RUTGERS UNIVERSITY

SOFTWARE ENGINEERING 1

Project: Blockchain-Based Health Monitoring System

Website: healthcolate.herokuapp.com

Github: https://github.com/PopDescSystm

Group 2: Pratik Mistry, Shounak Rangwala, Amod Deo, Pranit Ghag, Sen Zhang, Pranav Shivkumar, Swapnil Kamate, Vikhyat Dhamija, Lizhe Wei

Table of Contents

1. Customer Problem Statement

a. Problem Statement

The world is beginning to realize the importance of health in a common man's daily life and the influence it has on his activities down the line. While work and its related aspects are a critical part of our lives, health plays an even bigger role than most people imagine. In order to perform everyday activities efficiently, we need to have a good and healthy body. The scope of "exercise" is expanding and encroaching into other activities, like dance (zumba, aerobics and so on). Some kinds of exercise, like jogging, hiking, swimming, trekking and physical fitness are becoming hugely popular around the globe.

However, expensive health check-ups and inadequate facilities remain an obstacle to most people who wish to know the condition of their body and the ways to improve on it.The health condition of a person can be easily obtained by checking the vital signs of him. Crucial inferences can be drawn by analyzing the blood pressure stats, blood sugar levels, sleeping patterns and so on. Most people are aware of the consequences but always relied on medical help, which led them to incur monetary expenses.

We propose a digital ledger of sorts for the various users on the internet. This ledger contains data entered by the user as well as data streamed from the user's devices like Apple Watch, Fitbit etc. This data contains many important metrics such as the following:

- Age
- Location
- Weight, height and BMI (Body Mass Index)
- Blood pressure
- Heart rate
- Sleep patterns
- Cholesterol levels

Our aim is to create a fully operational user-interface that would be friendly to use and understand and you can enter these details onto the website and the ledger will be directly populated with these details. You can also give the application permission to access data, like your heart rate, blood pressure, blood sugar etc. from your Apple Watch, Fitbit etc., which will be collected at regular intervals from your device and registered into the ledger.

One of the critical needs of such an application is that any user should know how healthy or unhealthy he is as compared to others. He must be able to select different parameters (like sleep patterns, blood sugar levels, cholesterols etc.) and based on those parameters compare his data to the average/ general health of the other users. He can also select data for comparison from a specific location because geographically the health conditions considered normal can vary and also from a particular window of time, since people's health can vary in a location from time to time (eg. before and after the holiday season).

Also this application provides for the user to make his data available to or kept private from other users when those other users would be performing similar data analysis operations for their own purposes. However the application should stress on the fact that all users must try to make the data available to other users because that would be helpful to others in figuring out their health requirements and that their data would be anonymously shared.

Now, people would be naturally apprehensive to allow their personal details to be shared with this application, due to the concern of getting hacked. To address this concern, we are using the **Blockchain** technology, which provides several advantages over a regular database:

- The data in each block in the blockchain will be encrypted to create its hash which is like the block's fingerprint (a unique hash for each block). The data also contains the *previous* blocks hash in it, so if any attacker wishes to alter or change any blocks data, then he would have had to change the hashes of all the blocks following that block.
- Each node (i.e user) will be part of this network and will have an entire copy of the blockchain with them. Any change in the blockchain will be an addition of a block to the existing to the blockchain and the hash generated by the new block will be verified by all the nodes in the network. Only then can the block be added to the blockchain and this increases the security of the data.
- Since all the nodes have a copy of the entire blockchain we do not have to worry about maintaining a centralized server (a "honey-pot" if you may) which may be prone to hacking attacks or maybe at risk of losing all the data if it shuts down or encounters a hardware issue. Any node can leave the network and on re-entry it can be updated with the latest version of the blockchain from the other active nodes.

Now, you may be wondering why would I want to go through all this trouble? Well, because we are implementing a blockchain in this solution, all the personal details of the users on the network are securely stored in the blockchain. This solution does not have a centralized authority, like a server or cloud, that stores all the data; the data is shared among the various users in the network and can be compared among them.

Since the blockchain is nearly immutable, the existing blocks cannot be changed. All the information stored in the system will be the latest and most reliable information, that will be regulated at the point of entry to the system. It will be like a community helping out its members by sharing their health parameters. The information will be much more recent and relevant as compared to the similar data found in government databases.

Also, the government-provided statistics may be incomplete or, at times, outdated. Since the user manually enters his/her details into the blockchain, with the help of user interface, the data is up to date.

The interactive application must also provide for the users to communicate with each other using their usernames through a forum which will be included in the application. The application will also be able to provide you daily links to health-related, diet-related articles and videos.

So, with all these facts and details, we can say with some aplomb that our proposed approach is highly innovative.

b. Glossary of Terms

1. ***BMI (Body Mass Index)*** : BMI of a person is the weight of the person (in kgs.) divided by the person's height squared (in meters)
2. ***Sleep Patterns*** : Classifying people in different categories (like Healthy larks, sleep savvy seniors etc.) based on the average hours of sleep at night.
3. ***Cholesterol level*** : Record Low density lipoprotein (LDL) cholesterol along with the total cholesterol level so that they can control their dietary habits.
4. ***Blood Sugar*** : Compare the blood sugar of a person with the blood sugar chart so that they can self manage any diseases like diabetes. Can intake values both fasting and 1-2 hours after eating.
5. ***Blood Pressure*** : Compare the blood pressure input from the user and compare with the normal upper and lower bounds to judge whether it is normal or not
6. ***Ledger*** : A book or collection for recording the personal details of the users on the network.
7. ***Blockchain*** : A collection of blocks interconnected with each other; each block contains the details that are given by each user of the network.
8. ***Hash :*** Also called as the *hashtag function*, this is a function that is used to encrypt the blocks so that attackers cannot access the information in the blocks.
9. ***Node*** : Each user on the network is referred to as a node.

2. System of Requirements

a. Enumerated Functional Requirements

Identifier	Priority Weight	Requirements
REQ -1	10	System should have secure login
REQ -2	8	System should allow to update the parameters
REQ -3	7	System should integrate with health device
REQ -4	10	System should store data in secured manner
REQ -5	9	System should compare data with authentic data
REQ -6	7	System should restrict outlier data
REQ -7	5	System should notify the user if a new parameter is introduced
REQ -8	6	System should recommend health measures analyzing my data
REQ -9	4	System should allow users to give feedback and provide support as well
REQ -10	6	System should generate historical report for users to view/access
REQ -11	10	System should allow user to unregister and delete data upon unregistering
REQ -12	9	System should ask the user for consent to share data for statistical comparison

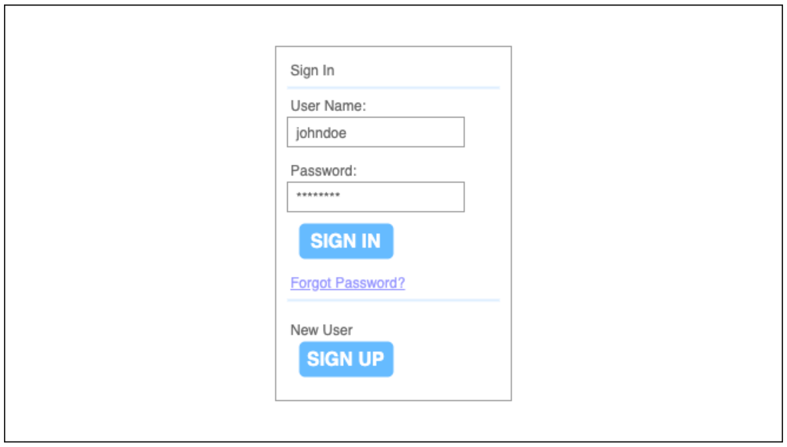
REQ -13	10	System should allow user to view/compare average statistical data of other users
REQ-14	3	System should allow user to communicate with other users in the system

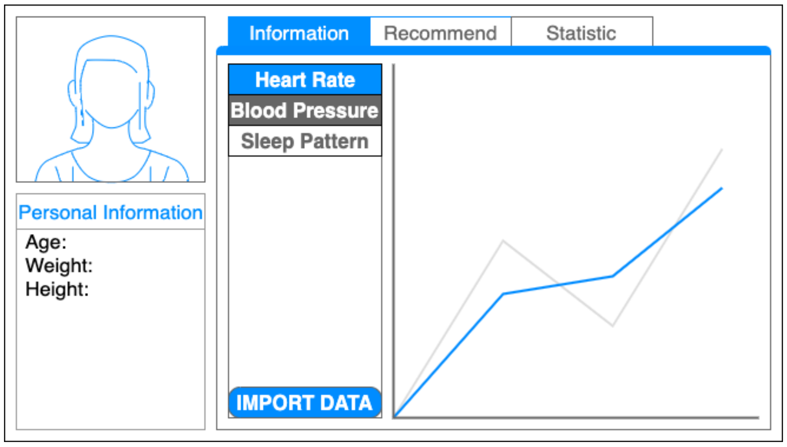
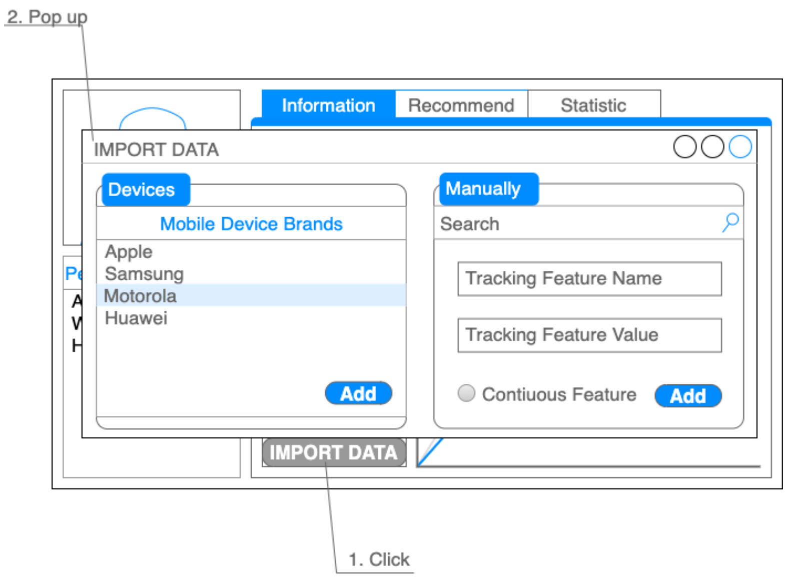
b. Enumerated Non-Functional Requirements

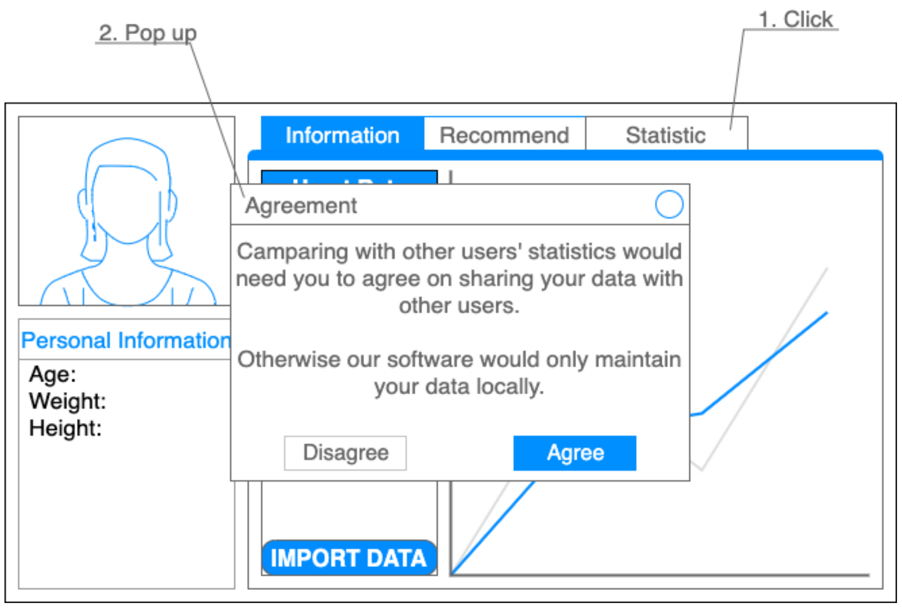
Identifier	Priority Weight	Requirements
REQ -15	8	As a system, size and generality of the data must be defined
REQ -16	10	As a system, all the user data must be encrypted
REQ -17	7	As a system, communication between system actors must be secured
REQ -18	5	As a system, application/portal must be supported in different browsers of both web and mobile platforms
REQ -19	2	System should be scalable and load balanced
REQ -20	4	As a system, data of the entire system must be archived periodically
REQ -21	6	As a system, backups of the data must be taken periodically
REQ -22	7	As a system, appropriate business continuity policies and disaster recovery strategy must be implemented
REQ -23	3	As a system, system maintenance should be done regularly in order to keep systems up to date

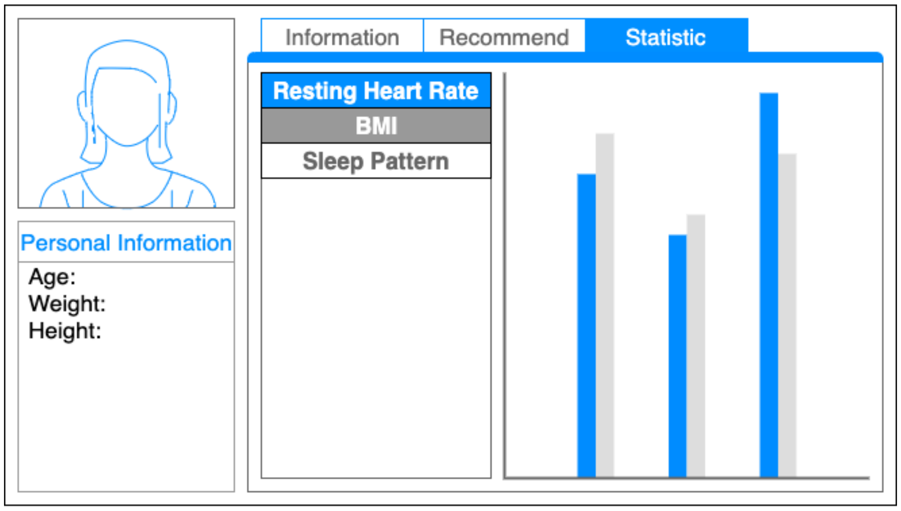
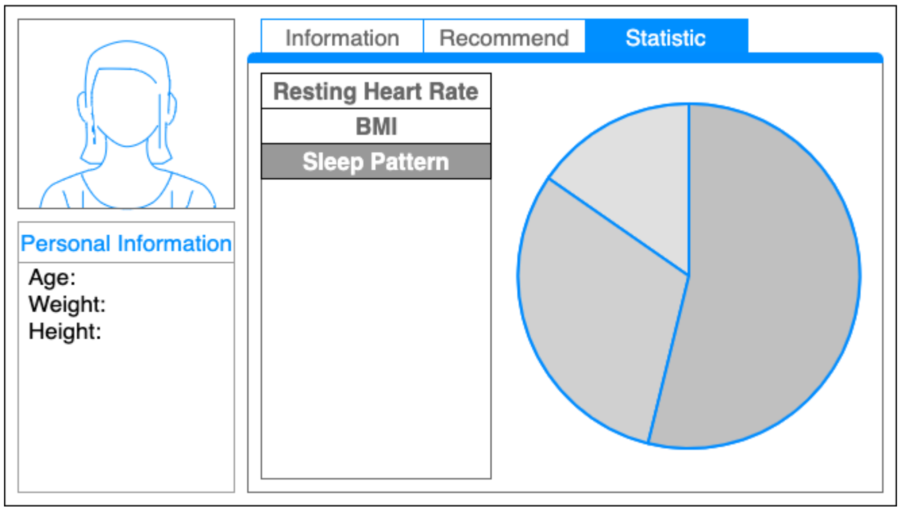
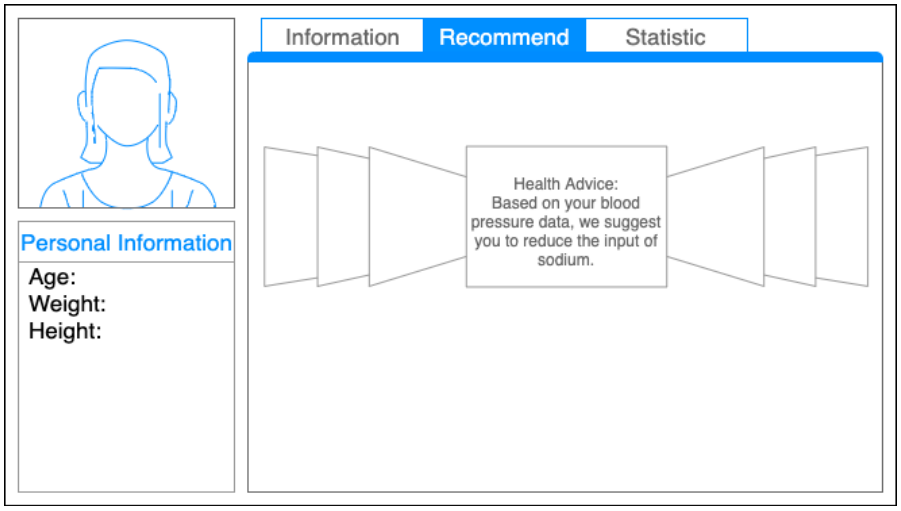
REQ -24	8	As a system, user requests/issues should be supported and addressed
REQ -25	8	As a system, deployment strategy should be implemented along with roll back strategy in case of deployment failure
REQ-26	10	As a system, data across all the nodes in the system must be synchronized
REQ-27	4	As a system, logging and monitoring of the system and application must be in place
REQ-28	9	User manual and Architecture Diagram along with proper Documentation of the system must be provided

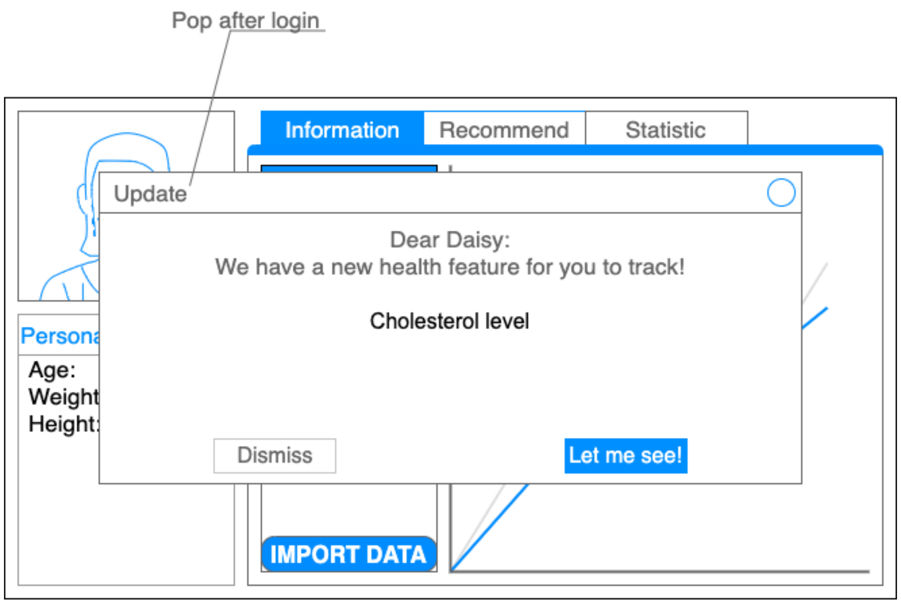
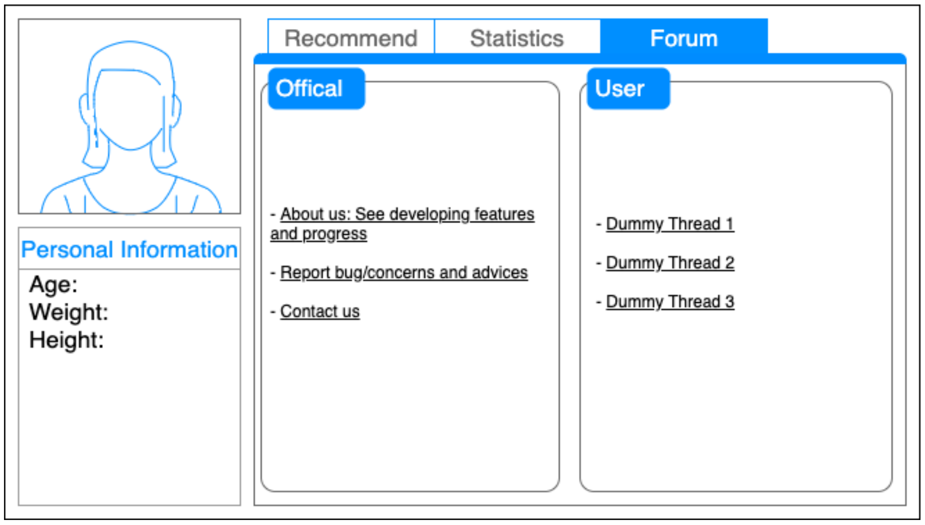
c. User Interface Requirements

Identifier	**Priority Weight**	**Requirements**
REQ-29	10	GUI must have a landing page(register and log in).  Sign In User Name: johndoe Password: ******** SIGN IN Forgot Password? New User SIGN UP
REQ-30a,b	10	GUI must have a main page of users' personal information. GUI must have a page for user to view his historical data.

		 Information Recommend Statistic Heart Rate Blood Pressure Sleep Pattern IMPORT DATA Personal Information Age: Weight: Height:
REQ-31	10	GUI must have a page to enter users' health data.  2. Pop up Information Recommend Statistic IMPORT DATA Devices Mobile Device Brands Apple Samsung Motorola Huawei Add Manually Search Tracking Feature Name Tracking Feature Value Contiuous Feature Add IMPORT DATA 1. Click
REQ-32	10	GUI must have consent option for sharing data for comparison.

		 2. Pop up 1. Click Information Recommend Statistic Personal Information Age: Weight: Height: Agreement Disagree Agree IMPORT DATA Camparing with other users' statistics would need you to agree on sharing your data with other users. Otherwise our software would only maintain your data locally.
REQ-33	8	GUI must have a page for statistical data for comparison.

		 Information Recommend Statistic Resting Heart Rate BMI Sleep Pattern Personal Information Age: Weight: Height:  Information Recommend Statistic Resting Heart Rate BMI Sleep Pattern Personal Information Age: Weight: Height:
REQ-34	4	GUI must have recommendation page on users' health data.  Information Recommend Statistic Personal Information Age: Weight: Height: Health Advice: Based on your blood pressure data, we suggest you to reduce the input of sodium.

REQ-35	4	GUI must have a page to notify users the new update.  Pop after login Information Recommend Statistic Update Dear Daisy: We have a new health feature for you to track! Cholesterol level Dismiss Let me see! Persona Age: Weight Height: IMPORT DATA
REQ-36a b,c	2	GUI must have a forum page for users' communication. GUI must have support page for any issues. GUI must have a page about us for our progress on development.  Recommend Statistics Forum Offical User - About us: See developing features and progress - Report bug/concerns and advices - Contact us - Dummy Thread 1 - Dummy Thread 2 - Dummy Thread 3 Personal Information Age: Weight: Height:

3. Functional Requirement Specifications

a. Actors and Goals

Actor	Actor's Goal	Use Case Name
Visitor/User	To login to the web portal/application to enter health data, view statistics, get health recommendations, view comparison reports, etc.	Login (UC-1)
Visitor/User	To register in the system/application via web portal	Register (UC-2)
Visitor/User	To add data in the system/application via web portal	Add Data (UC-3)
Visitor/User	To view the historical health data for analysis	Historical Data Presentation (UC-5)
Visitor/User	To compare health data with other registered and active users	Comparison Data Presentation (UC-6)
Visitor/User	To get health recommendations based on the data entered in the system	Health Recommendations (UC-7)
Visitor/User	To communicate with other registered and active users and also with system administrator/support team	Communication (UC-8)
Visitor/User	To have its data validated before adding data in the system	Data Validation (UC-4)
System Admin	To resolve issues that users might face while user performs any operations or has any other issues	Communication (UC-8)
Database/Repository	To store the user information, login details, data, recommendations, historical and comparison data	UC1, UC2, UC3, UC4, UC5, UC6, UC7, UC8

b. Use Cases

i. Casual Description

UC#1 Logging In

The user can enter the portal/application by entering his credentials used for registration. He can then perform various operations, like entering his data into the application, viewing the statistics, getting health recommendations, viewing the comparison reports and so on.

UC#2 New Registration

The user can register into the blockchain-based network to be a part of the community. Then he or she can start with the service provided by us and all the other users. After the user finished registration, he or she can log in to the whole system and do all the operations listed in UC#1. Also, he can unregister from our system, and his data will be cleared from the blockchain-based network.

UC#3 User Data Addition

The user, once registered into the system, will be able to integrate his or her health device with the system and securely store the health data without manual intervention. If needed the user can also manually update the parameters of their health data

UC#4 Validation of Data

The data validator will check if the data added by the user into the web portal/application is within the appropriate range. It will allow the user to proceed with adding or updating his data if it is valid; if any outliers exist in the data entered, an error message will pop up prompting the user to re-enter his valid data.

UC#5 Historical Data Representation

Users can request our system to generate a historical data representation of their health information. The graphical view of their cumulative health data will allow users to better perceive the changes in their health parameters. Users can monitor and take preventive measures with this information.

UC#6 Comparison of Data Report

Users can request our system to generate the histograms, pie charts and scatter plots to show his position in the whole population depends on his or her requested parameter. For example, the user could see how the other users are distributed on age in a pie chart, or where is his or her position to the whole users in a histogram. The UI would also allow the user to customize multiple parameters in the comparison. Last but not least, before a user joining the pool of user comparison, he or she must agree to share his or her data to the whole network as a base of comparison. The user can choose to exit the pool with his data removed from the cloud whenever he or she wants.

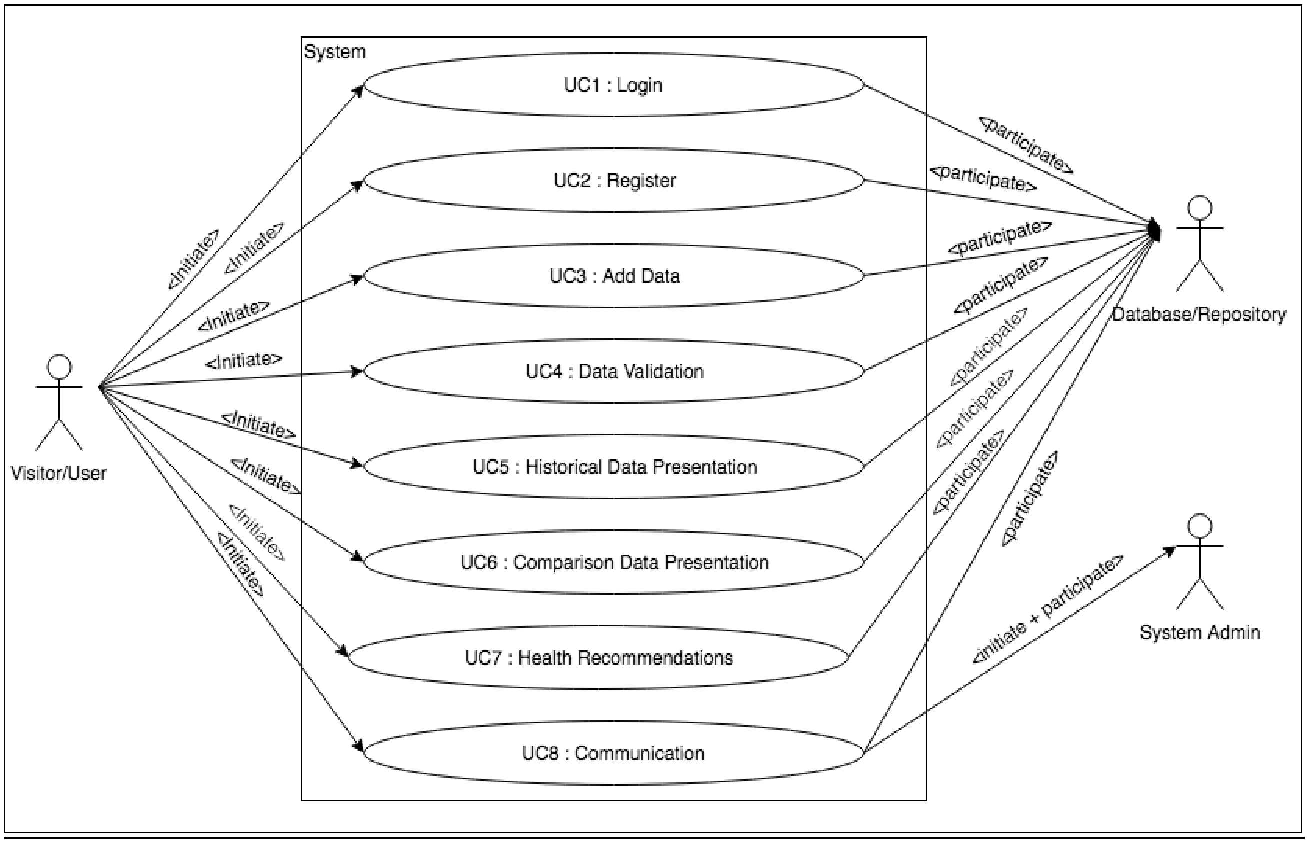
UC#7 Health Recommendation

Our system can analyze the user's health data and suggesting certain health recommendations to appropriate users. These are general health recommendations which will include exercises, YouTube video recommendations etcetera to give users a chance to normalize their health parameters.

UC#8 User Communication

Our system can provide a forum to let users communicate with each other and give us, the development team, feedback and report bugs.

ii. Use Case Diagram



iii. Traceability Matrix

The below table depicts the mapping of the various requirements of our system with the use cases defined previously. The requirements are given based on a scale from 1 to 10, 1 being the lowest priority and 10 being the highest priority.

REQ'T	**PW**	**UC1**	**UC2**	**UC3**	**UC4**	**UC5**	**UC6**	**UC7**	**UC8**
REQ1	10	X	X						
REQ2	8			X					
REQ3	7			X					
REQ4	10			X					
REQ5	9				X				
REQ6	7			X					
REQ7	5			X					
REQ8	6							X	
REQ9	4								X
REQ10	6					X			
REQ11	10		X						
REQ12	9						X		
REQ13	10						X		
REQ14	3								X
Maximum Weight		10	10	10	9	6	10	6	4
Total Weights		10	20	37	9	6	19	6	7

iv. Fully Dressed Description

Use Case UC#2: ***New Registration***

Related Requirements: REQ-1,REQ-11
Initiating Actor: Users
Actor's Goal: To allow user to have secure login.
To allow user to unregister and delete the data.

Participating Actors: Active users,System Admin,System

Precondition: System should be active and running.
User should have established connection with the system.
User should meet the requirement for registering.

Postcondition: User can enter his data in the system.
User can update or delete his data as per his requirement.
User will have option to share/hide his data from other users.

Failed end condition: User entering wrong credentials, re-enter valid credentials
Unable to correct to system,check system status

Flow of events for success scenario:

1. → **User** enters his credentials into the portal
2. .← The credentials are verified by the system.
3. .→ The **user** is allowed to access the portal.

Flow of events for extensions:
User enters invalid/out-of-bounds data.

1. ← **System** detects the error and sends an error message to the user prompting to enter the data again.

User enters a weak password while registering.

1. ← **System** should suggest a strong password to the user in order to keep the data more secure.

Use Case UC#3: ***User Data Addition***

Related Requirements: REQ-2, REQ-3, REQ-4, REQ-6, REQ-7

Initiating Actor: Users

Actor's Goal: To add and update data which is within appropriate range.
To append encrypted data into the blockchain with
Verification from all users.
To inform the user if any new parameter is available in the
System for monitoring.

Participating Actors: User fitness device
System, other Active Users, System admin

Precondition: User should be active on the system
User's communication link with the system should be online
The blockchain must be initialised and active.

Postcondition: New encrypted data blocks should be appended to the
blockchain and updated across all the users present in the
network.

Failed end condition: Data input unsuccessful, check data range
Unable to connect with system, check connection

Flow of events for success scenario:

1. → **User** enters the web application and chooses option to input data.
2. → **User** gives permission to the system to access the **fitness device.**
3. → **User** inputs data.
4. **System** uses the data input by the **user** to make a new block
5. include:: *ValidationOfData (UC-4)*
6. **← System** adds the new validated block to the blockchain and gives user success message

Flow of events for extensions:

3a. **User** inputs invalid/out-of-bounds data

1. **← System** detects error and sends an error message to the **user** asking to re-input the data.
2. **→ User** inputs correct data

5a. Failure to validate new block from more than 50% active **Users.**

1. **← System** a) detects error, b) marks failed attempt, c) signals to **System Admin**
2. **← System Admin** will try to detect issue (maybe problem with hashing function)
3. **→ User** data gets validated.

Use Case UC#6: ***Comparison of Data Report***

Related Requirements: REQ-12, REQ-13

Initiating Actor: Users

Actor's Goal: To provide users the option to choose to share their data for statistical comparison.
To generate the relevant histograms, pie charts, scatter plots so that the user can compare his data with other users

Participating Actors: Active users, Other Users, System

Precondition: User should be active on the system.

User must have the links available for selecting or deselecting the information sharing option and to view their comparison report.

System must have the active users who have selected the option of sharing their data for comparisons.

User must have actively participated in the block chain system based on which the user must have maintained the statistical data block chains for other users.

Postcondition: User must be able to be or not be a part of the comparison subsystem of our system and can accordingly view the comparison report while having the comparison of its own data with the statistical information maintained based on other users' data.

Failed end condition: Not able to view the comparison data properly (maybe only his own data without any comparison or with partial comparison) ,User unable to connect to comparison page (Error message page displayed)

Flow of events for success scenario:

1. → User can select/deselect the option of being part of the comparison sub system.

2. ← System can provide the user with the services accordingly.

3. → User click the link for accessing the comparison report page.

4. ← System provides the user with the comparison report.

5. → User can view the comparison report comparing his/her data with the statistical information derived out of other users' data.

Flow of events for extensions:

I. When users select the option for doing some necessary selection to be the part of comparison of subsystem and some abnormality happens.

1. User can then raise the issue with the system administrator.

II. User click the link and error message is displayed or improper comparison report is visible.

1. User can raise the issue with the system administrator. The user can check its connection and troubleshoot system from its end.

c. System Sequence Diagrams

i. Use Case - 2

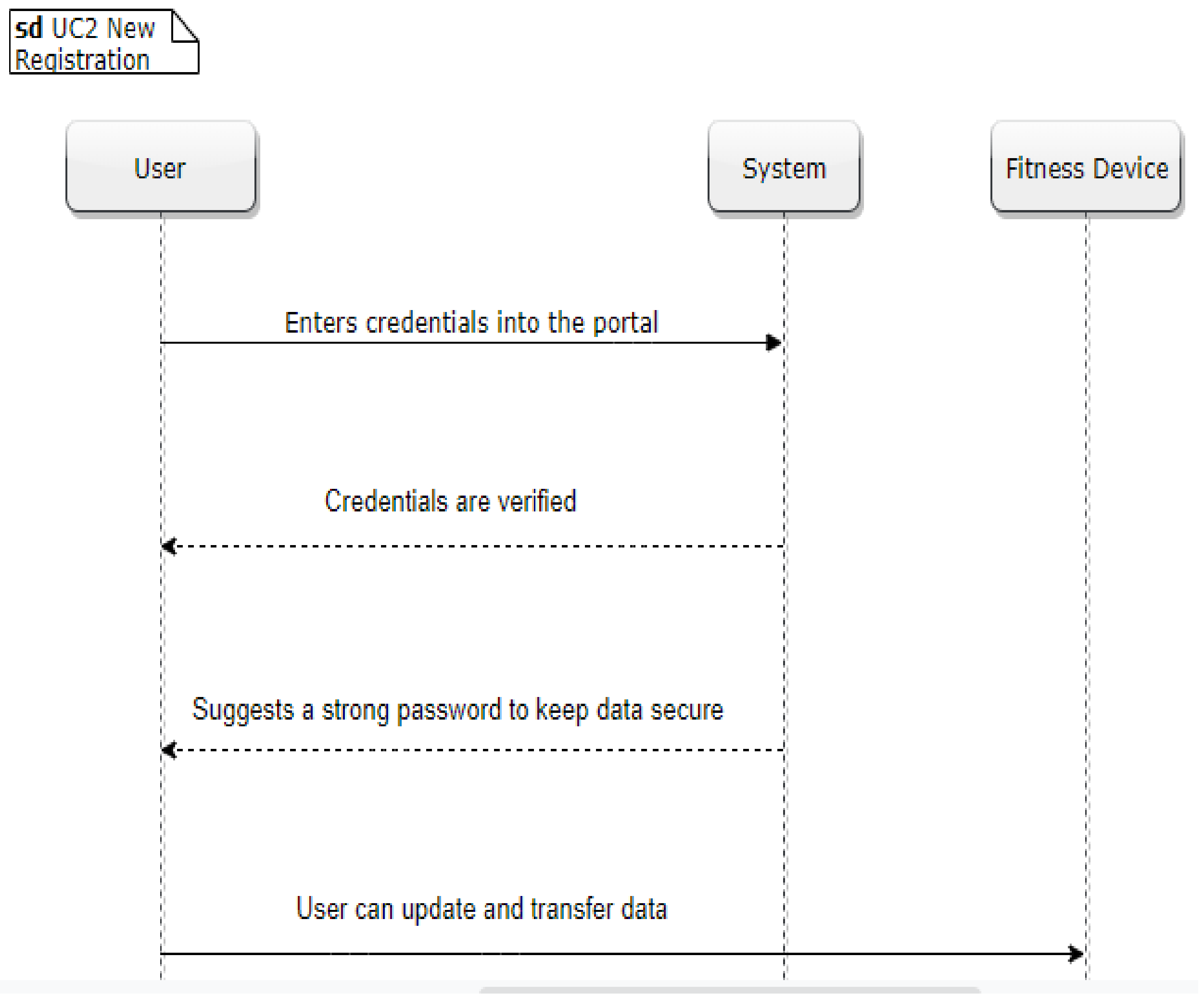
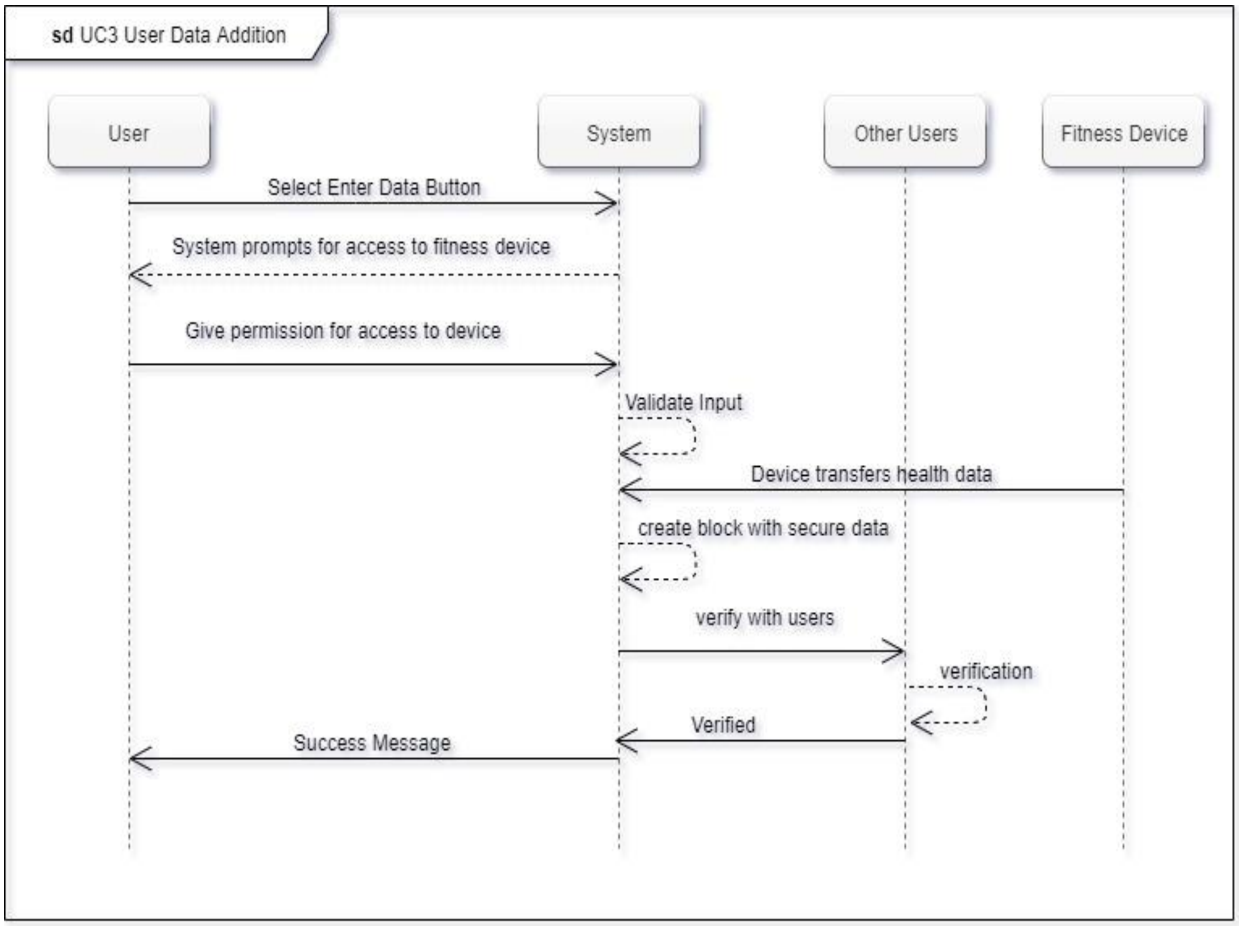

sd UC2 New Registration
User
System
Fitness Device
Enters credentials into the portal
Credentials are verified
Suggests a strong password to keep data secure
User can update and transfer data

Fig 1: The above figure depicts the sequence diagram for **Use Case 2**, which is “New Registration”. The user will enter his credentials into the web portal which will be verified by the system. Once the credentials are verified, the user has to enter a unique username and password, which will serve as his login. The system should suggest a strong password to the user to keep the user’s data secure. Once he successfully enters this, he will be allowed access into the system, where he can view his data, get health recommendations, view comparison reports, among other operations.

ii. Use Case - 3


sd UC3 User Data Addition
User
System
Other Users
Fitness Device
Select Enter Data Button
System prompts for access to fitness device
Give permission for access to device
Validate Input
Device transfers health data
create block with secure data
verify with users
verification
Verified
Success Message

<u>Fig 2-a</u> shows the sequence diagram of **Use Case 3** which is “User Data Addition”. The user will select the option to enter the data to the blockchain. The System then prompts the User to input the data and give permission to access data from the fitness device. The user then grants the system access to the fitness device and also manually inputs the data into the system. The System verifies if none of the data is an outlier. It then creates a block that will be added to the blockchain. The block is then sent to be verified by all the other users in the network and if it is verified by more than 50% of the users, it is added to the blockchain and a success message is then sent to the user.

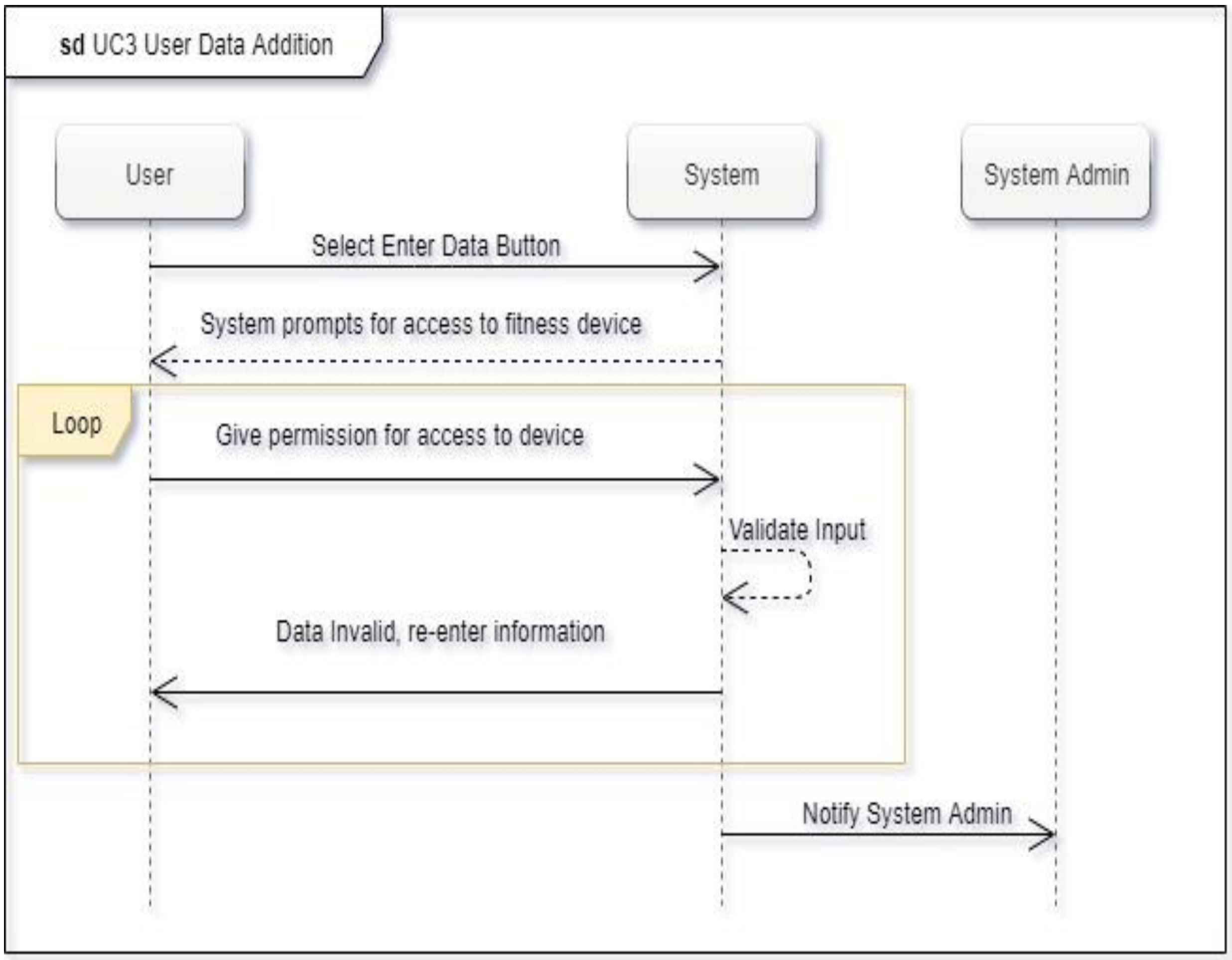

sd UC3 User Data Addition
User
System
System Admin
Select Enter Data Button
System prompts for access to fitness device
Loop
Give permission for access to device
Validate Input
Data Invalid, re-enter information
Notify System Admin

Fig 2-b shows the error scenario when the user entered data fails validation checks performed by the system. Out of range data or outlier data entry could be the root cause of this issue. In this scenario the system shows an error message to the user and requests the user to re-enter appropriate information. Also, the system admin is notified of this situation such that it can monitor the further transactions of the user with the system.

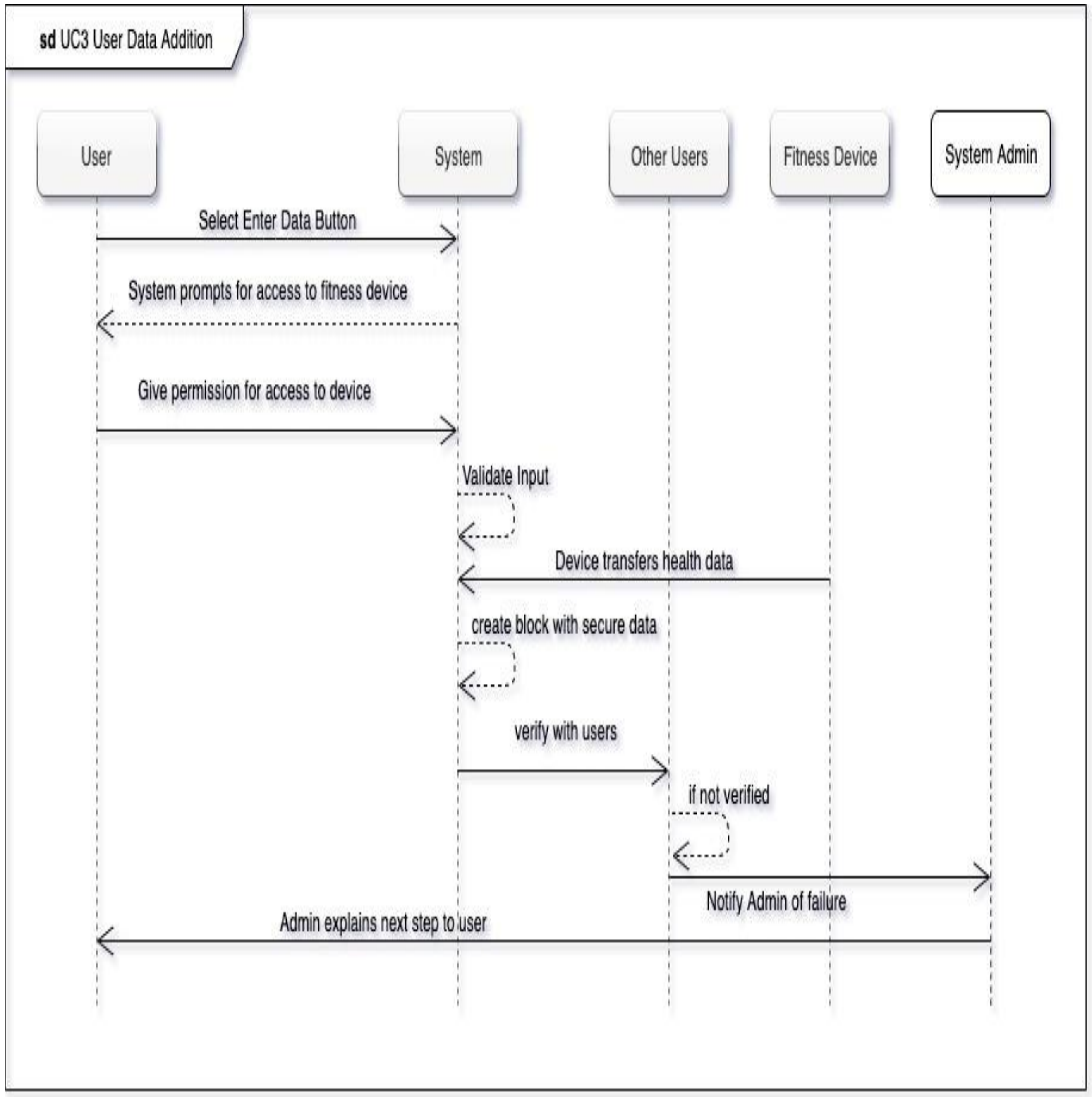

sd UC3 User Data Addition
User
System
Other Users
Fitness Device
System Admin
Select Enter Data Button
System prompts for access to fitness device
Give permission for access to device
Validate Input
Device transfers health data
create block with secure data
verify with users
if not verified
Notify Admin of failure
Admin explains next step to user

Fig 2-c shows the error scenario when the block created by the system is not validated by more than 50% of the users of the blockchain. This is a major issue which may be because of a problem in the hashing function. The system notifies the system admin who will inspect why the error occurred and then will tell the user what steps have to be taken next (maybe reenter the data etc.)

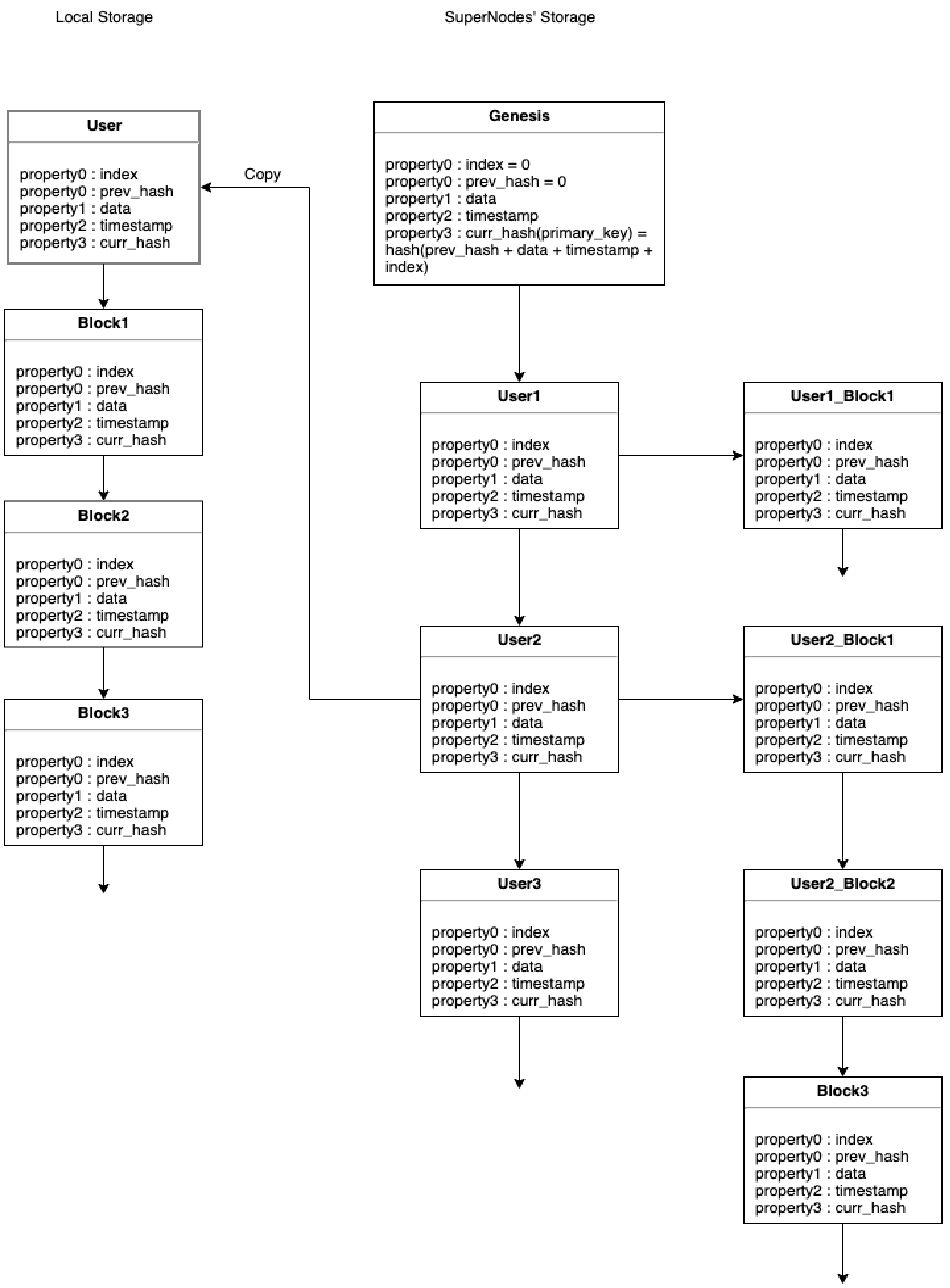

Local Storage
SuperNodes' Storage
User
property0 : index
property0 : prev_hash
property1 : data
property2 : timestamp
property3 : curr_hash
Copy
Block1
Block2
Block3
Genesis
property0 : index = 0
property0 : prev_hash = 0
property1 : data
property2 : timestamp
property3 : curr_hash(primary_key) = hash(prev_hash + data + timestamp + index)
User1
User1_Block1
User2
User2_Block1
User3
User2_Block2
Block3

Fig 2-d shows structure of our Block-chain based storage. We store users' data both locally and remotely. On users' local app, we store and generate data blocks; on the system consist of super nodes and other users, they store the whole data blocks and verify newly adding blocks. If we have a total of m users, each user have n data blocks on average, we can reduce the complexity from m multiply n to m plus n. Also, we can process multiple users' verification without conflict.

iii. Use Case - 6

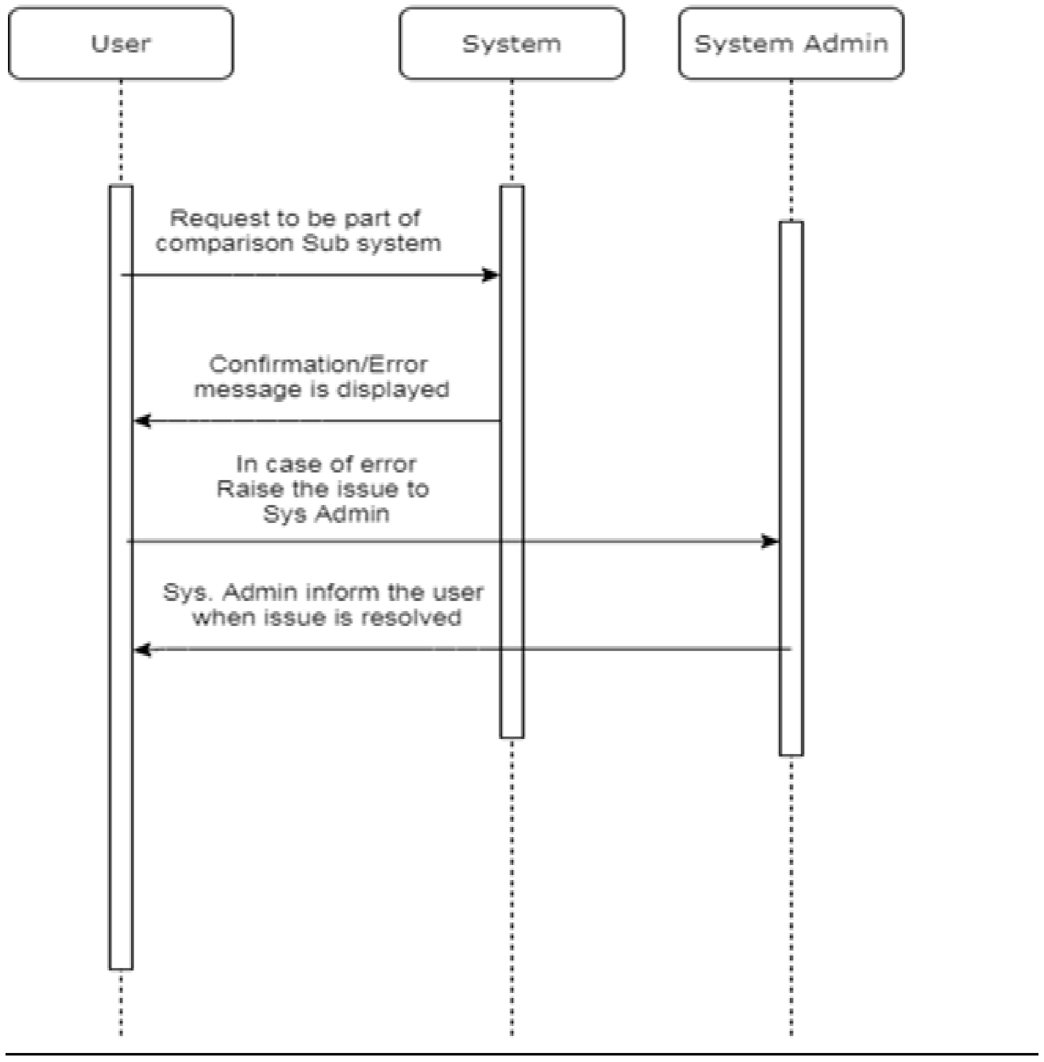

User
System
System Admin
Request to be part of comparison Sub system
Confirmation/Error message is displayed
In case of error Raise the issue to Sys Admin
Sys. Admin inform the user when issue is resolved

Fig. 3-a depicts **Use Case (UC) 6**, where the user has the option to be a part of the comparison subsystem. In case some issue occurs, or the confirmation of selection is not shown to the user, he/she can raise the issue with the system administrator and after resolving the issue the system administrator will revert to the user.

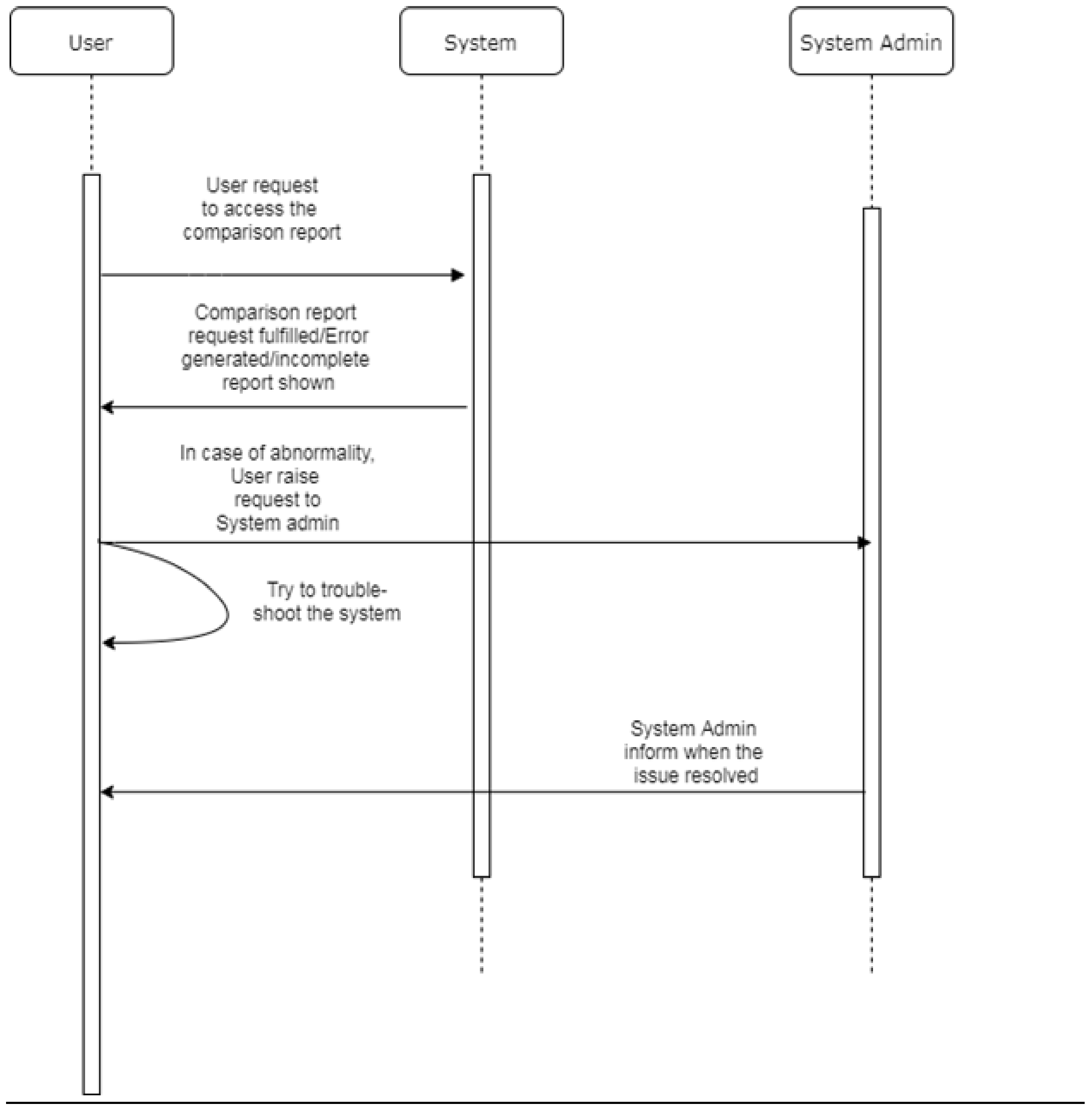
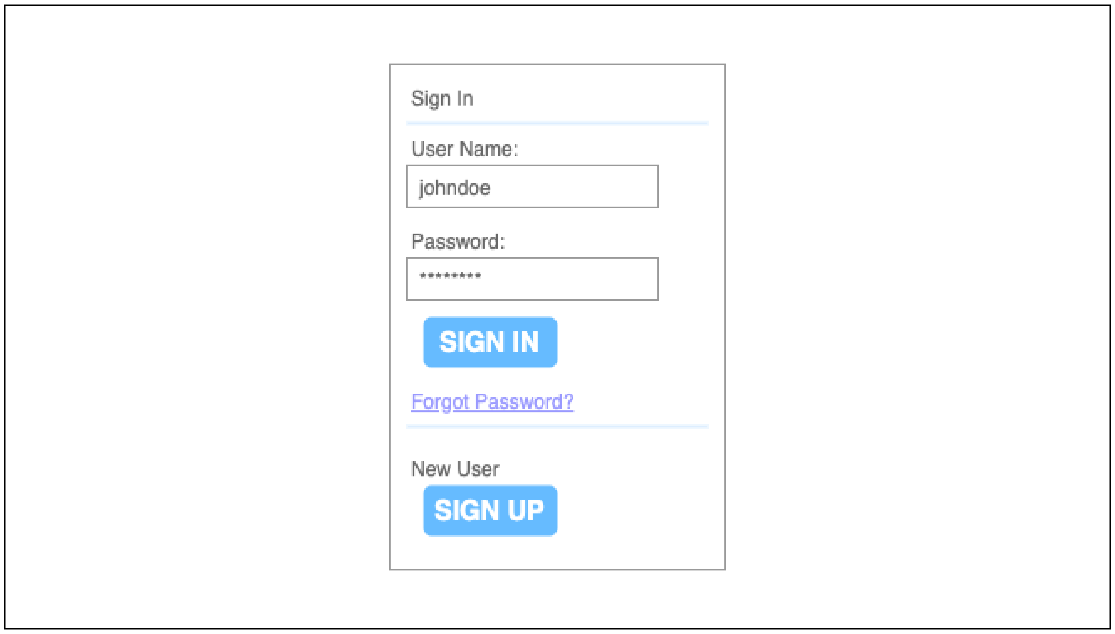

User
System
System Admin
User request to access the comparison report
Comparison report request fulfilled/Error generated/incomplete report shown
In case of abnormality, User raise request to System admin
Try to trouble-shoot the system
System Admin inform when the issue resolved

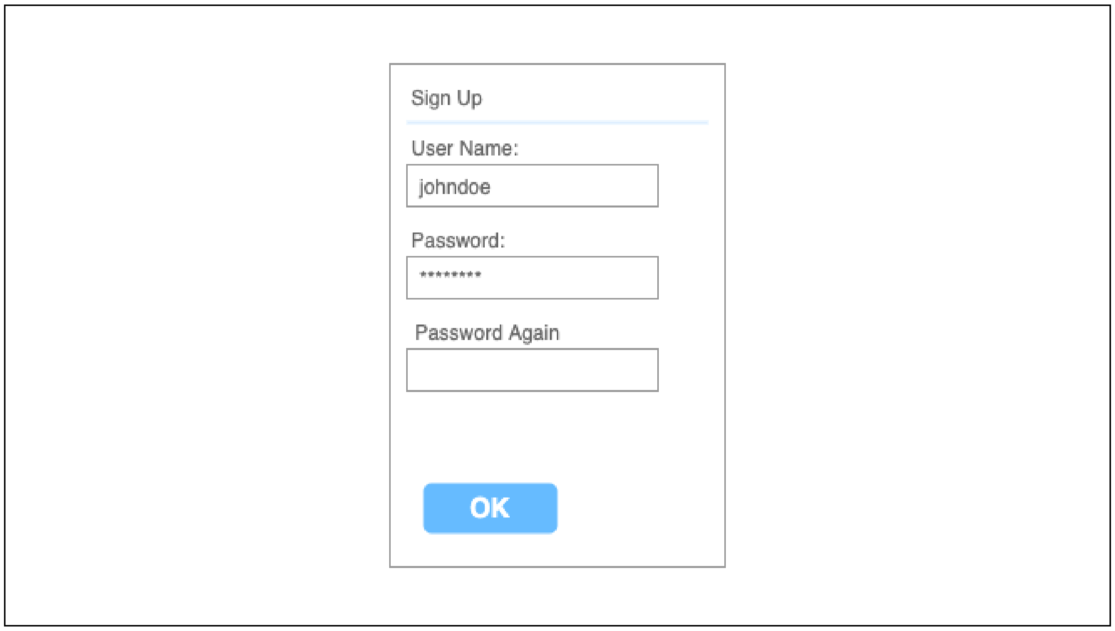
Fig. 3-b Here, the user tries to access the comparison report and in case any issue/abnormality crops up, he/she can raise the issue with the System administrator who can try to troubleshoot the system from its end. After resolution of the issue, the system administrator will revert to the user with the appropriate status message.

4. User Interface Specification

a. Use Case Effort Estimation

i. Registration:





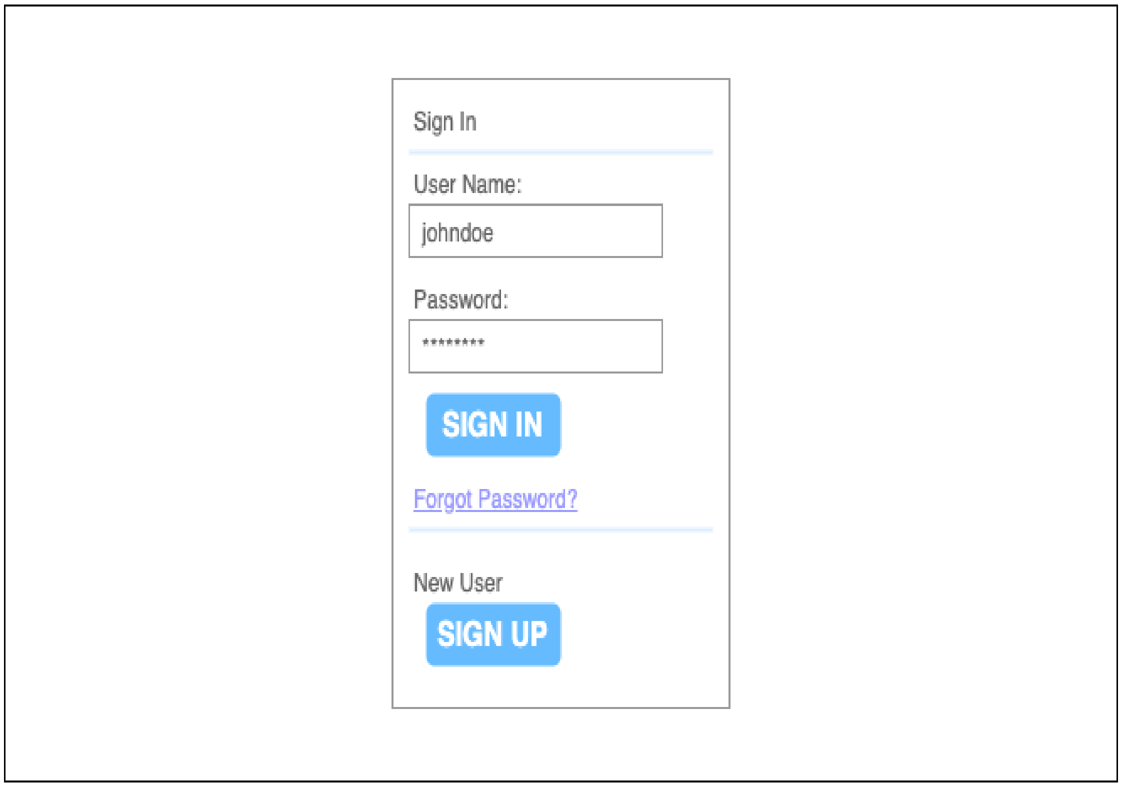
Navigation: total 2 mouse clicks, as follows

a. Click “SIGN UP” button
b. Finish data entry in the new page
c. Click “OK” to finish registration

Data Entry: total 3 keystrokes, as follows

a. Press keys to input username
b. Press keys to input password
c. Press keys to input password again

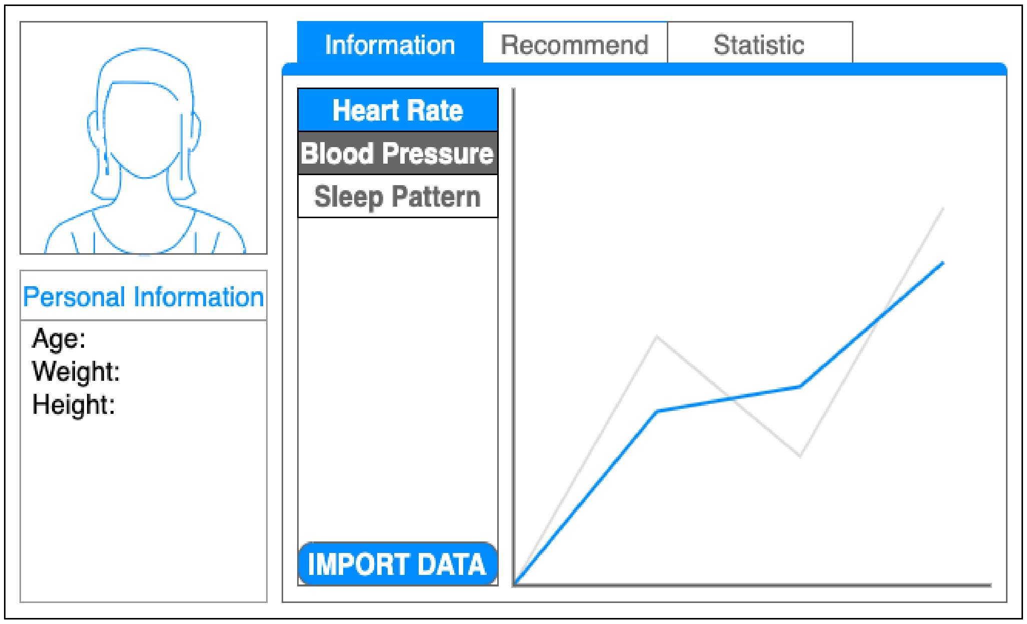
ii. Log In:



Navigation: total 1 mouse clicks, as follows
a. Finish data entry in the new page
b. Click “OK” to finish registration

Data Entry: total 2 keystrokes, as follows
a. Press keys to input username
b. Press keys to input password

iii. Personal Information



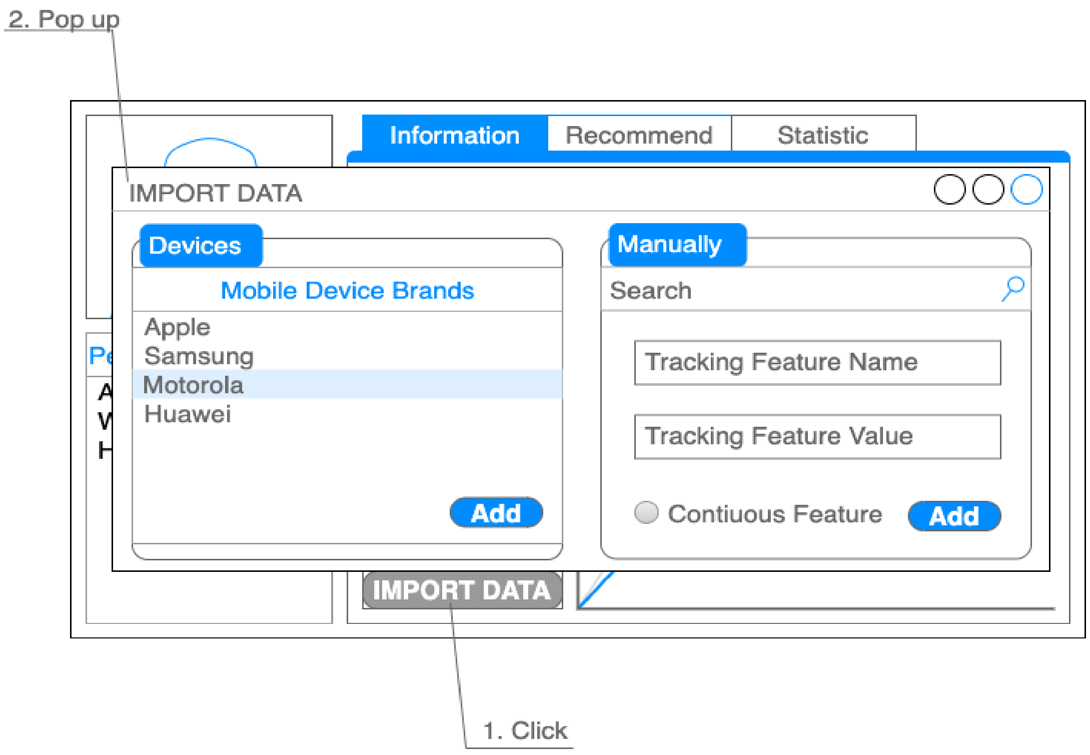
Navigation: total 2 mouse clicks, as follows

a. Click “Information” page

c. Click on whatever parameter that the user wants to see

d. The static parameter will always be showed on the left

iv. Data Import



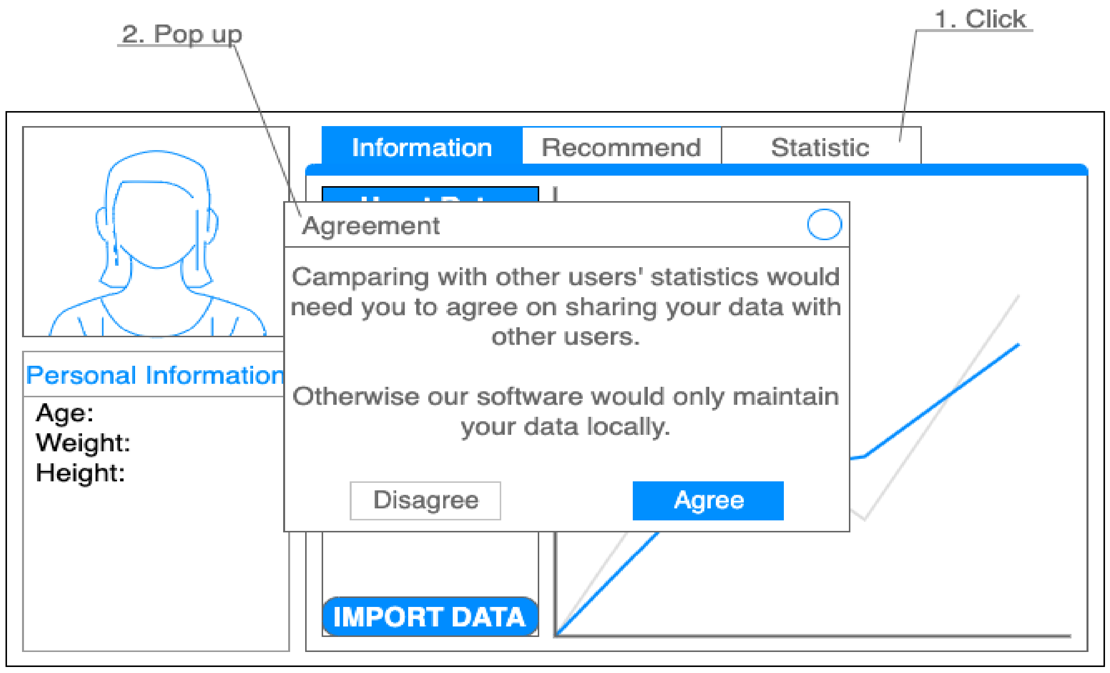
Navigation: total 2 mouse clicks, as follows

a. Click ”IMPORT DATA” button
b. Finish data entry in the new page
c. If the user has a health device, then select the right device to go further
d. Click “Add” to finish data input

Data Entry: total 2 keystrokes and 1 mouse click, as follows

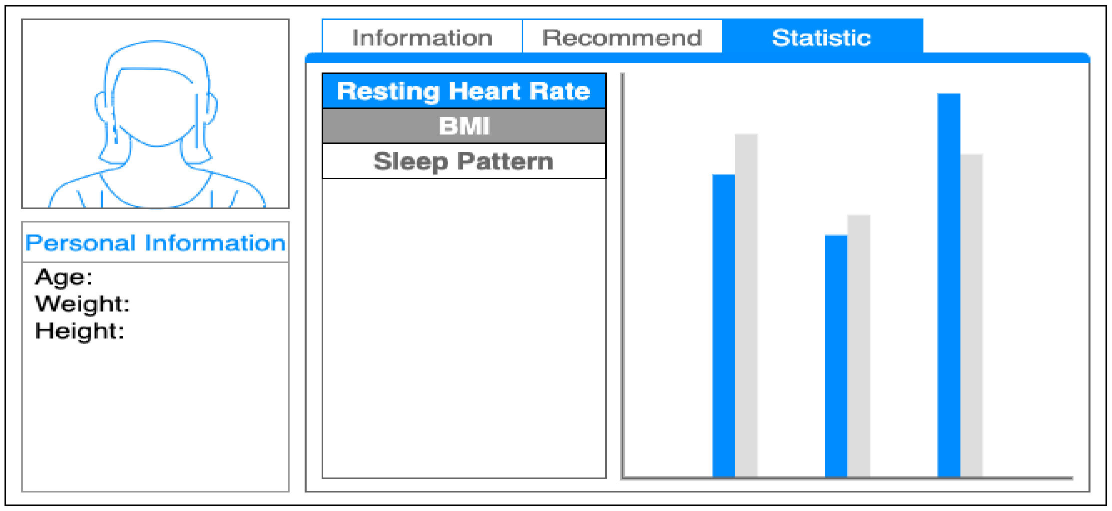
a. Press the keys to input the parameter name
b. Press the value of that parameter
c. Click the continuous feature box if the parameter has continuous value

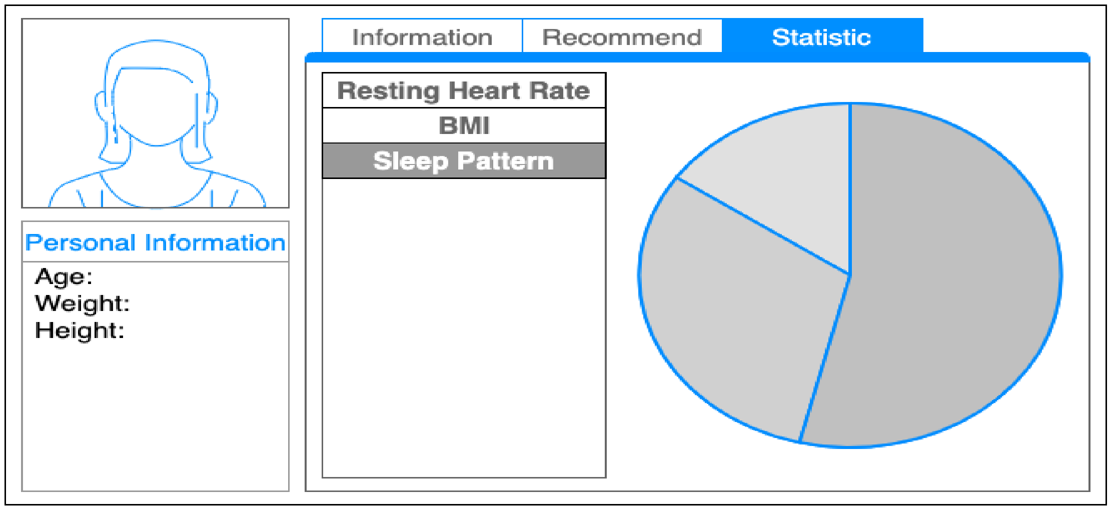
v. Statistics



Camparing with other users' statistics would need you to agree on sharing your data with other users.

Otherwise our software would only maintain your data locally.

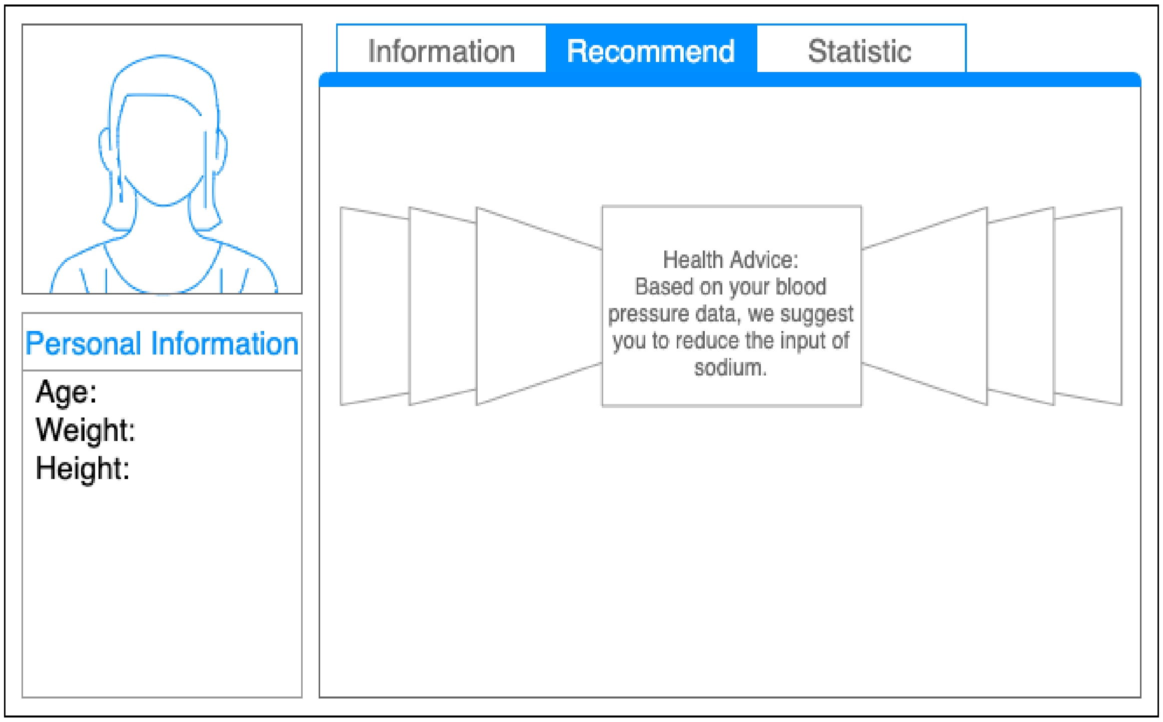




Navigation: total 3 mouse clicks, as follows

a. Click ”Statistic” page
b. Finish agreement in the new page
c. Click on the parameters that the user wants

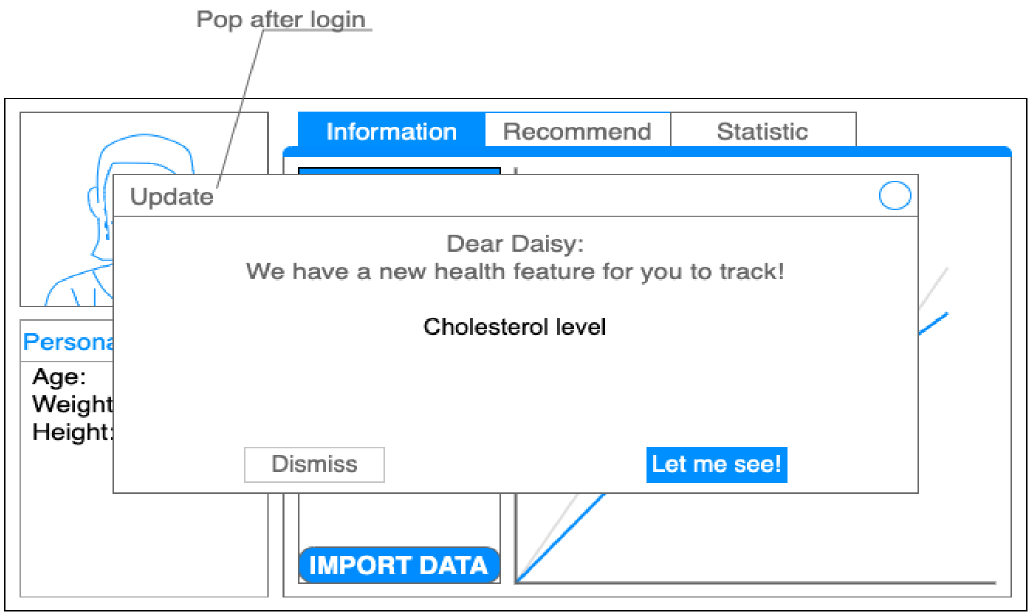
vi. Recommendation



Navigation: total 1 mouse clicks, as follows

a. Click “Recommend” page
b. The recommendations will show on this page

vii. New Update



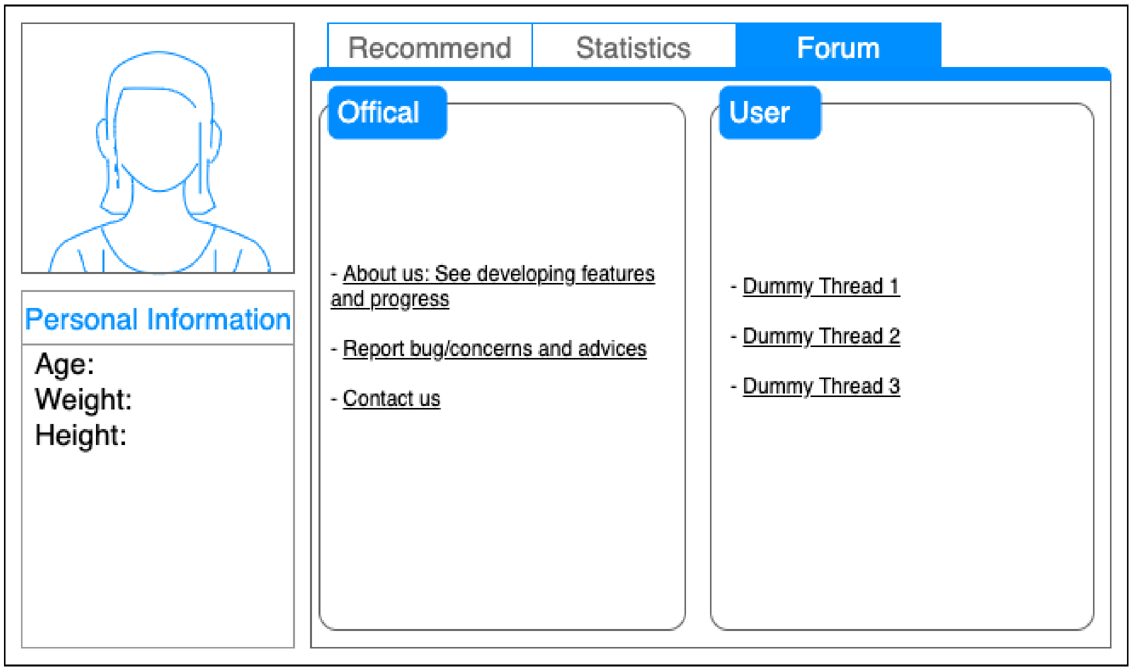
Dear Daisy:
We have a new health feature for you to track!

Cholesterol level

Navigation: total 1 mouse clicks, as follows

a. The new parameter update page will pop up after the user logged in
b. Click “Dismiss” or “Let me see!” to move further

viii. Forum



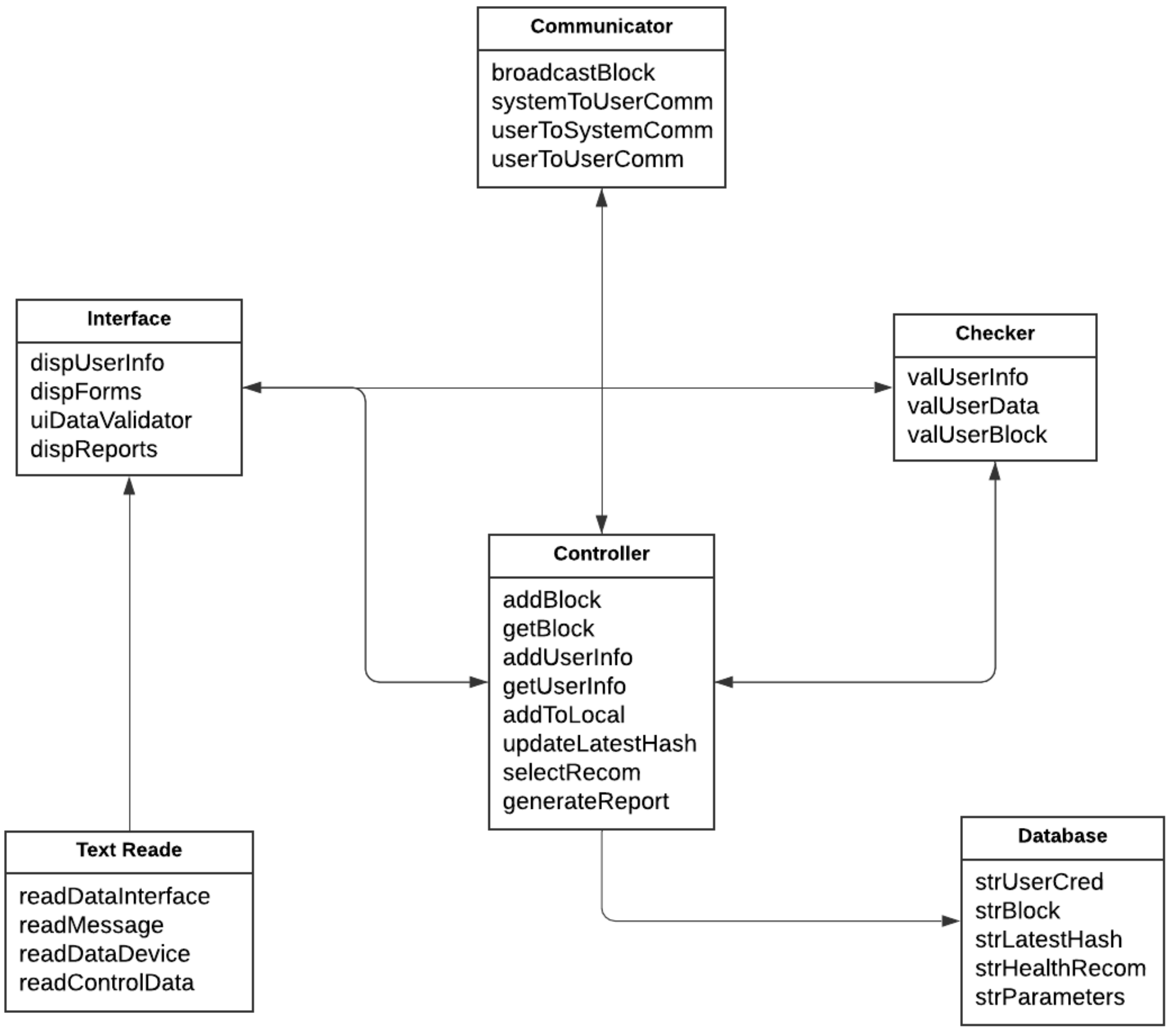
Navigation: total 1 mouse clicks, as follows

4. Click “Forum” page
5. The forum will show on this page.

5. Domain Analysis

a. Domain Model

i. Concept Definitions



Responsibility	Type	Concept
R1: Display data and user information	D	Interface
R2: Display forms for user interaction	D	Interface
R3: Validation of input data before submission of web forms	D	Interface
R4: Display statistical and comparison report and recommendations	D	Interface
R5: Validate user entered user id and password	D	Checker
R6: Validate data entered by user (validate outliers)	D	Checker
R7: Validate new block from all the users	D	Checker
R8: Read health data, credentials and personal information of the user	D	TextReader
R9: Read messages sent by user to support team/other users	D	TextReader
R10: Read streaming data from the device (E.g. Apple Watch)	D	TextReader
R11: Read customized parameters for report generation	D	TextReader
R12: Broadcast hashed block and new parameter to all users	C	Communicator
R13: Send messages, recommendations, personal information, historical and comparison data to users	C	Communicator
R14: Send messages to system for block validations and feedback/requests	C	Communicator
R15: Send messages to other users	D	Communicator
R16: Add block to the database	D	Controller
R17: Get block from the database	D	Controller
R18: Add user information or get user information	D	Controller
R19: Get user information	D	Controller
R20: Add block to local blockchain	D	Controller
R21: Update latest hash in blockchain	D	Controller
R22: Select recommendation to user user	D	Controller
R23: Generate comparison report or historical report	D	Controller

R24: Store user credentials	K	Database
R25: Store blockchain data	K	Database
R26: Store latest hash for each user	K	Database
R27: Store health recommendations	K	Database
R28: Store existing parameters	K	Database

ii. Association Definitions

Concept Pair	Association Description	Association Name
Controller → Interface	Controller gets request from the interface and sends the result for display	Send result
Interface → Controller	Reads user and device data input and messages	Read Input
Interface → Checker	Sends data to checker for verification (userID, data)	Check Info
Checker <-> Controller	Checker sends validation result to controller	Validation message
Communicator <-> Interface	Send and receive messages between users and system	Get message Send message
Controller <-> Communicator	Controller responds to requests from communicator	Get request Send response
Controller → BlockChain DB	Controller send appropriate data to DB. Controller acquires data from DB.	Generate requests Append data

iii. Attribute Definitions

Responsibility	**Attribute**	**Concept**
R1: Display data , user information	dispUserInfo	Interface
R2: Display forms for user interaction	dispForms	Interface
R3: Validation of input data before submission of web forms	uiDataValidator	Interface
R4: Display statistical and comparison report and recommendations	dispReports	Interface
R5: Validate user entered user id and password	valUserInfo	Checker
R6: Validate data entered by user (validate outliers)	valUserData	Checker
R7: Validate new block from all the users	valUserBlock	Checker
R8: Read health data, credentials and personal information of the user	readDataInterface	Text Reader
R9: Read messages sent by user to support team/other users	readMessage	Text Reader
R10: Read streaming data from the device (E.g. Apple Watch)	readDataDevice	Text Reader
R11: Read control parameters for report generation	readControlData	Text Reader
R12: Broadcast hashed block and new parameter to all users	broadcastBlock	Communicator
R13: Send messages, recommendations, personal information, historical and comparison data to users	systemToUserComm	Communicator
R14: Send messages to system for block validations and feedback/requests	userToSystemComm	Communicator
R15: Send messages to other users	userToUserComm	Communicator
R16: Add block to the database	addBlock	Controller
R17: Get block from the database	getBlock	Controller

R18: Add user information	addUserInfo	Controller
R19: Get user information	getUserInfo	Controller
R20: Add block to local blockchain	addToLocal	Controller
R21: Update latest hash in blockchain	updateLatestHash	Controller
R22: Select recommendation to user user	selectRecom	Controller
R23: Generate comparison report or historical report	generateReport	Controller
R24: Store user credentials	strUserCred	Database
R25: Store blockchain data	strBlock	Database
R26: Store latest hash for each user	strLatestHash	Database
R27: Store health recommendations	strHealthRecom	Database
R28: Store existing parameters	strParameters	Database

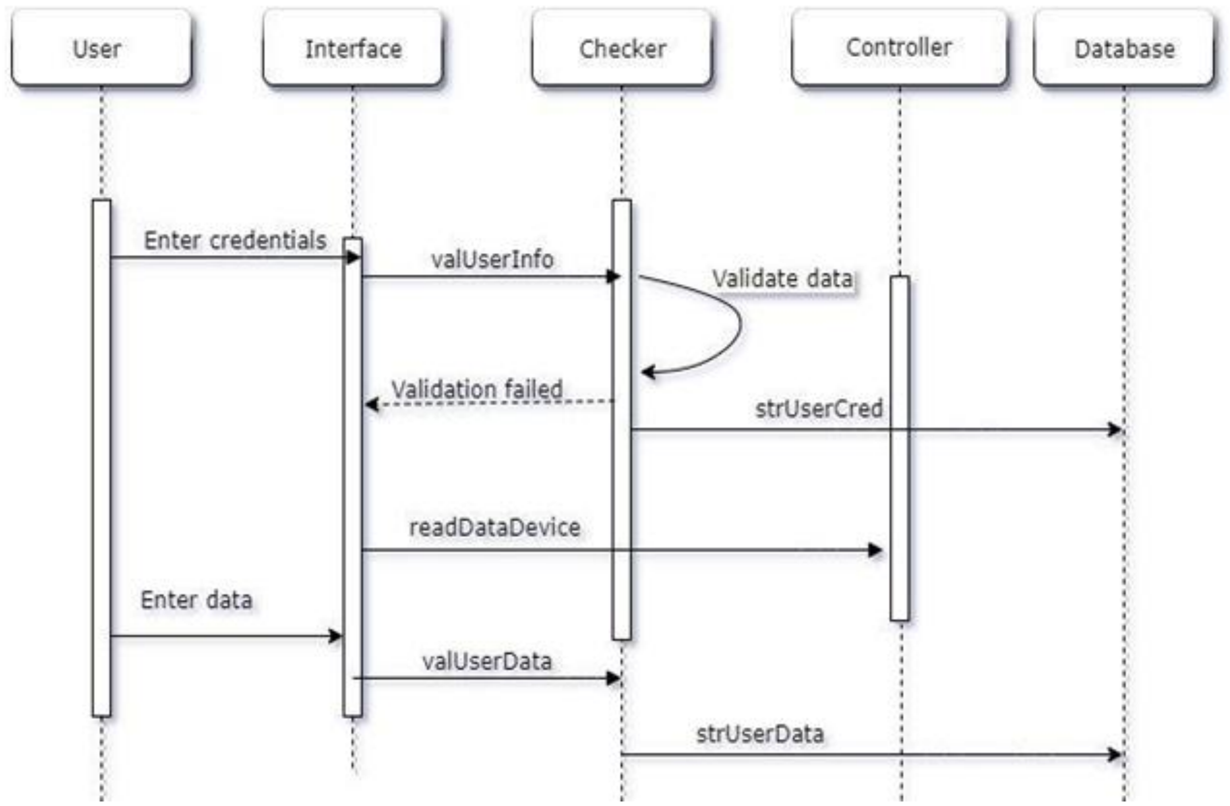
iv. Traceability Matrix

The below table depicts the mapping of the various classes to the use cases that we defined earlier.

	UC1	**UC2**	**UC3**	**UC4**	**UC5**	**UC6**	**UC7**	**UC8**
TextReader	**x**	**x**	**x**	**x**				
Interface	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
Controller	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
Checker	**x**	**x**		**x**				
Communicator	**x**	**x**		**x**	**x**	**x**	**x**	**x**
Database	**x**	**x**	**x**		**x**	**x**	**x**	

7. Interaction Diagrams

i. UC-2: New User Registration:



Description:

The above interaction diagram is for Use Case 2 (New User Registration). The user enters his credentials into the system, and these are then validated and verified by the Checker. If the credentials are valid, the data will be stored in the database. However, if the validation fails, the user will be prompted to enter his credentials again. The user can then enter his health data into the system, which is then validated by the checker (to check for any outliers) and is stored in the database. The data from the streaming device is read by the Controller, which stores this data in the database.

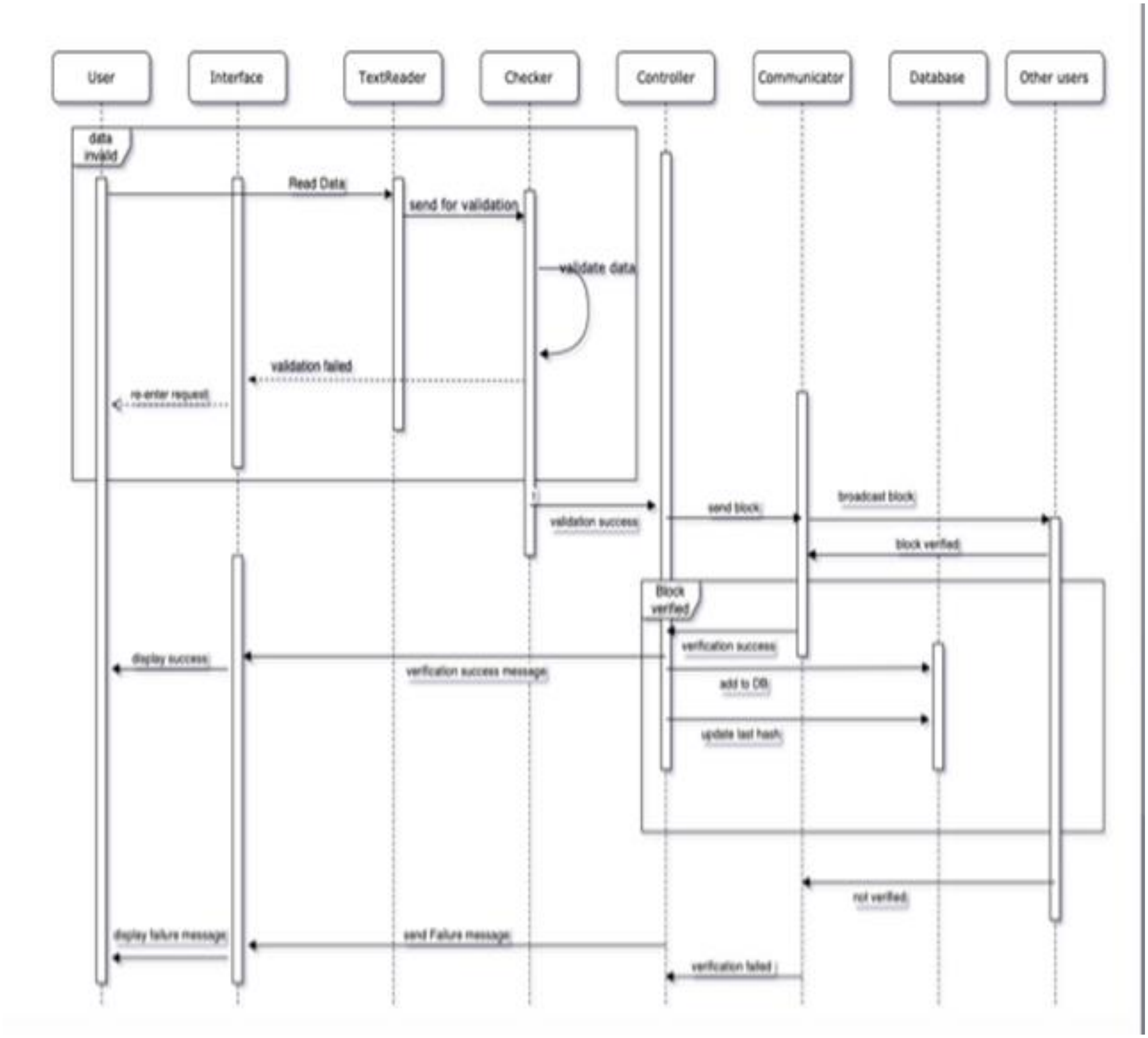
Design Pattern Used: Publisher-Subscriber Design Pattern

Reason of choosing this Design Pattern: Our application, specifically this use case, is event driven, which means that various events are generated and based on these, some actions are performed. As a result, the publisher-subscriber design pattern was found suitable for designing this use case as it separates the publisher and subscriber (the detector and generator respectively) of the event (publisher) and the subscriber (the action doer based on the event or the one who subscribes to the events).This leads to obvious benefits as the publisher has to generate the events and subscriber has to do with the actions based on the events. So the doer actions are clearly separated from the event generator. (Loose Coupling)

Various events-based Subscriber- Publisher pattern used are mentioned below:

Events	**Publisher**	**Subscriber**
validateUserInfo(as shown in the diagram above to check whether the user is already existing in the system , password is matching or not)	UserInterface	Checker
validateUserData(as shown and also mentioned in the explanation, this event is to check whether there are any outliers in the data or not)	UserInterface	Checker
isUserValid(as shown in the diagram above to check whether the user is already existing in the system , password is matching or not)	Checker	UserInterface
isDataValid(as shown and also mentioned in the explanation, this event is to check whether there are any outliers in the data or not)	Checker	UserInterface

ii. UC-3: User Data Addition



Description:

The above interaction diagram is for Use Case 3 (User Data Addition). The user enters the data to the TextReader through the interface. The TextReader send the data to Checker for validation. The checker validates and updates the status to the interface, which in turn sends the message to the user. If validation is successful, the checker requests the controller to forward the block to the other users for verification through the communicator. After the block is verified by the users, it is added to the database by the controller. If the block is not verified, then the controller is sent an error message from the users and it delivers the “failure to add” message to the user through the interface.

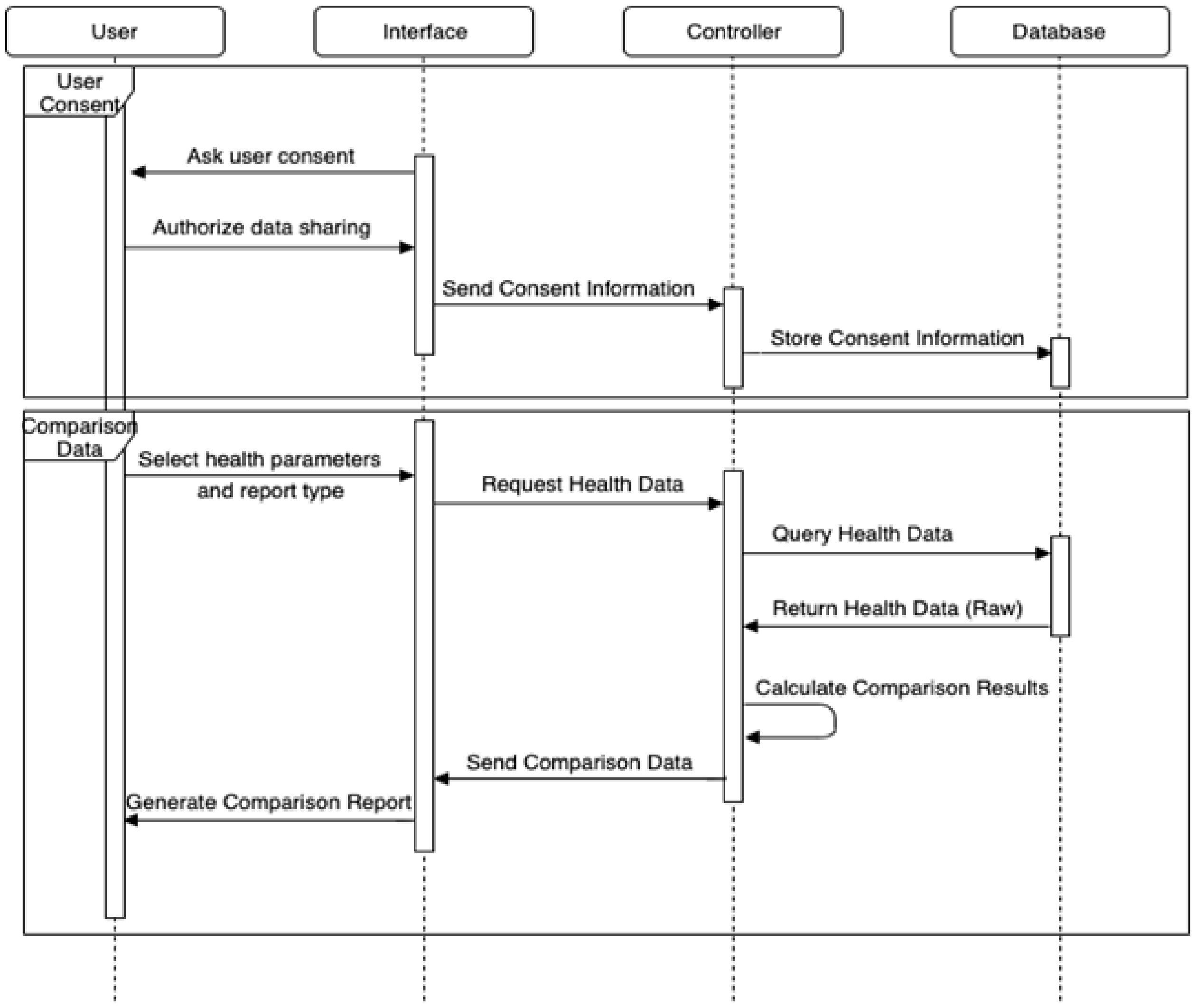
Design Pattern Used:_ Publisher-Subscriber Design Pattern

Reason of choosing this Design Pattern: As mentioned previously, our application and specifically this use case is event driven. So for designing this use case , publisher-subscriber design pattern was suitable since it separates the publisher and subscriber i.e. the detector and generator of the event (publisher) and the subscriber (the action doer based on the event i.e. the one who subscribes to the events).This leads to obvious benefits as the publisher has to generate the events and subscriber has to do with the actions based on the events. So the doer actions are clearly separated from the event generator. (Loose Coupling)

Some of the events-based Subscriber- Publisher pattern used in this use case are mentioned below:

Events	**Publisher**	**Subscriber**
isDataValid	Checker	Controller
isDataValid	Checker	User
isBlockVerified	All Users	Communicator
blockVerifyResult	Communicator	Controller followed by Interface

iii. UC-6: Comparison of Data Report



Description:

The above interaction diagram is for Use Case 6 (Comparison of Data Report). The interface will ask user for consent for sharing personal health data with the system for comparing data with others. Users will authorize and allow data sharing with the system. The interface will send the consent information to the controller which will then store the consent information in the database.

User will select custom health parameters and report type for viewing comparison data and submit the request to the interface. The interface will request health data from the controller. The controller will query database for health data based on users' selected parameters. The database will send queried result sets to the controller in raw format. The controller will calculate the comparison data based on raw data and user id which will later result in calculated comparison data to the interface. The interface will generate the comparison report in the type of user requested and present it to the user.

Design Pattern Used:_ Command based Pattern

Reason of choosing this Design Pattern:_In this use case , as the commands are flowing from one class to another , so in order to take the benefits of the command design pattern i.e. of uniform signature /interface based pattern where the various commands are being provided by the server/doer and its implementation is decoupled from the perspective of the client and various different implementation of the doers and the servers have to change the implementation of these commands only.

Controller provided commands in the use case :

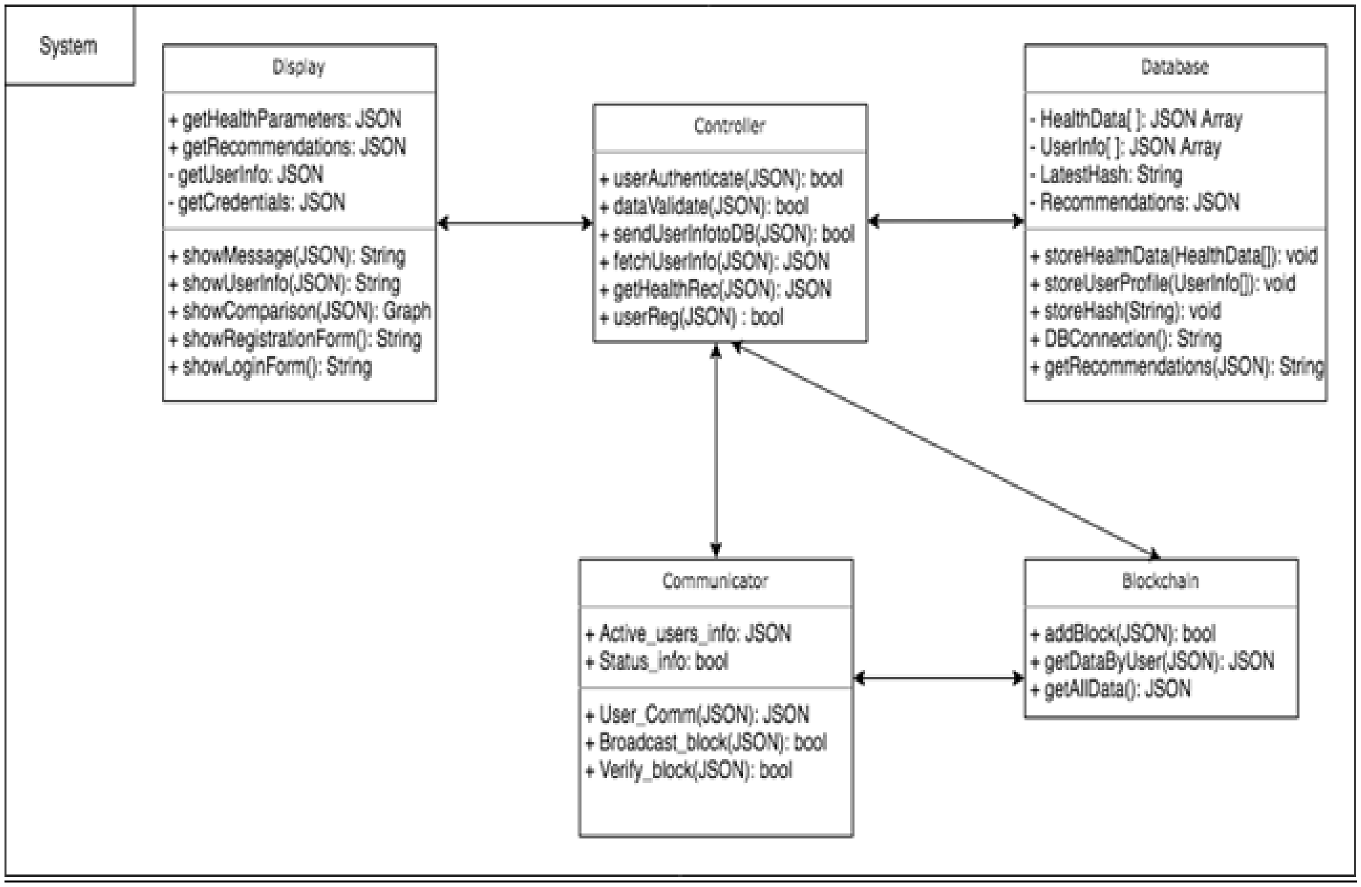
SendConsent,HealthDataRequest and the Interface generates commands :

1. Command (SendConsent) then Command.execute()
2. Command (HealthDataRequest) then Command.execute()

In this manner , with the motivation and goal of producing less coupled design , the command design pattern was used in this use case.

8. Class Diagram and Interface Specifications

a. Class Diagram


System
Display
+ getHealthParameters: JSON
+ getRecommendations: JSON
- getUserInfo: JSON
- getCredentials: JSON
+ showMessage(JSON): String
+ showUserInfo(JSON): String
+ showComparison(JSON): Graph
+ showRegistrationForm(): String
+ showLoginForm(): String
Controller
+ userAuthenticate(JSON): bool
+ dataValidate(JSON): bool
+ sendUserInfotoDB(JSON): bool
+ fetchUserInfo(JSON): JSON
+ getHealthRec(JSON): JSON
+ userReg(JSON) : bool
Database
- HealthData[ ]: JSON Array
- UserInfo[ ]: JSON Array
- LatestHash: String
- Recommendations: JSON
+ storeHealthData(HealthData[]): void
+ storeUserProfile(UserInfo[]): void
+ storeHash(String): void
+ DBConnection(): String
+ getRecommendations(JSON): String
Communicator
+ Active_users_info: JSON
+ Status_info: bool
+ User_Comm(JSON): JSON
+ Broadcast_block(JSON): bool
+ Verify_block(JSON): bool
Blockchain
+ addBlock(JSON): bool
+ getDataByUser(JSON): JSON
+ getAllData(): JSON

b. Data Types and Operation Signatures

i. Display:

The User interface is vital to a software product because it is how the user will interact with the product and if the UI is immersive and user-friendly the product becomes an instant success. The various members of our I/O class are listed below:

1) Attributes:

a) getHealthParameters: This is a JSON object containing key-value pairs that contain information of parameters such as Blood Pressure, height, weight, blood sugar, etc.

b) getCredentials: This is a JSON object which will have only 2 key-value pairs, a user ID and a password.

c) getRecommends: This is a JSON object which contains only 1 key-value pair based on the comparison report the user runs. The value is stored in the database.

d) getUserInfo: This JSON object will return the complete history of the user’s inputs to the blockchain. This will be an array of all the objects that the user has entered before.

2) Functions:

a) showMessage(JSON): String

This function sends a message to be displayed on the screen. It can be a success or failure message that the block has been validated. It can also be a string variable that contains a message that the user can send to another user. It also shows if login is successful or not.

b) showUserInfo(JSON): String

takes getUserInfo object as an argument. Displays this information on the screen.

c) showComparison(JSON): Graph

Displays the comparison report by taking getHealthPara objects as arguments.

d) showLoginForm(): String

Displays a form for the user ID and password. Returns the getUserCred object.

e) showRegForm(): String

Displays a form for registration of new user

ii. Communicator:

The basic idea of the Communicator class is to handle the communication of information among the various users within the system. The attributes and the functions to be performed by the communicator class are as below:

1) Attributes:

a. Active_users_info: JSON

This attribute holds the information about the active users in the system for handling the communication for the peer verification for the implementation of the blockchain system.

b. Status_info: bool

This attribute holds the status information regarding the verification process.

2) Functions:

a. userComm(JSON): JSON

This function is to receive requests for the addition of the block into the blockchain system and then send the status messages/responses to the user.

b. broadcastBlock(JSON) : bool

This function is to broadcast the block of data for verification to the various other users.

c. verifBlock(JSON): bool

This function is to receive the verifications and based on that communicates the status with the Controller whether to add the block or not and works for providing the status messages accordingly.

iii. Database:

Information which is generated by the user and required by the system needs to be preserved for further utilization. This database class will store the required data into the supernode server. The variables and functions utilized by this database class are:

1) Attributes:

a. HealthData[ ]:

This array of health data will be used for temporary processing of the health data variables.

b. UserInfo[ ]:

The information of the user and their profile will be contained in this array.

c. LatestHash:

Hash data of the last block in the blockchain will be contained here.

d. Recommendations []:

This will keep the recommendations for the users as requested through the controller.

2) Functions:

a. storeHealthData(HealthData[]): void

This function will store the health data of user once it has been validated and verified.

b. storeUserProfile(UserInfo[]): void

User profile information will be stored through this function.

c. storeHash(String): void

As the latest hash will be required to identify the latest block for the particular user, it will be stored through this function.

d. DBConnection(): String

This function will return the DB Connection to the requester/web server

e. getRecommendations(JSON): String

Function returns the recommendations for specific values of health data to the controller

iv. Blockchain

The Blockchain class temporarily holds the data and manipulates the operation of storing, adding and retrieving the data from the database. The various functions performed by the Blockchain class are listed below:

1) Functions:

a. getAllData(): pre-defined JSON format

This function will return a whole set of data retrieved from the database. This return value will be in JSON format and given as the background data for comparison. The function will go through every block and form a result with the latest data.

b. getDataByUser(JSON): pre-defined JSON format

This function will take a user's id as an input and return all the data from blocks that stored on his/her chain. The function will start retrieving the data from the user's initial block and record every data when going through the blockchain.

c. addBlock(JSON): Boolean

This function will take the verification information from the communicator and do the adding and again verifying the block operation. The function will return a Boolean result stands whether a new block is added to the blockchain.

v. Controller

The basic idea of the Controller class is to control the flow of information to the other modules within the system. The various functions performed by the controller are listed below:

1) Functions:

a. userReg(JSON): bool

This function is used to register a new user into the system. It takes all the credentials that were given by the user as the input.

b. userAuthenticate(JSON) : bool

This function is used to authenticate the user, which means that the system checks if the user is present on the system database or not. It returns a Boolean variable to indicate the status.

c. dataValidate(JSON) : bool

After the user enters his data into the system, this function checks the validity of the data, which basically checks if the data entered has any outliers or not. It returns a Boolean variable to indicate the status.

d. sendUserInfotoDB(JSON) : void

Once the user inputs his personal health data into the system, this function sends the data to the database, where it is safely stored.

e. fetchUserInfo(JSON) : JSON

This function returns the user information when the user requests it from the system.

f. getHealthRec(JSON) : JSON

This function provides health recommendations to the user on request, depending on the health conditions of the user. It is then sent to the display for the user to view it.

c. Traceability Matrix

The below table depicts the mapping of the domain concepts with the various classes listed in the previous section and shows how the domain concepts are used in each class:

	Classes				
Domain Concepts	**Display**	**Controller**	**Database**	**Communicator**	**Blockchain**
TextReader	**X**	**X**			
Interface	**X**	**X**			
Controller		**X**			
Checker		**X**	**X**		
Communicator		**X**		**X**	
Database		**X**	**X**		**X**

9. System Architecture and System Design

a. Architectural Style

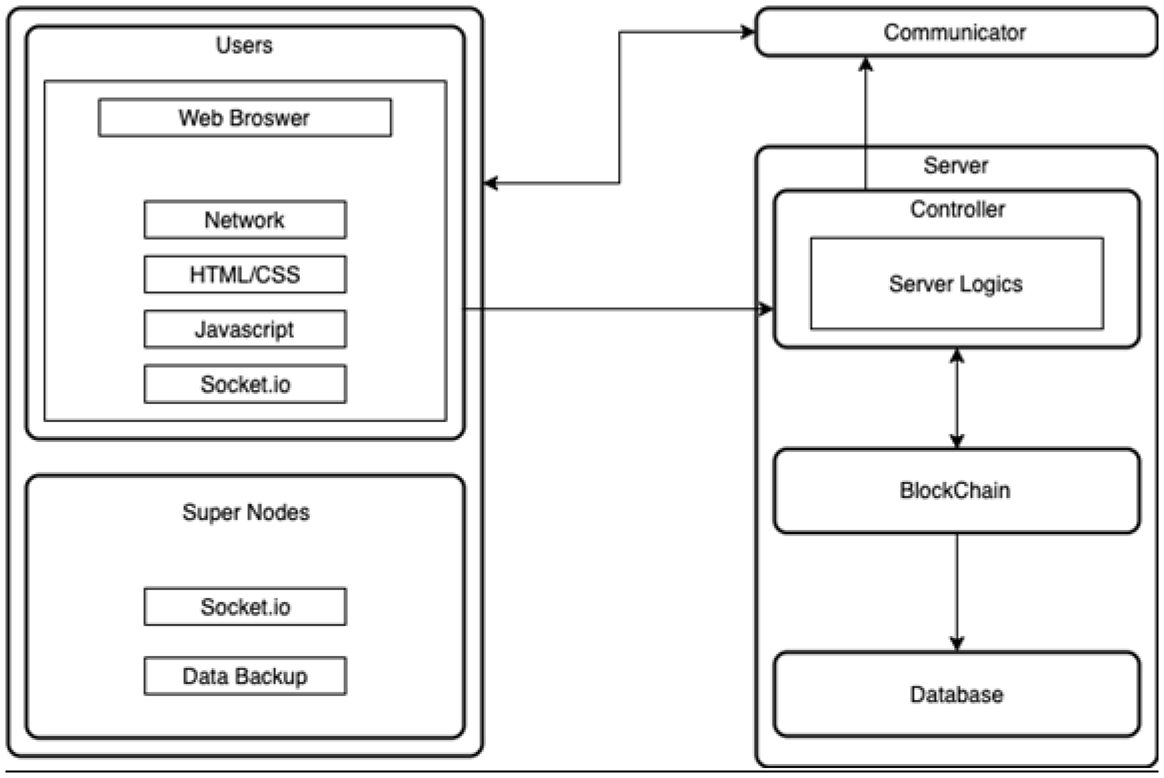
Our system will not follow the architectural model of common existing blockchain practices. To be fair, bitcoin-like architecture is kind of silly and has a lot of defects. Ideally, every user will store the transaction information from all the other users. But terabytes of that information are still getting larger, it is ridiculous to let each of all the users hold terabytes or even petabytes in the future just for a paying system. Also, the concept of decentralization is a joke nowadays. Most of the bitcoin users are using third-party platform to hold their assets. And the news of bitcoin stealing from the inside of the platforms is not rare.

So we would use a hybrid architectural model of client/server model and a peer to peer model. On the server-side, it will have all the functions of the controller, communicator, blockchain, and database class. The server will do the main data storage and manipulation. It will also take all the requests from all the users and return the value accordingly. The client will send HTTP requests like GET and POST to the server through a lightweight front-end interface.

As for the peer to peer part, we would have more than the maximum number of online users' virtual machines identified as super-nodes. All the users and super nodes will communicate through the communicators in the server to each other and get the verification done. The data will be backed on super-nodes and all the super-nodes will synchronize the data after each addition operation.

By this architecture, we would have an easily updated interface for the user. As for the backend side, we could adopt distribution methodologies to scale up.

b. Identifying Subsystems



In this system, there are three subsystems. The client-side - the web browser, the server-side, which holds all the logic and algorithms, and a communicator side, which contains all the users and super nodes. The client-side means the website framework, and it contains the structure of the user interface. The user will send HTTP requests to the server, and the server will analyze, process and response the request back to the user. For the verification part, the user will send a request and enough information to the server, and the server will broadcast and receive all the verification information, then do the following operations. When involving data manipulation, the server will store the data through the blockchain module.

c. Mapping Subsystems to Hardware

As we have shown in the diagram above, mapping the subsystems to specific hardware is simple. The web application will run in the browser of the User's PC and the data entry, comparison reports as well as the forums will be running on the interface of this web application. All interactions for input and display of data is through the PC.

The super nodes are special instances that will not be inputting any additional data from their side. Their only purpose is to validate the blocks that are being entered by other users and if they are validated, sending them to the server for further processing. If they are not validated, they will send the failure message through the communicator to the respective user. They will also store the data as a backup.

The web server will be handling the input of the validated data into the blockchain. Every time the user needs the data from the database to generate the comparison report, the logic for such a comparison will be done in the server itself. The server will maintain the blockchain and the database which will be populated based on the blockchain entries.

Lastly, the communicator will be responsible for maintaining all channels for broadcasting messages from the server to all the users as well as communicating amongst users.

d. Persistent Data Storage

A database is needed to store the blockchain, information regarding the users which include user login information and the statistical data of the population. MongoDB provides us with a database that stores data in key-value pair format. The wide variety of fully developed features allows us to focus more on the actual organization and management of the data in relation to other modules. All that is needed is a simple call to the database to retrieve the raw data and the custom-designed objects shown in the Class Diagram then do their own processing of the data. The key is required to acquire data in this application which makes using this database simple. MongoDB helps us store data in the cloud which is important for us to host a website accessible to everyone. The goal is for the user to record his/her health parameters to compare with population descriptors. Other objects illustrated in the Class Diagram do not have direct access to the database. The database will be only accessible to the controller as shown in the class diagram. The controller will interact with the database and issue requests for various types of data that are stored in the database that the user interface would display or graph on the screen and data that the controller would compare. Thus, the Controller is the only object that has direct access to the database.

e. Network Protocol

In our system, all the communications will be encrypted by SSH and HTTPS. For the verification part, which involves most of the online users communicate, we will adopt the Socket.io protocol.

f. Global Control Flow

Our system’s actions are based on the events generated through the User Interface thus making it an “event-driven” system that depends on the user interaction. The user can choose to enter/modify health information or view their health analysis information in an order which they prefer.

Also, our system does not incorporate any hard timeouts which the user needs to adhere to. Once the user is on our system’s webpage, they have the freedom to explore its functionality as and when they want. Being an “event-response” system there are no hard deadlines for our system as compared to the real-time systems.

g. Hardware Requirements

The hardware that we will be utilizing for the project are listed below:

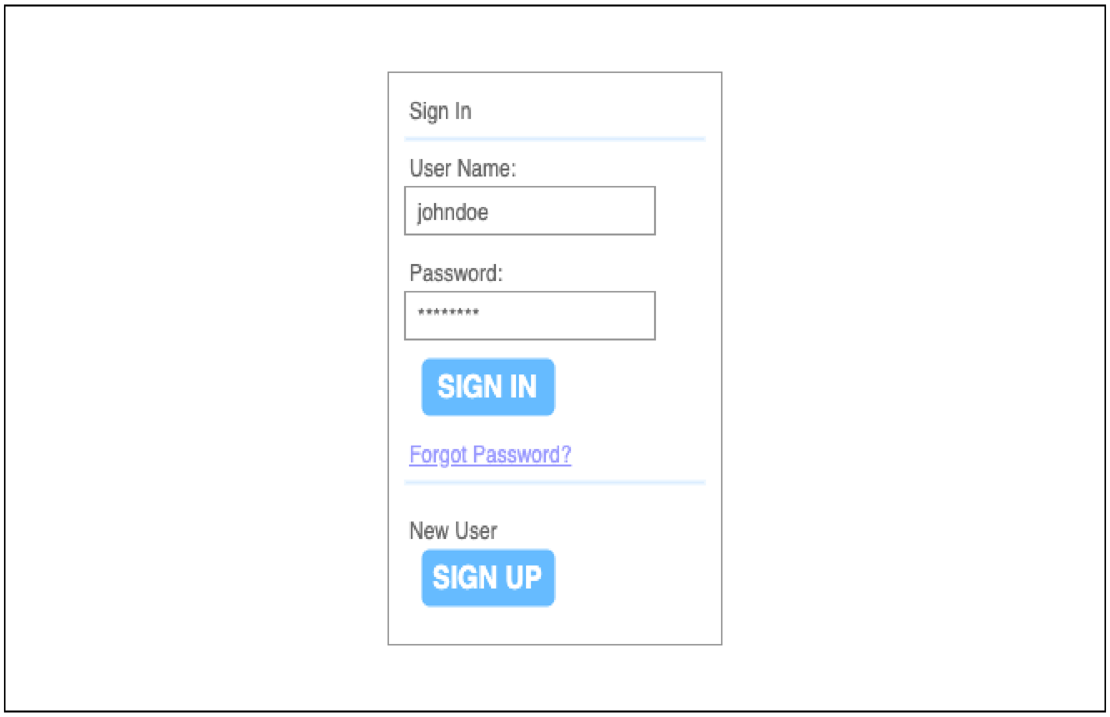
1. A screen displays.
2. Storage space for the local blockchain.
3. Network communication with the main server (database).
4. A mobile device that could access the system website.

Note that all the requirements could be fulfilled by a personal laptop.

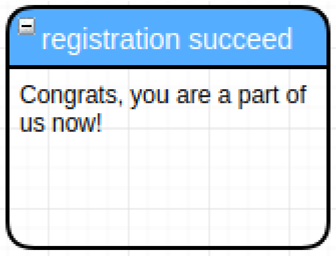
10. User Interface Specification

a. Preliminary Design

i. Registration

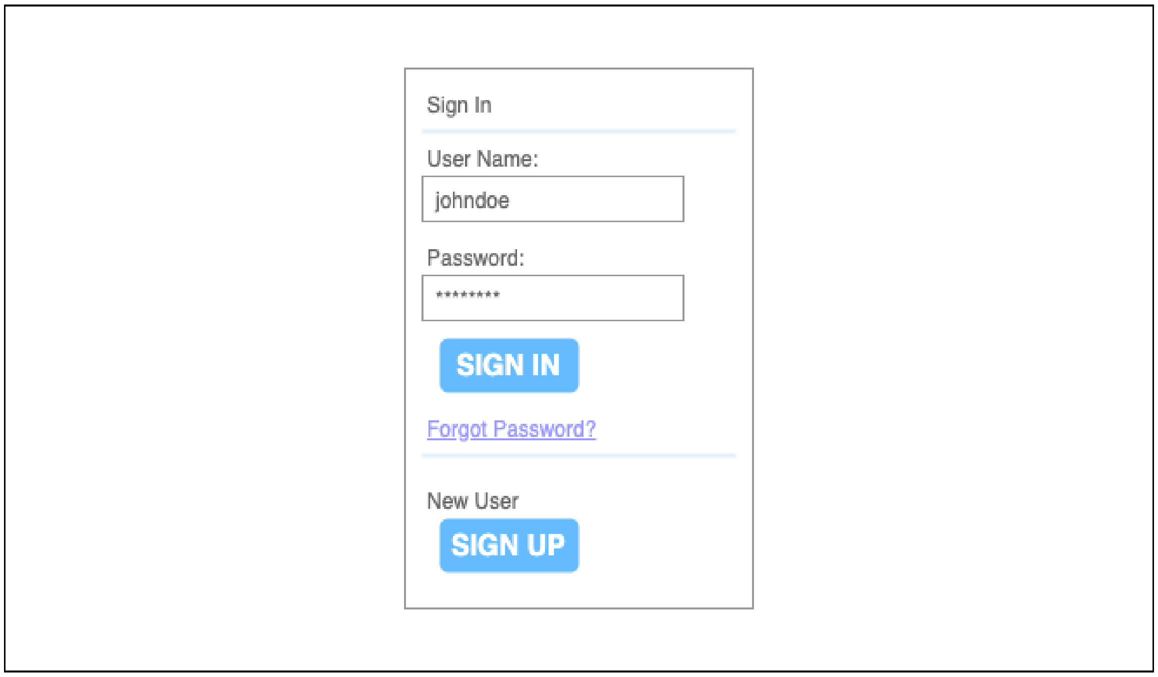


Once a valid input (user ID and Password) is submitted, a “registration succeed” message should be shown via the UI.



Congrats, you are a part of us now!

ii. Log in


Sign In

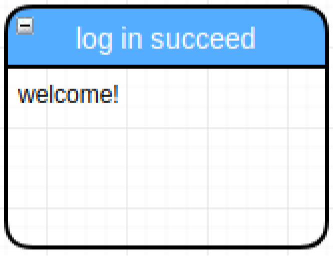
User Name:

johndoe

Password:

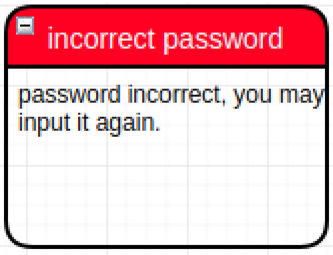
****

If a registered ID and correct password are submitted, a “login succeeded” message should be shown.


log in succeed

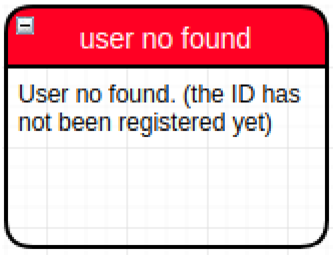
welcome!

If a registered ID and wrong password are submitted, a “incorrect password” message will be shown.


incorrect password

password incorrect, you may input it again.

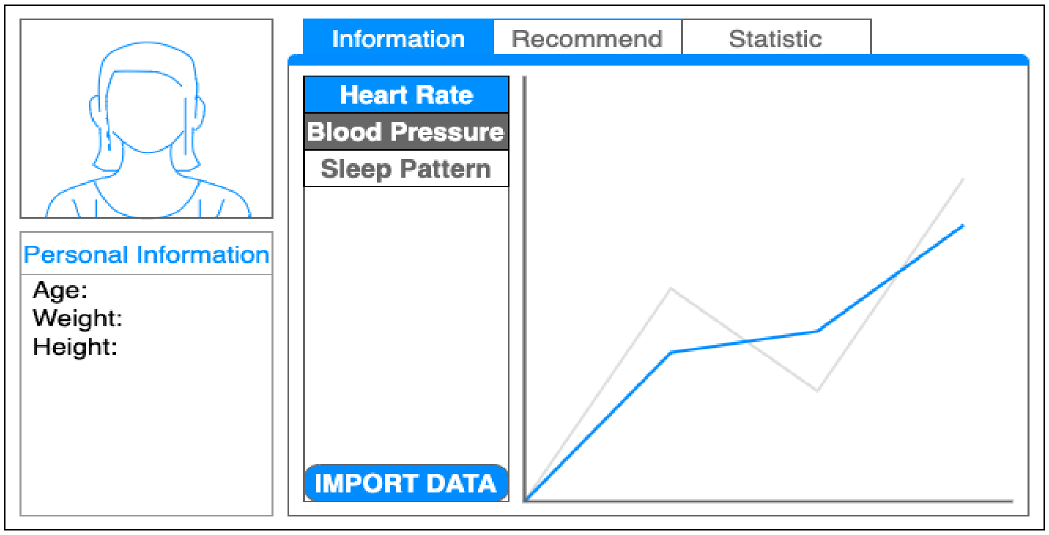
If an unregistered ID is submitted, no matter what the password is, it will give an “ID no found” message.


user no found

User no found. (the ID has not been registered yet)

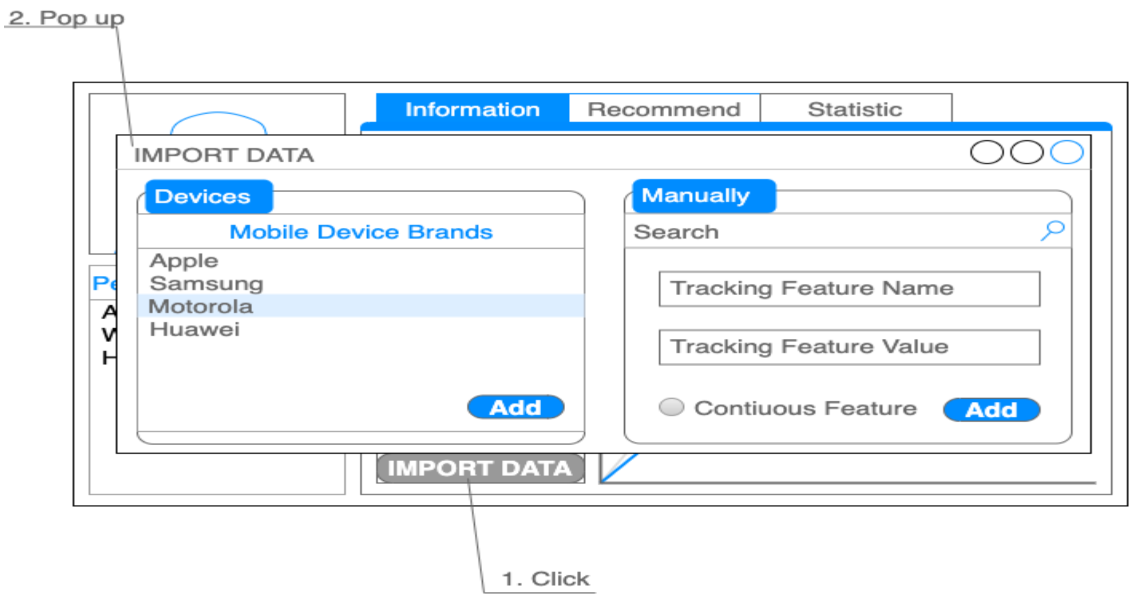
iii. Personal Information

Once a user successfully logged in, the personal information should be shown directly, along with the historical diagram.



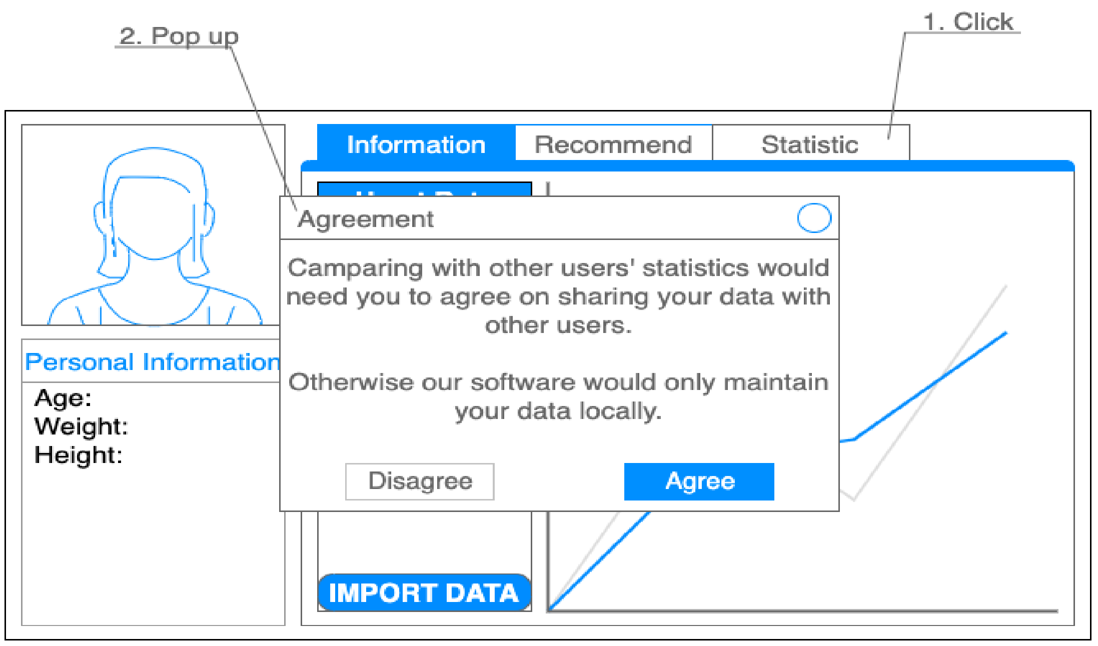
iv. Data Import

Once the personal information page is displayed, users could click on the "import data" button, and a data input page would be shown where users could import data freely.



v. Statistics

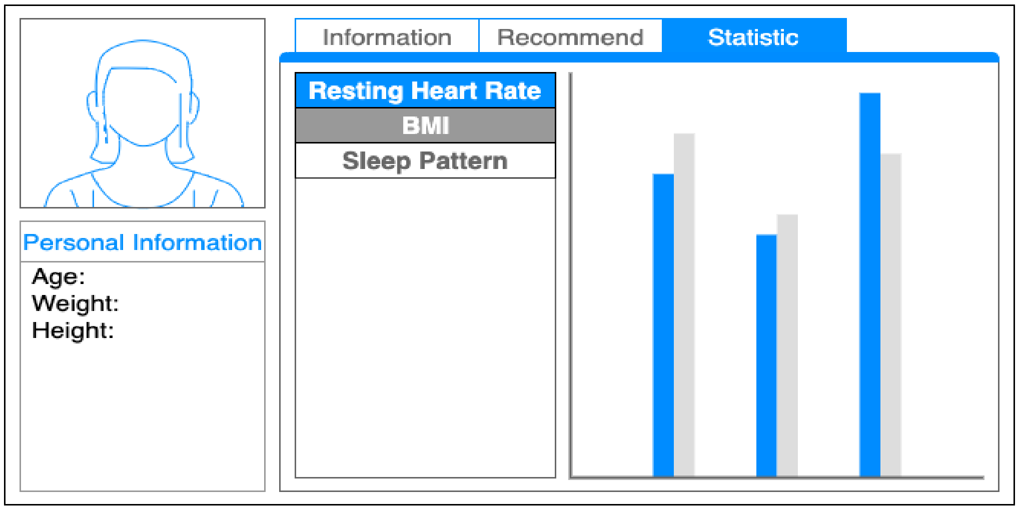
Users could click on the statistic page to view the comparison, but a consent page that inquire users if they are willing to share their personal data for the statistics would pop up.

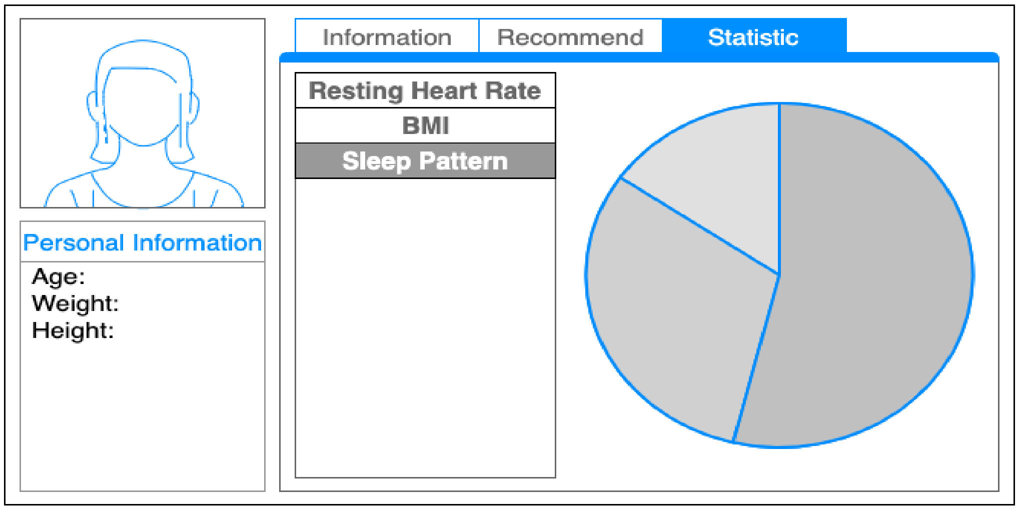


Camparing with other users' statistics would need you to agree on sharing your data with other users.

Otherwise our software would only maintain your data locally.

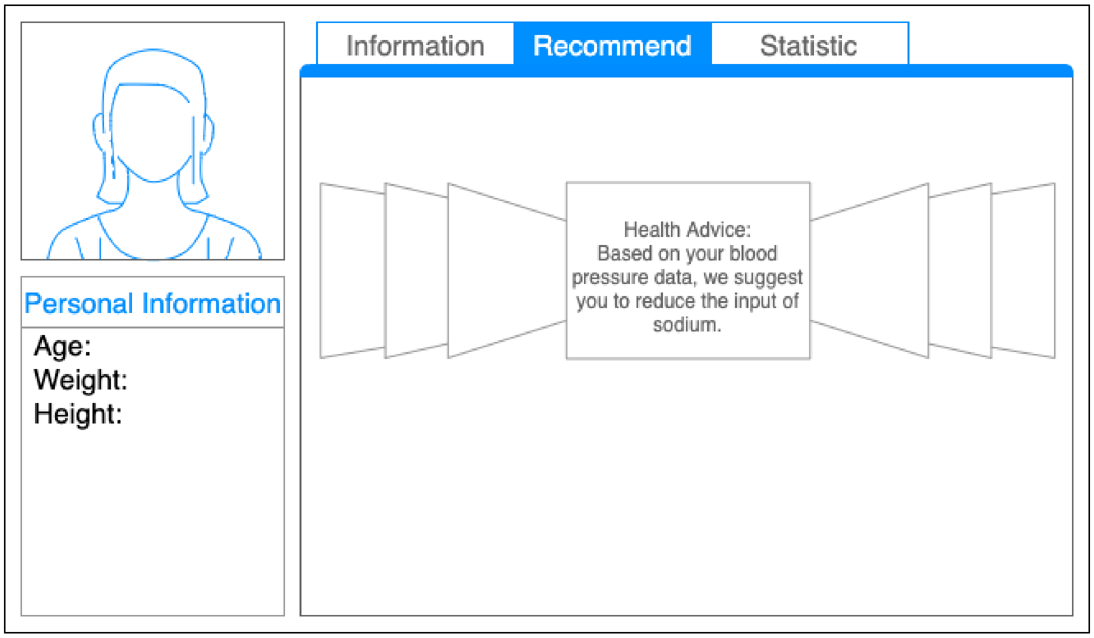
After that the statistic page would be shown





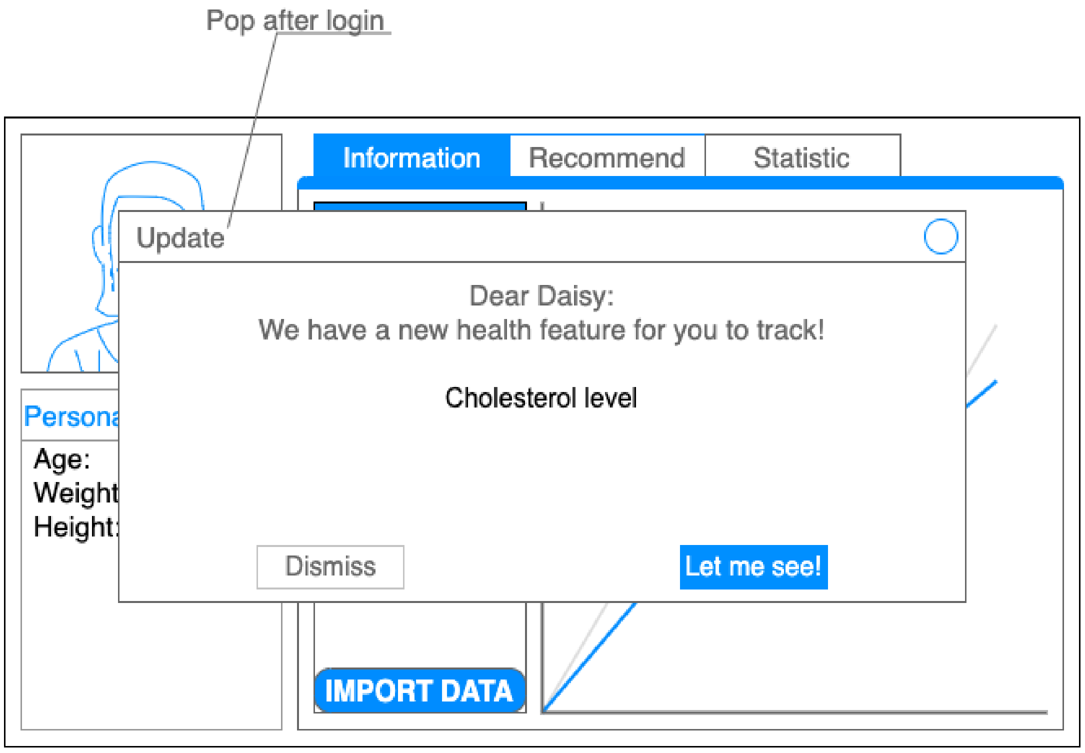
vi. Recommendation

The user could click in the recommendation and the UI would provide the health recommendation for the him or her based on his or her health data.



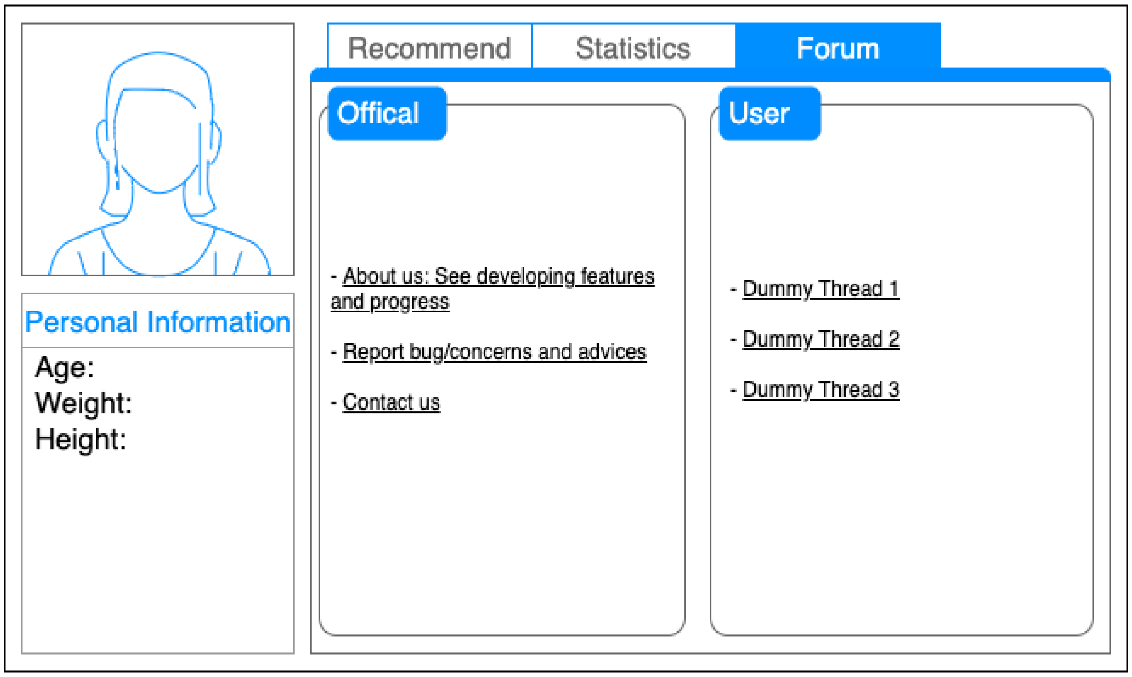
vii. New Update

If a new parameter is updated, the UI would notify the user with a notification page.



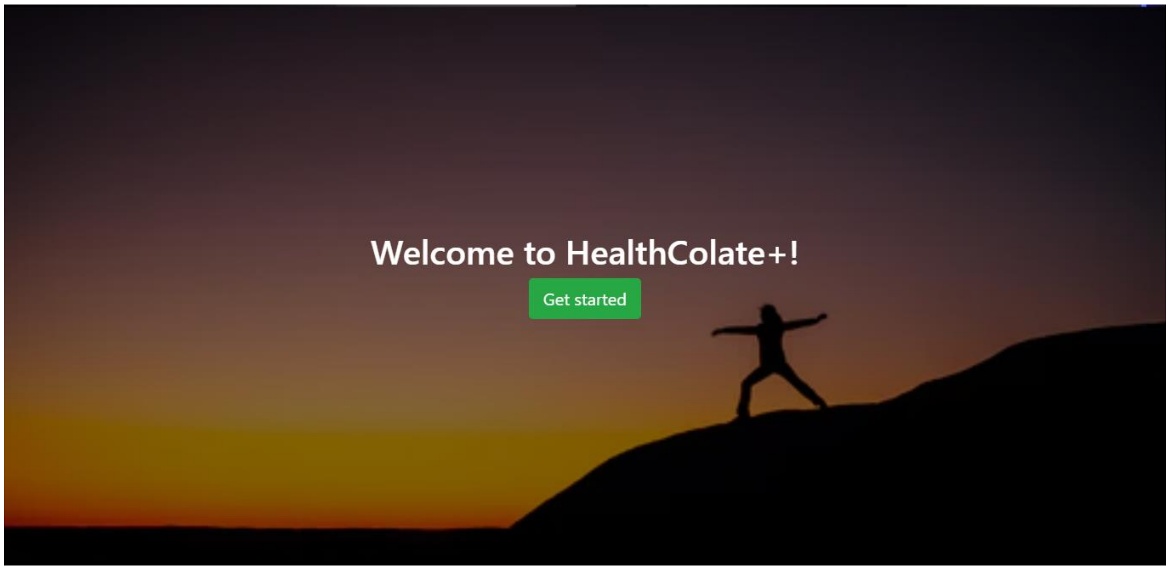
viii. Forum

A forum page would be popped once a user clicks on the forum button, the support, communication and system announcements are all there.

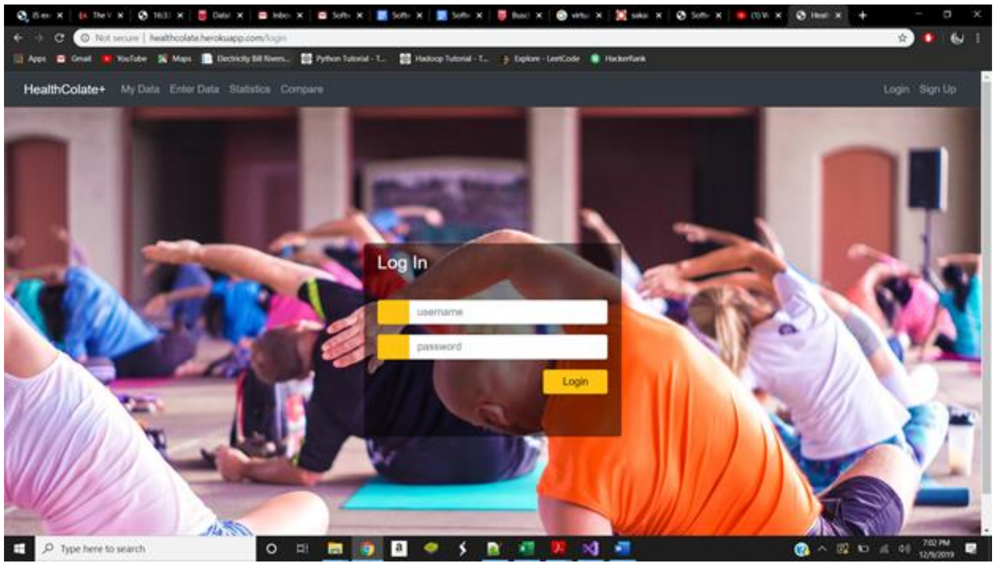


ix. Further Developments

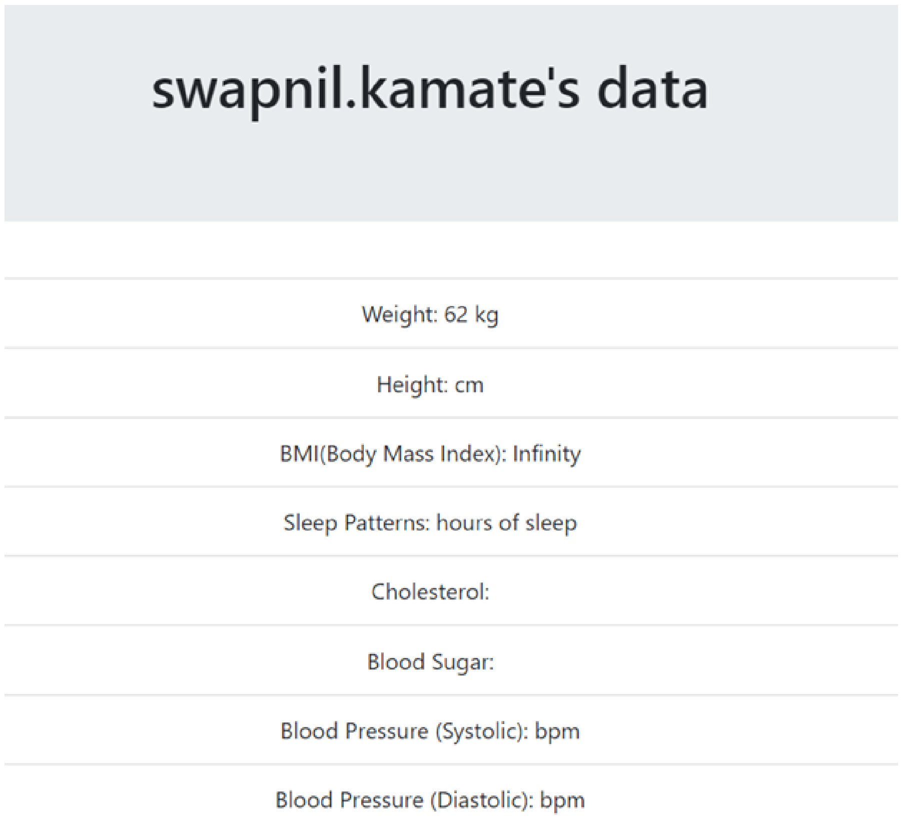
In the next stage of developing our application, we improvised on the User Interface that we had designed in the first stage. First, we renovated the display of the home page for our website, which is shown below:



We also improved the login page, where users could log into the application:



Once the user logs into the website, he is taken to the landing page, which basically displays his data that he had entered. The landing page is shown below:


swapnil.kamate's data

Weight: 62 kg

Height: cm

BMI(Body Mass Index): Infinity

Sleep Patterns: hours of sleep

Cholesterol:

Blood Sugar:

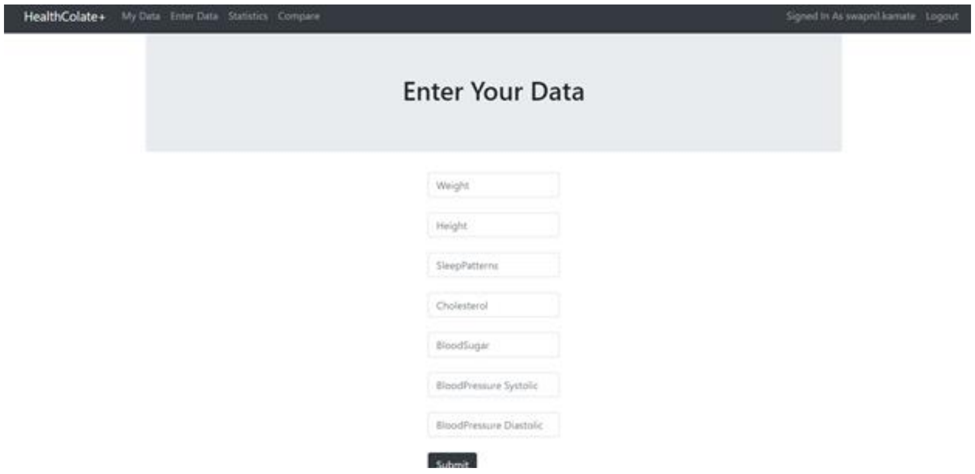
Blood Pressure (Systolic): bpm

Blood Pressure (Diastolic): bpm

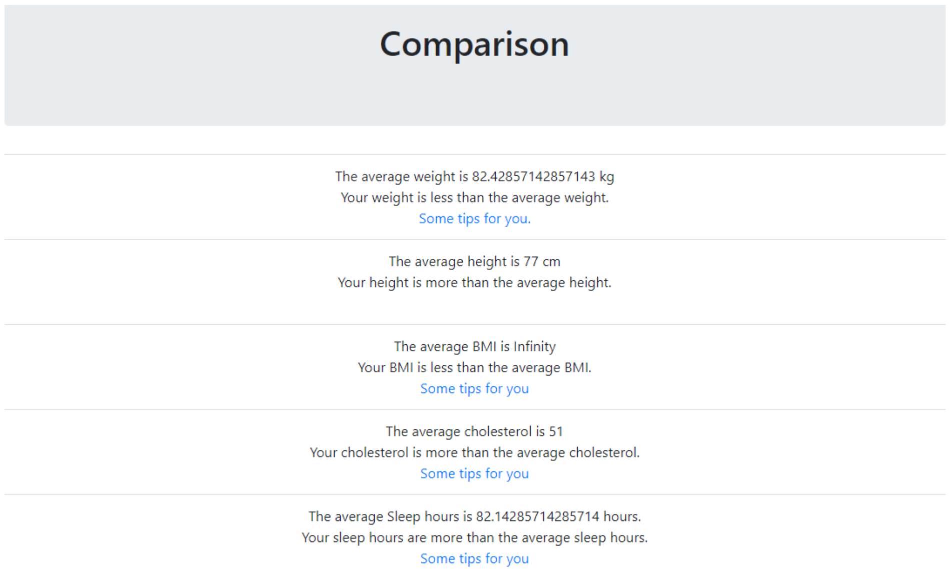
If the he/she is a first-time user, the landing page would display as shown above, but with no data, since it hasn't been entered yet. Once the user has entered his data into the website and submit it, it gets stored in the database and can be displayed appropriately.

The webpage also has options for entering data, viewing the comparisons with other users and statistical analysis of the user's health.

The page for entering data is shown below:



When the user opts to view the comparison tab, what his position is among the various users on the network, he is compared against the average value of all the parameters, and depending on this interpretation, health recommendations are provided. This page is shown below:


Comparison

The average weight is 82.42857142857143 kg
Your weight is less than the average weight.
Some tips for you.

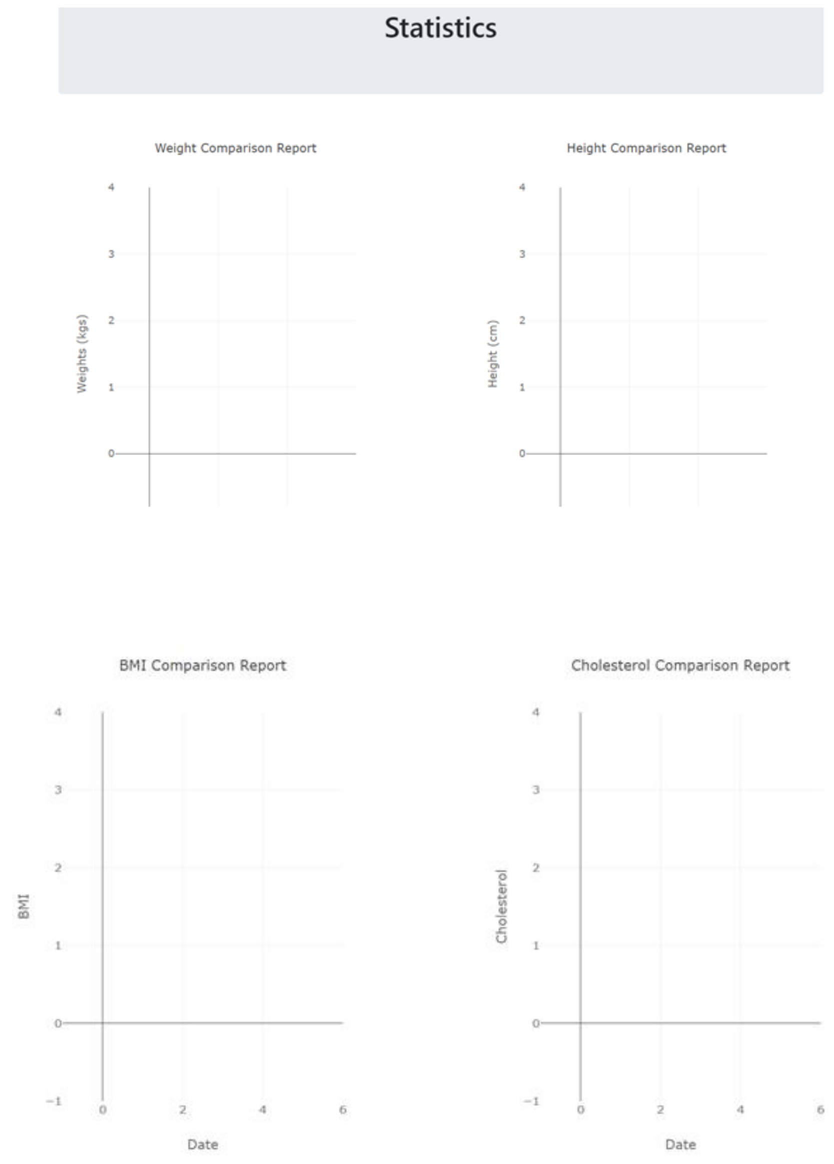
The average height is 77 cm
Your height is more than the average height.

The average BMI is Infinity
Your BMI is less than the average BMI.
Some tips for you

The average cholesterol is 51
Your cholesterol is more than the average cholesterol.
Some tips for you

The average Sleep hours is 82.14285714285714 hours.
Your sleep hours are more than the average sleep hours.
Some tips for you

Finally, when the user goes to the Statistics tab, he can view the charts of his various health parameters. A sample graph is shown below:


Statistics
Weight Comparison Report
Weights (kgs)
Height Comparison Report
Height (cm)
BMI Comparison Report
BMI
Date
Cholesterol Comparison Report
Cholesterol
Date

11. Design of Tests

a. Class Tests

Goal: To test the functions of the application, we plan to test the following:

a. Basic view of the web page.
b. Entering the user credentials and displaying the same on the screen.
c. Store the user data on a database.
d. Display the history and present conditions of the user on the screen.
e. Display the graphical representation of the health conditions of the user.
f. Display recommendations based on the current health of the user.

Results: The results for the above listed goals are as follows:

a. The web page was successfully displayed.
b. The user credentials entered by the user were successfully displayed.
c. The data entered by the user was stored on the database.
d. The past health conditions of the user were successfully displayed.
e. The health data of the user was displayed as graphs.
f. Health recommendations were displayed based on the user's health

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

Use Cases Completed

I. UC-1: Logging in

II. UC-2: New Registration

- The login and registration system work for new and existing users respectively.
- The user ID and password are stored in a MongoDB database.
- If the user enters wrong credentials three times in a row, then the system locks him out completely and informs the system admin about possible brute force attack.

III. UC-3: User data addition

- User data is added to the database as JSON objects.
- Previous hashes and current hashes are being calculated and appended to the JSON object.
- However, these objects can be modified in the database by the system admin and the immutable feature of the blockchain has still to be added.
- The user can update his parameters but then that will be inserted in the main database as any other data entry and will be counted as a second entry instead of an update in the user's information.

IV. UC-4: Validation of Data

- The JSON object is sent by socet.io to other users of the network for validation
- The user will calculate the hash independently and send a success or a failure message to the system admin
- If more than 50% of the users give a success message, the block is added to the database. The blockchain infrastructure is currently in development.

V. UC-6: Comparison of data report

- The data can be displayed on the webpage as histograms, bar graphs and pie charts using plotly.js and a MySQL database
- The data query is generated manually and hardcoded in the PHP file. We are currently working on integrating a button to select the different parameters to be selected whose data will be used to make a graph.
- The UI of the comparison report was made independently of the main data addition and login module and the integration of the same is underway.

VI. UC-5: Historical Data Representation

- The user should be able to see his history of updates and now we are not able to group the updates together and present it to him like we envisioned it to do.
- The latest user data will be used for the comparison report generation and now, all the commits made by the user are used for the data analysis

VII. UC-7: Health Recommendations

- The health recommendations ideally should be generated by ML algorithms that will generate them based on the comparison report output. We however will try to implement a basic version wherein the health recommendations will be stored in a database.
- The health recommendation will be displayed irrespective of the user needing them and which health recommendation is to be shown will be decided only by the parameters that we will use to generate the comparison report.

VIII. UC-8: User communication

- The users should be able to communicate with the system admin and the other users regarding any issue they have with the system or just in general.
- We can send messages over socet.io but we still must figure out how to maintain a record of the messages.

Current Status:

In the first phase of our project, we implemented a primitive version of the basic blockchain-based health system. This phase was meant to form the most basic and essential functions of our system: adding data blocks to the main and local block-chains, communicating with the database and simple user interface. We added three sample users to the network. One of the three users could add their data to the blockchain, which could be verified by the other users. If any outliers exist in the data entered by the user, the user would get an error message displaying the reason. The data validator of the system would check if any data entered by the user is invalid and if so, an error message pops up prompting the user to re-enter his data. We used JavaScript and MongoDB, for the creation of the blockchain and backend part, where we saved the user's data and shared it among other users. We completed this phase by October 29th, 2019.

Further, a super node was made available within the network, which was used to store the blockchain data of all the users and essentially served as the database for the users. This was basically a relational database, which was designed using SQL.

In the second phase of the project, we designed the user interface with basic functionalities, like allowing the users to register and login to our system. While registering with the system, the user was given an option to share his personal data with other users and only if he consented, the data was broadcast to the other users who could compare the data and verify its authenticity. Once the user logged into the system, his credentials were saved onto the system and he could view his current health data, while also updating it. If the user added new data, all the other users on the network would be notified. Users could also generate historical reports, which provided details of his previous health data and could also compare their data with the data of the other users. The second phase included provisions for data analysis that was done using the data stored in the blockchain. Necessary upgrades were made to the user interface. We used the open-source library like Electron to create this sort of desktop application.

This phase was completed by November 20th, 2019. By this time we had the application which was able to perform most of the main functionalities that are necessary for the application.

In the third phase, we addressed the scalability issues related to the network that we created. In other words, we modified the system to accommodate multiple users on the network. We also provided recommendations to users based on their present health conditions, with each metric having its own priority depending on the seriousness of the health conditions. For example: heart rate and blood pressure had a higher priority compared to sleep patterns. A forum for the users to communicate with other users was created during this stage.

If the user faced any problems with logging into the system, a support system was used to address the issues, which also addressed issues related to login, data updating, health parameter comparison etc.

Breakdown of Responsibilities:

Depending on the strengths of our project members, we have split the work in the following way:

- Pranit Ghag: MongoDB database configuration, cloud database management and user interface
- Vikhyat Dhamija: Socket.io and user interface
- Sen Zhang: Socket.io for blockchain verification communication and blockchain infra.
- Shounak Rangwala: Front end data visualization, project management
- Pratik Mistry: Cloud database management, Front end data visualization
- Pranav Shivkumar: Basic web page design, documentation
- Amod Deo: Documentation, User interface, Web page design, e-archive
- Swapnil Kamate: User interface, documentation, project management
- Lizhe Wei: Documentation, blockchain infrastructure.

The integration is coordinated by Sen Zhang, Vikhyat Dhamija, Pratik Mistry, Pranit Ghag and Shounak Rangwala.

The testing is coordinated by Swapnil Kamate, Pranit Ghag and Amod Deo.

13. REFERENCES

[1] Marsic, Ivan. "Software Engineering book"
<http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf>

[2] Marsic, Ivan. “Safe Sharing of Population Descriptors”
<https://content.sakai.rutgers.edu/access/content/group/fbfe7282-89ce-4a4d-a619-943dee895665/Programming%20Project/Programming%20Project_%20Safe%20Sharing%20of%20Population%20Descriptors.pdf>

[3] Marsic, Ivan. “Report #1 - User Effort Estimation”
<https://www.ece.rutgers.edu/~marsic/Teaching/SE1/report1-appA.html>

[4] Rosic, Ameer. “What Is Blockchain Technology? A Step-by-step Guide For Beginners”
<https://blockgeeks.com/guides/what-is-blockchain-technology/>

[5] Rouse, Margaret. “What Is Encryption? - Definition from Whatis.com”
<https://searchsecurity.techtarget.com/definition/encryption>

[6] “Global Glossary Of Blockchain Terms 2.0 in 5 Languages”
<https://blockchaintrainingalliance.com/pages/glossary-of-blockchain-terms>

[7] Software Engineering I Lecture Slides

[8] “Fitbit Health Monitoring Analytics”
https://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2014-g5-report3.pdf

[9] “GRASP (General Responsibility Assignment Software Patterns)”
https://whatis.techtarget.com/definition/GRASP-General-Responsibility-Assignment-Software-Patterns

[10] “Cohesion_wikipedia”
https://en.wikipedia.org/wiki/Cohesion_(computer_science)

[11] “Monitoring the health of web page analytics code”
https://patents.google.com/patent/US20110035486