censorship:*The protocol will not restrict or prevent specific use cases. Ex. You can run an infinite loop on the blockchain as long as you are willing to pay the per-computational-step transaction fee.

9.f Global Control Flow

The system is event driven. The user interacts with the system by first registering his username and password. The user login and the next step is the user input the necessary data to the system and has access to view their data and standing compared to public data which is obtained using blockchain service and is presented in form of bar charts and line graphs. The system does not log anyone in until the initial event is driven by an external request. For time dependency, the backend of the system periodically processes pending data.

To tackle the issue of concurrency in our application. Considering the use of blockchain, having many threads accessing the same memory(multithreading) can produce race conditions that are very hard to reproduce and fix.

For a better solution than a multithreaded approach, our code runs things in parallel, which refrains us from creating new threads and allowing the need to sync them. The virtual machine and the operating system run the I/O in parallel and for sending data back to the Javascript, the JavaScript part is the one that runs in a single thread.

Our code consists of small portions of synchronous blocks that run fast and pass data back end and front-end processing functions. This process is faster since it doesn't block the execution of other pieces of JavaScript. Our application just invokes the function and does not block the execution of other pieces of code. It will get notified through the callback when the query is done, and we will receive the required user health data.

To decrypt the values of the JavaScript query that returns a few thousand results of different user's health data. Task at the back end is performed in such a way that it will split into smaller synchronous code blocks that will notify Node.js to split the task into smaller chunks using callback functions and so it can continue executing pending things that are in the queue after receiving callback.

We are using asynchronous I/O approach. The code consists of small portions of synchronous blocks that run fast and pass data to different places where needed. This approach doesn't block the execution of other pieces of JavaScript.

There are no timers in the system with no time constraints, since the system runs on blockchain data. Any user who is familiar or new to the system can input their data by logging in and compare it with the public.

9.g Hardware Requirements

- Processor (CPU) with 2 gigahertz (GHz) frequency or above.
- A minimum of 2 GB of RAM.
- Monitor Resolution 1024 X 768 or higher.
- A minimum of 20 GB of available space on the hard disk (for blockchain nodes).
- Keyboard and Mouse or compatible pointing device.
- Network card should be enabled and installed to access internet.
- A bandwidth that can successfully load a modern Javascript application. Since there is no time sensitivity, a recommended 1 mbps should be more than enough.

10. Algorithms and Data Structures

10.a Algorithms

RSA (Rivest–Shamir–Adleman)
We will be using RSA to encrypt the user's data. Rsa is an asymmetric cryptographic algorithm, meaning that it uses a public and private key system. The public key can be shown to everyone, but the messages encrypted by the public key can only be decrypted by the private key. Here is a breakdown of how RSA encrypts data.

Generate key:

1. Generate two large unique prime numbers p and q
2. Compute $n = p \times q$ and $\varphi = (p - 1) \times (q - 1)$
3. Select a random number $1 < e < \varphi$ such that $gcd(e, \varphi) = 1$
4. Compute the unique integer $1 < d < \varphi$ such that $e \times d \equiv 1 \pmod{\varphi}$
5. (d, n) is the private key
6. (e, n) is the public key

Encryption

1. Represent a message as an integer m in the interval [0, n-1]
2. Send out the encrypted data c

$$c = m^e \bmod n$$

Decryption:

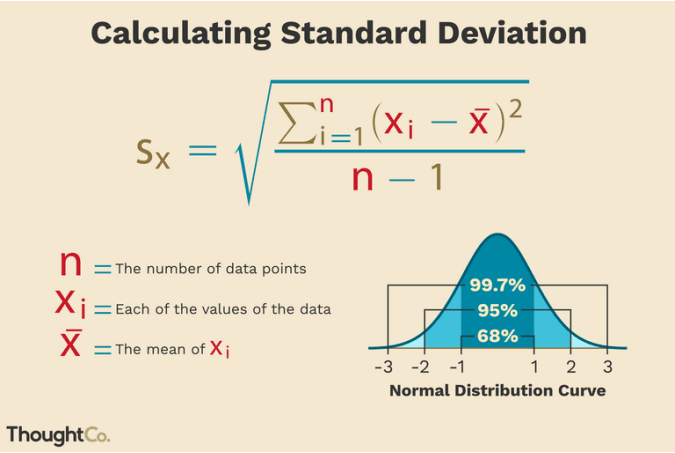
1. Decrypt the key using

$$m = c^d \bmod n$$

We will also be doing computations on user data. We will be finding the average and standard deviation of the data.
Average is the sum of values divided by the number of values.

Standard Deviation

The standard Deviation measures how spread out the numbers are in a set of data.



10.b Data Structures

Data structures that have relevance to the blockchain itself should be referenced in the Ethereum Yellow Paper. We will not discuss the use of Merkel Trees or Hashes that references previous structures as it is not directly related to our work. Rather, we can talk about the structures we use in our Smart Contracts (Ethereum code running on the EVM) to store data about users.

Before we jump into specifics, it's important to understand how Hash Tables work, particularly in the Ethereum blockchain. It is a structure that offers fast access to a particular key and maps to a particular value. The structure does this by quickly hashing the key and looking at the key up on an indexable array. In ethereum, hash tables are virtually initialized such that every possible key exists and is mapped to a value whose byte-representation is all zeros: a type's default value. The similarity ends here, though: The key data is not actually stored in a mapping, only its keccak256 hash used to look up the value. Because of this, mappings do not have a length or a concept of a key or value being "set". Mappings are only allowed for state variables (or as storage reference types in internal functions).

User Descriptor Smart Contract
Our smart contract that keeps data relevant to a particular user keeps user data on a few important data structures. The most important data structure used is the concept of a Hash Table (Hash-map or Dictionaries in many languages). For a user's data to be secure, we created a Mapping of all user's private addresses to their data. This is known to us as the User Descriptor table. When invoking to contracts to retrieve data for a particular user, only the user's ID is used to access this table. As smart contracts are read only, as long as the methods defined use the current user's address to access the data, user's data are opaque to other users.

Our user's are expected to store data for an infinite number of units; so what we have decided to do is create a second Hash Table that is used as the value of the User Descriptor table. This second Hash Table, known as the Unit table, is a key of units ('lb', 'inches', etc) to a value that is a growable vector of unit values. The unit values furthermore is a structure that stores a unit value, the timestamp of the submission of that value, and the latitude and longitude (if entered) for that particular submission.

One problem we had with this data architecture was that Hash Tables were not iterable. What this meant was that we could not easily iterate on the Unit table to retrieve the key units for the user. We had to supplement this Hash Table with a growable vector of units that we could present to the user before we accessed the Hash Table with the key unit.

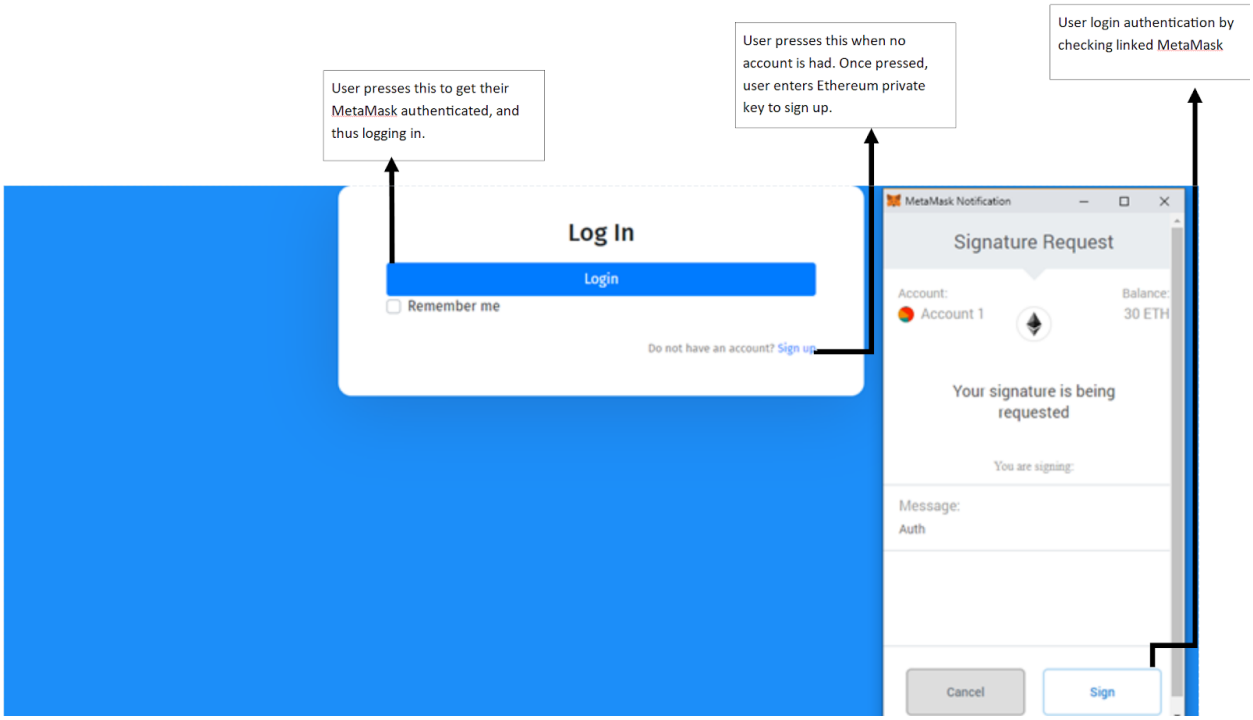
We use the growable vector of unit values to easily index the end of the array and paginate responses to them. Since arrays are contiguous and indexable, pagination is as quick as possible.

As we traverse and transform data structures from the backend to the frontend, pagination data largely become growable Javascript arrays that the charting libraries can easily consume.

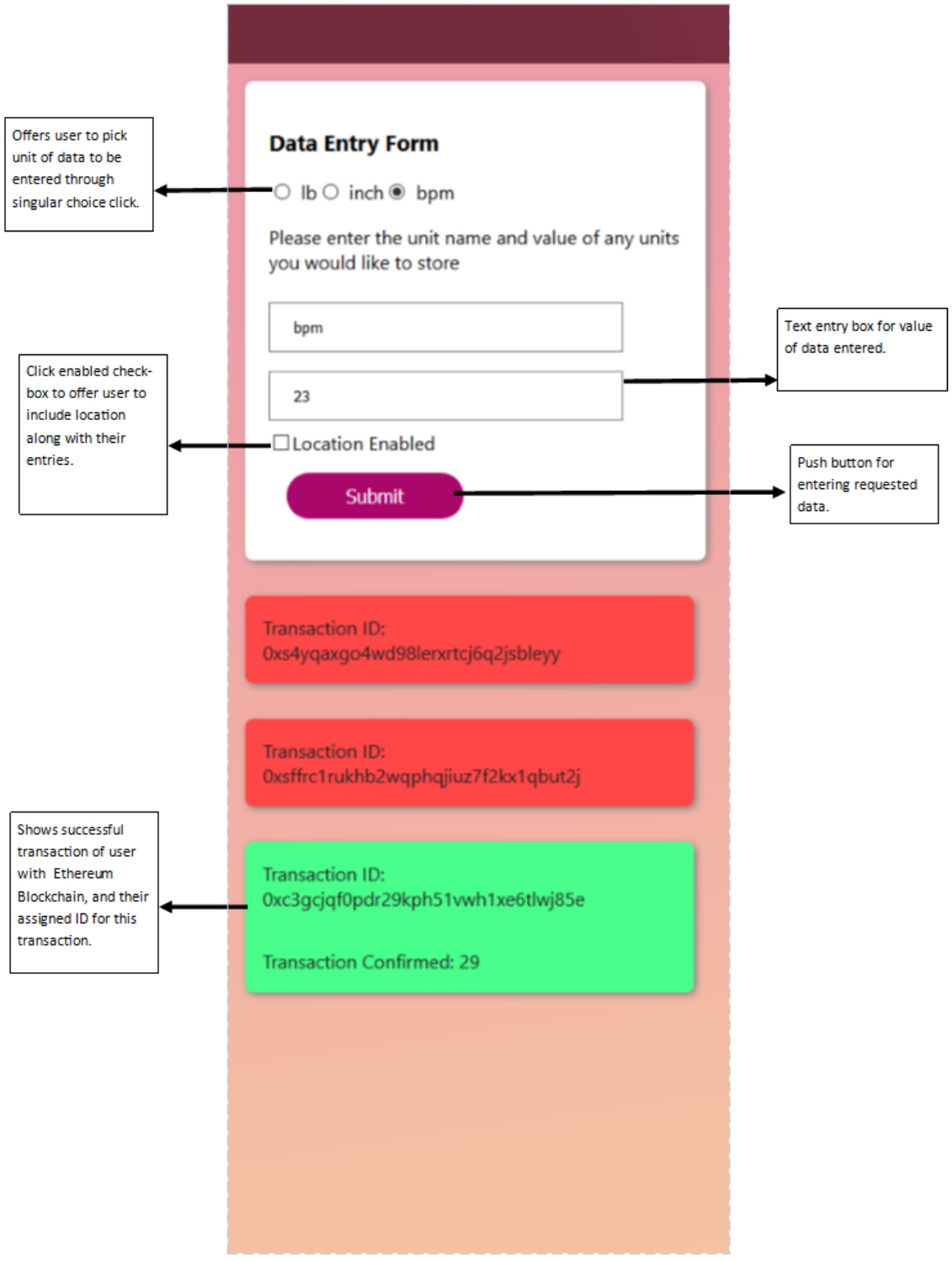
11. User Interface Design and Implementation

11.a Design

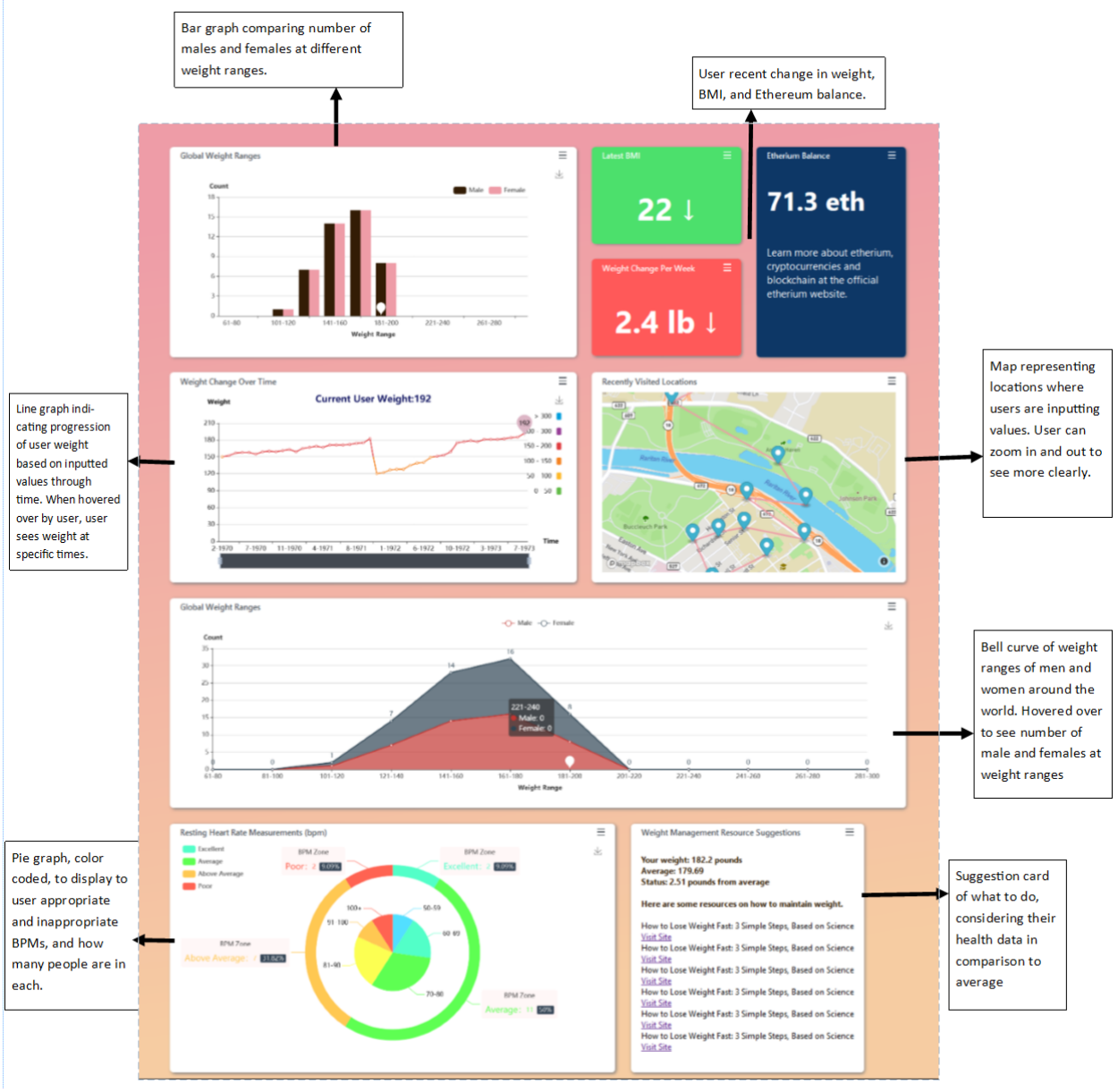
UC-1 Login:



UC-3: User Data Input:



UC-4 - ReceiveDataForUser



11.b User Effort Estimation

UC-1 Login

1.Navigation
A: Click Sign in at top.
B: Click Sign on MetaMask window to right.

Create an Account.

1.Navigation
A: Click Sign up to create an account.
B: Click Sign up.

1.Data Entry
A: Enter Ethereum private key

UC-4 ReceiveDataForUser
1.Click on Personal Data tab on left side
2.Click on available units (Everything else will autofill)

11.c Changes That We Have Made

Area of Change	What changed	Why we changed it
Graphical Visualization of Aggregate Population Data and Non-Graphical Data Display	Rather than offering a way to search for the graph the user wants, we offer all the graphs and non-graphical data on a single page.	This offers the user access to graphs and data that they might not have known about. Additionally, they can consider more information in relation to each other (i.e. consider weight, age, and sex simultaneously).

Signup and login entries	System adjusted to function with MetaMask instead of usernames and passwords. For Login only a button is required that accesses MetaMask, and for Signup user pastes in Private Key	Secures user login better by not allowing for vulnerable email address to be used in the login process, and rather for the secured blockchain to control account creation and control instead.

11.d Ease-of-Use Employed

A large portion of our process was making the User Interface as simple as possible. Certain ways we do so are by providing singular button press input, graphical and non-graphical data in a combined page, the option to scroll over the graphs for more precise numbers, and offering the user the option to save the graphs in a .JPEG format.

Login being condensed to a singular login button, followed by authorization of account use allows for user to have a more effective and efficient login experience. Rather than having users memorize passwords and enter emails, we offer a singular button press to connect to a preexisting Ethereum account.

Graphical and non-graphical data being all presented together offers a singular page to look through, rather than having to navigate through a search bar and find the exact one they need.

Offering the user to scroll over their graphs to see more precise numbers allows the user to understand precise values when wanted, without having to scroll through a table of data or having to approximate based on the graph and its axes.

Offering the user the option to directly save their graphs allows for an easier way of saving graphs that a user might want to share with their friends, family, doctors, etcetera. If not offered, the user would have to screenshot and crop out their preferred parts of the graph, whereas here we offer the user to circumvent that effort by giving them the option to simply save their information directly.

12. Design of Tests

12.a Test Cases

1.

Test Case Description: Test for user descriptors. This test will be used to check the user's inputs for sex, weight and weight units, and height and height units. Related Use Cases: UC-3, UC-4, UC-5 Criteria for Passing: The units of weight must be pounds or kilograms; gender is male, female, other; and height is meters/centimeters or feet/inches.	
Fail Procedure:	Pass Procedure:
First try to input the wrong information, such as typing the incorrect type of units or invalid text, or not writing male, female, or other in gender. The user, after inputting the wrong information will be alerted with an error and will be prompted to change the input to the given valid inputs.	Input the correct information for weight, height, gender. The user's correct information will be recorded by the system in order to create the user's health stats. The user will be able to proceed to use the app, and see the information having been correctly processed in the received data page.

1.

Test Case Description: Integration test for GraphQL api. The test determines if the GraphQL endpoint gives an OK response Related Use Cases: Criteria for Passing: Test finds the status code of GraphQL, which is 200	
Fail Procedure:	Pass Procedure:

GraphQL endpoint does not have a status code of 200. The endpoint does not work and the test fails.	GraphQL endpoint does exist. The test looks for the 200 status code by requesting server to expect this code. The server finds the status code, so GraphQL endpoint works. The test passes.

1.

Test Case Description: Test for correct global descriptors. Test determines that correct information is being presented Related Use Cases: UC-4, UC-5 Criteria for Passing: Test finds that all numbers are accurate	
Fail Procedure:	Pass Procedure:
Graphs and public's data does not accurately represent limited, isolated inputs used for test. Given a set number of predetermined inputs, the received data is not the appropriate presentation.	Graphs and public data accurately represent limited, isolated inputs used for test. Given a set number of predetermined inputs, data portrayed presents it appropriately.

12.b Test Coverage

We are using unit testing and integration testing in this project. This means we are able to account for most errors that may occur, because unit testing may check a component (unit) of the code but integration testing will check for errors that may occur combining these components. We will continue to make more unit and integration tests before the second demonstration to make sure there are no errors. Otherwise we have covered most of the major errors that the application could possibly have.

12.c Integration Testing

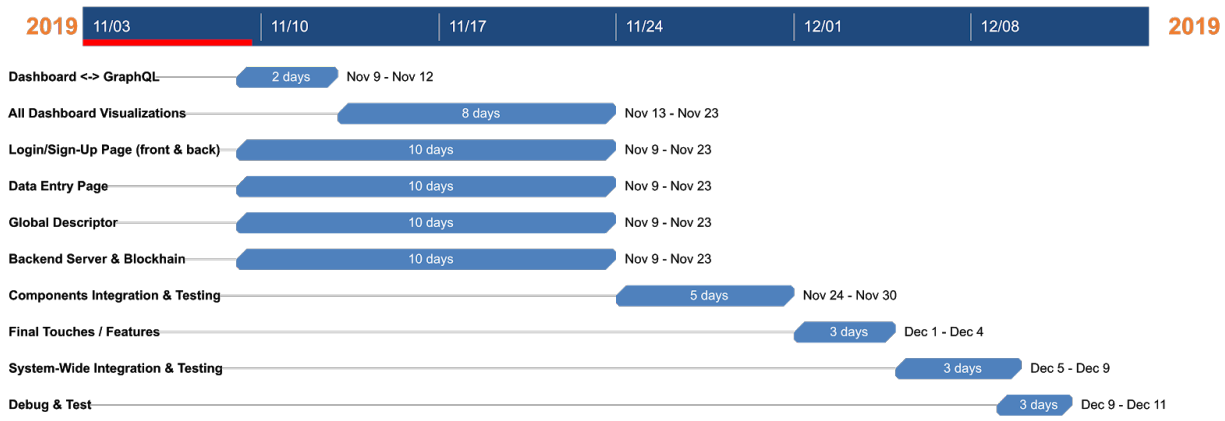
Our integration testing has a top-down approach. We will test the big concepts and modules of our code before moving on to the smaller modules. These big concepts, which are the GraphQL API, global descriptors, and user descriptors, take priority for our group.

12.d Plans for More Tests

We plan on testing more requirements or use cases such as user authentication, public access of health data, and data administration

13. History of Work, Current Status, and Future Work

Report 2 Plan of Work:



13.a History of work

We decided to include our commit history for both of our two repositories that shown concretely our sequence of work and timelines. While perhaps come of these commits seem incoherent, they do paint a good picture of the things that we were working on. We usually stayed on track, specifically completing the authentication, the form fill out page, and significant work on the backend.

For this iteration, we were able to add the global descriptor service which added data from the frontend, through the backend, to finally the blockchain. In addition to this, we added a write through cache that helps us return values quickly to have a responsive UI, especially for global descriptors.

We added integration with metamask to augment our authentication flow in parallel which took two members of the group to complete and approximately the time since the first demo to this one.

For the UI we added the data entry form, and one new chart that compares BPM measurements. The majority of our time was spent connecting the dashboard via graphql. We assumed this would take the least amount of time, but the complexity of our UI and understanding the concepts of GraphQL took our members the longest to implement.

Here is the complete history of commits of our group:

author	**time**	**commits**
Eric Rivera	Sun Dec 8	Suggestions API and Weight Suggestions Card (#47) * Added webpage suggestions * Added Weight Suggestions Card
Khalid Akash	Sun Dec 8	Data entry (#48) * Implemented form, need to add subscription * Setup subscription client * Completed data entry form implementation * Fixed linting errors
Smeet97Kathiria	Fri Dec 6	Line Graph Update (#44) * Visual LG Changes * Add graphql to Linegraph * lg changes * Minor changes to LG * minor fix
Eric Rivera	Fri Dec 6	Added BPM Pie chart (#45) * Added BPM Pie chart * Changed aesthetics to be a bit more flat
Eric Rivera	Thu Dec 5	Merge pull request #43 from AkashWorld/eric-QLconnect QL Reconnect & update error handling
Khalid Akash	Tue Dec 3	Actions test (#36) Create new actions script

SuvaShahria	Sat Nov 2	GlobalDescriptors to Mock Resolvers (#33) Adds a few of the global queries in the backend to the mock resolvers so we can test easily.
Khalid Akash	Fri Nov 1	Added BMI, Weight change, Insertion Subscription to mock server (#28)
Khalid Akash	Wed Nov 6	GraphQL Mock server (#27)
Khalid Akash	Tue Oct 2	Changed router to hash router, deploy script
Eric Rivera	Tue Oct 2	Added save button to charts, in english (#26)
Smeet97Kathiria	Tue Oct 2	Visual LG Changes (#25) Add range indicator for line graph
Eric Rivera	Mon Oct 2	Add Weight Area Chart (#19) * Add Weight Area Chart * Changed width of area line graph
Khalid Akash	Mon Oct 2	Line graph (#17) Added line graph to user-analytics page.
Khalid Akash	Mon Oct 2	Added three new cards, changed color scheme to yellow-pink
Eric Rivera	Sun Oct 2	Added Weight Bar Chart (#14) * Added Weight Bar Chart
Netanel Arussy	Thu Oct 2	Create data-entry.jsx (#12) * Create data-entry.jsx * Removed comments, added page to router * Adjusted setState to include previous state + new state change
Khalid Akash	Thu Oct 2	Akash/map demo (#13) * Map working, working on background * Fixed page orientation * Finished map graph, touched up card component and initiated user descriptor page
Khalid Akash	Tue Oct 2	Eslint adjust (#10) Updated eslint to be more strict.
Khalid Akash	Thu Oct 1	Card implemented (#6)
Khalid Akash	Fri Oct 1	Added router (#4)
Khalid Akash	Sun Oct 6	Merge pull request #2 from AkashWorld/project-setup Project setup
Eric Rivera	Fri Oct 4	Merge pull request #1 from AkashWorld/create-react-branch Create react branch

Eric	Fri Oct 4	Merge branch 'master' of https://github.com/thisisericc/Health-blocks
root	Thu Dec 5	Merge branch 'LoginAdjusted' of https://github.com/AkashWorld/blockchain-frontend into LoginAdjusted
Khalid Akash	Tue Dec 3	Formatted, changed link to direct client (will need to change back to link later)
root	Mon Nov 1	Commit has some of the login mechanisims; there things that still need to be worked out though
Khalid Akash	Tue Dec 3	Merge branch 'master' of https://github.com/AkashWorld/blockchain-frontend
Khalid Akash	Mon Nov 1	Added getBalance query
Netanel Arussy	Wed Dec 4	Delete data-entry.jsx I want to just upload an entirely new one. When it tried to merge earlier it failed.
Netanel Arussy	Wed Dec 4	Ok I figured it out
Khalid Akash	Sun Dec 8	Fixed linting errors
Khalid Akash	Sun Dec 8	Completed data entry form implementation
Khalid Akash	Sat Dec 7	Setup subscription client
Khalid Akash	Sat Dec 7	Implemented form, need to add subscription
John Doe	Tue Nov 1	Connect Weight Range to GraphQL
hs697	Sat Oct 1	Add files via upload HTML and JS. Draft.
hs697	Mon Oct 2	Add files via upload Progress so far. I still need to import the 'create new account' and 'login' parts into App.js file. Also, I need to add handling errors/exceptions
John Doe	Sat Dec 7	Added Weight Suggestions Card
Khalid Akash	Fri Dec 6	Added webpage suggestions
Khalid Akash	Fri Dec 6	Cleaned out unnecessary files (#34)
Khalid Akash	Fri Dec 6	Adjusted unixTimestamp to float, should change to specific scalar (#32)
Khalid Akash	Thu Dec 5	Trend (#31) * Finished cached implementation of average calculation * Updated type chain command to include double quotes

Khalid Akash	Thu Dec 5	Fixed dependency cycle in script (#29)
Khalid Akash	Thu Dec 5	Implemented web crawler search for GraphQL (#28)
SuvaShahria	Wed Nov 2	Graphql2 (#26) * test * update * pushing so i can pull when everything breaks * insertValueGlobal not recognized * bugfix * working * query's done * insertValue now adds to global * some fixes * update * update * insertvalue * Removed unnecessary subscription
RizThe1andOnly	Wed Nov 2	Login (#27) Adds authentication flow to backend via metamask.
Khalid Akash	Fri Nov 1	Changed insert Values return a transactionHash, and implemented a subscription to check transaction results (#25)
Khalid Akash	Wed Nov 1	Added weight and bmi trend query (#24)
SuvaShahria	Tue Nov 1	Global descriptor (#23) Added global descriptors smart contract.
Khalid Akash	Tue Nov 5	Added support for GraphQL subscriptions - server side (#22)
Khalid Akash	Sun Oct 2	Format fix (#16) Added formatter pre-commit. Strong linting against vars.
Khalid Akash	Tue Oct 2	Migrations update (#15)
Khalid Akash	Wed Oct 1	[Initial] Implementation of GraphQL to User Descriptor Smart Contract (#14)
Khalid Akash	Mon Oct 1	GraphQL API (#13) * Implemented GraphQL server https://www.youtube.com/watch?v=HCv8z7X7Q9s&feature=youtu.be
Khalid Akash	Wed Oct 9	Merge pull request #11 from AkashWorld/RizExpressSetupBranch Riz express setup branch
Rizwan Chowdhury	Wed Oct 9	Merge branch 'RizExpressSetupBranch' of https://github.com/AkashWorld/Smart-Contract-Service into RizExpressSetupBranch

Khalid Akash	Tue Oct 8	Added @types/express to fix type error, changed the name of server initialization file to server.ts from index.ts, changed main.ts to index.ts, fixed Router compilation issue by calling the Router() constructor
Khalid Akash	Fri Oct 4	Merge pull request #10 from AkashWorld/akash/testing-infra Added unit, integration, and coverage tests, git hook.
Khalid Akash	Wed Oct 2	Added unit, integration, and coverage tests. Added a git hook so that when committing, all tests and syntax errors are caught first. Added github actions so we can continually test when pushing a branch to github
Khalid Akash	Wed Sep 2	Merge pull request #3 from AkashWorld/akash/fix-build Fixed build process by rearranging the order of type generation
Khalid Akash	Wed Sep 2	Merge pull request #2 from AkashWorld/docs Added docs and updated readme
Khalid Akash	Tue Sep 2	Merge pull request #1 from AkashWorld/akash/contributions Updated contributions README
root	Fri Dec 6	Merge branch 'login-iter-2' of https://github.com/AkashWorld/smart-contract-backend into login-iter-2
root	Thu Dec 5	Merge branch 'loginadjustments' of https://github.com/AkashWorld/smart-contract-backend into loginadjustments
root	Thu Dec 5	Merge branch 'master' of https://github.com/AkashWorld/smart-contract-backend into loginadjustments
root	Sat Nov 2	Merge branch 'master' of https://github.com/AkashWorld/smart-contract-backend into Login
root	Sat Nov 2	Merge branch 'Login' of https://github.com/AkashWorld/smart-contract-backend into Login
root	Tue Oct 2	helloMerge branch 'master' of https://github.com/AkashWorld/Smart-Contract-Service into Login
Rizwan Chowdhury	Sun Oct 2	Merge branch 'Login' of https://github.com/AkashWorld/Smart-Contract-Service into Login
Rizwan Chowdhury	Sun Oct 2	Merge branch 'master' of https://github.com/AkashWorld/Smart-Contract-Service

13.b Future Works

Future works for this project would be really understanding the limitations of using blockchain for a performance application. This means proliferating the write-through cache throughout the backend. Along with storing data in the blockchain, we can store data into a relational/NO-SQL database that are less sensitive.

14. References

Ethereum. “Ethereum/Wiki.” *GitHub*, github.com/ethereum/wiki/wiki/White-Paper.

Truffle Suite. “Truffle: Overview: Documentation.” *Truffle Suite*, www.trufflesuite.com/docs/truffle/overview.

“web3.Js - Ethereum JavaScript API.” *web3.Js - Ethereum JavaScript API - web3.Js 1.0.0 Documentation*, web3js.readthedocs.io/en/v1.2.0/.

“The U.S. Government and Global Non-Communicable Disease Efforts”, www.kff.org/global-health-policy/fact-sheet/the-u-s-government-and-global-non-communicable-diseases/.

WHO“Global report on diabetes”, www.who.int/diabetes/global-report.

“UML Reference”: https://www.uml-diagrams.org/class-reference.html

“RSA Encrpytion”: http://mathworld.wolfram.com/RSAEncryption.html

“The Node.js Event Loop, Timers, and process.nextTick()”: https://nodejs.org/de/docs/guides/event-loop-timers-and-nexttick/

“JSON-RPC 2.0 Specification“: https://www.jsonrpc.org/specification

“An overview of HTTP”: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview