RUTGERS UNIVERSITY

SOFTWARE ENGINEERING 1

Project: Blockchain-Based Health Monitoring System

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

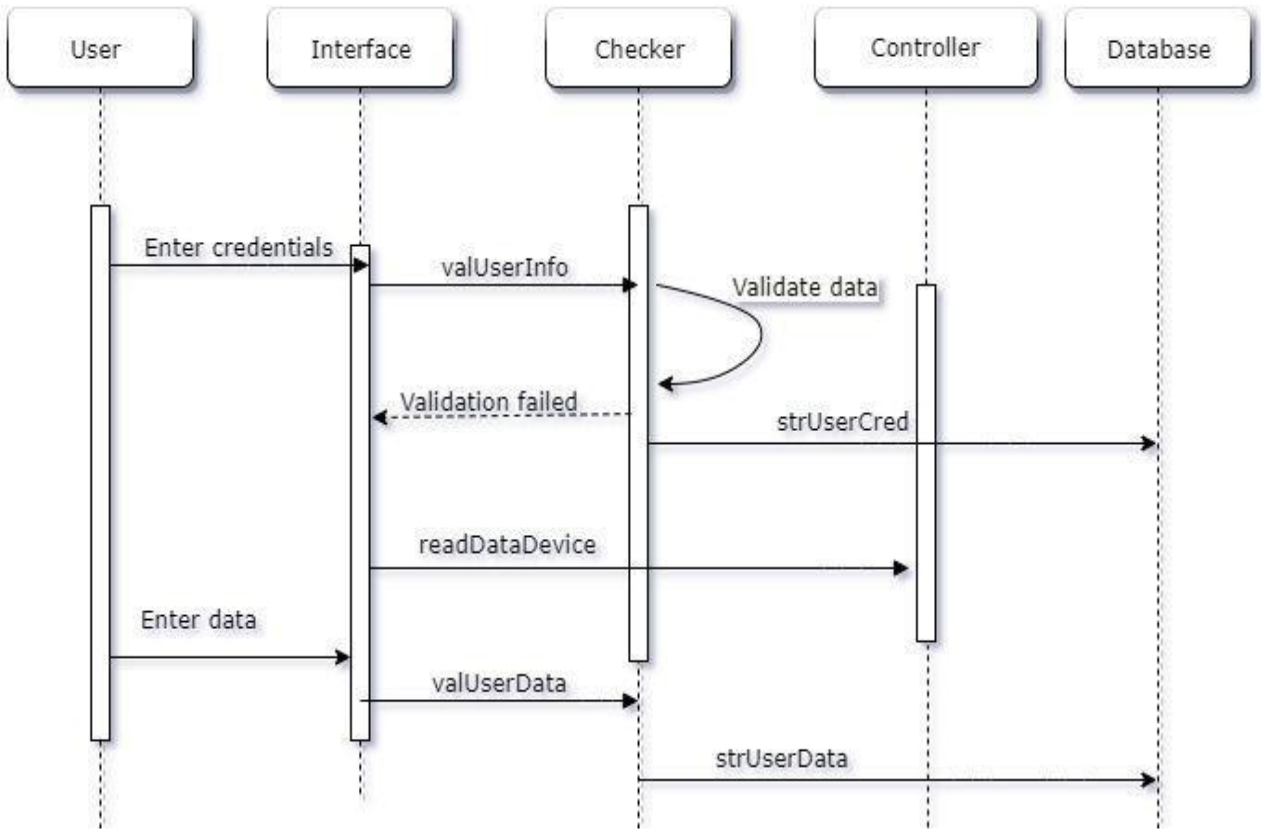
Table of Contents

SECTION 1

1. Interaction Diagram

1.1 Sequence Diagram:

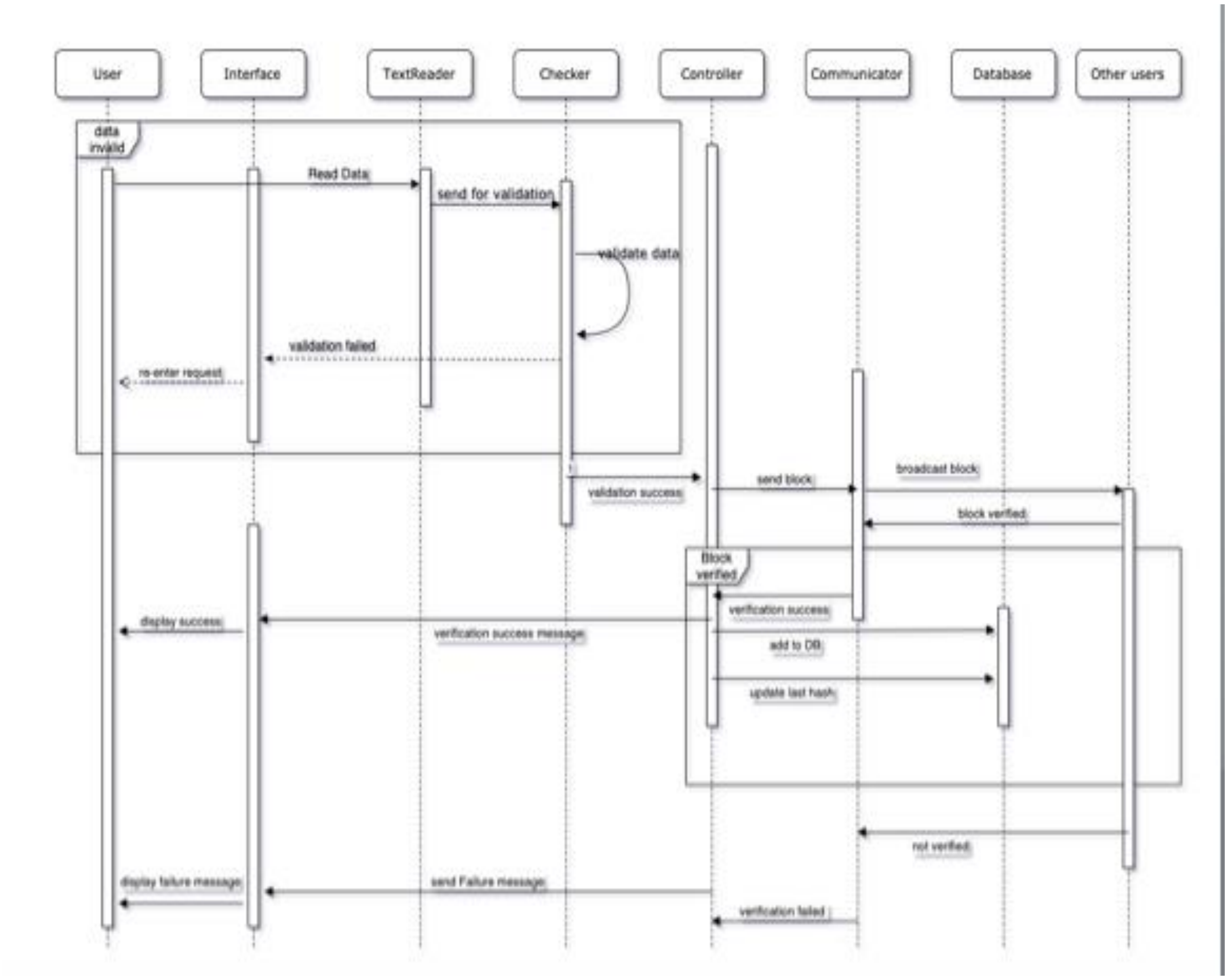
I. UC-2: New User Registration:



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.

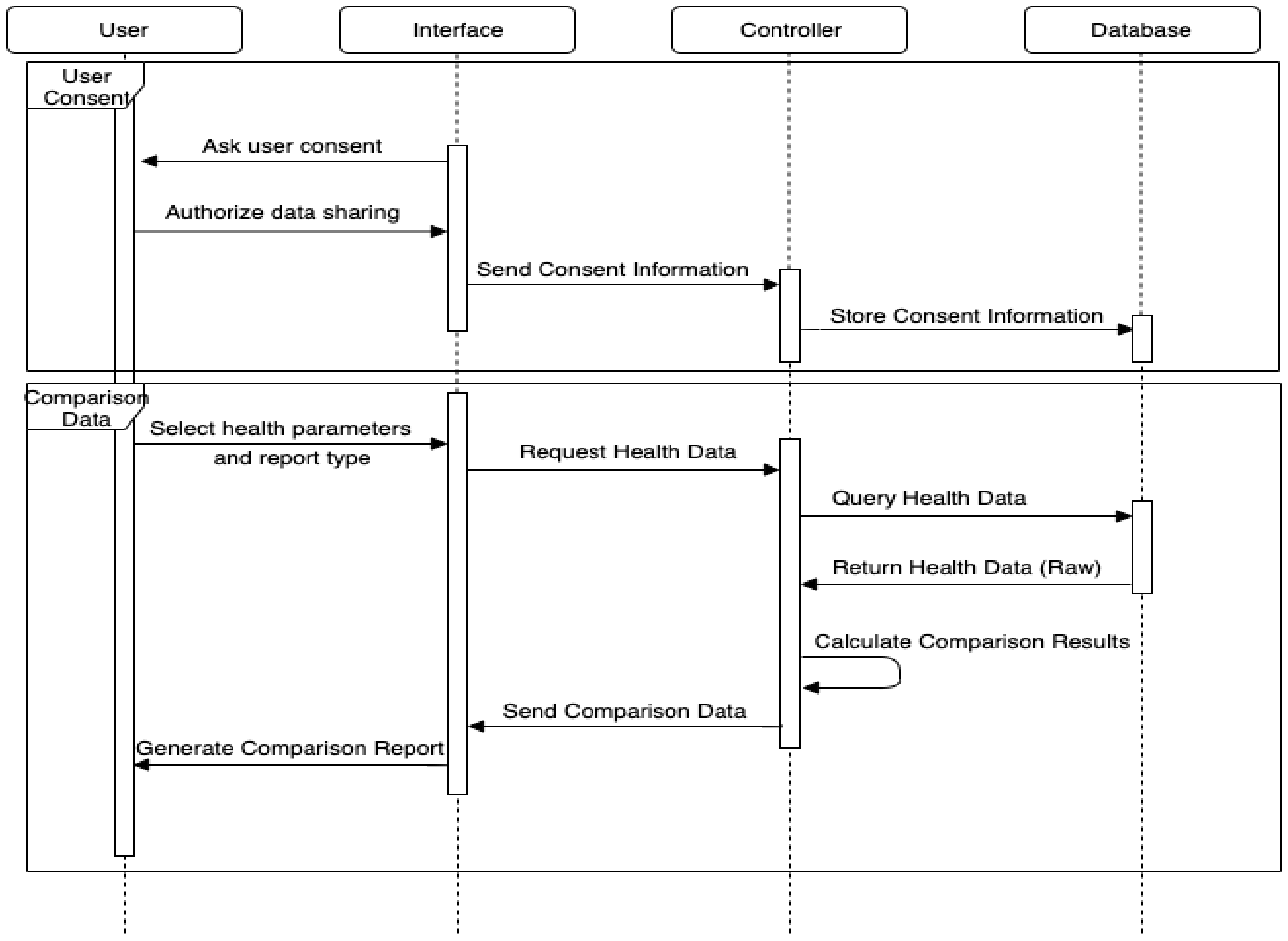
II. UC-3: User Data Addition



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.

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.

2. Design Principles:

The design principles we used to assign responsibilities to objects are included in GRASP (General Responsibility Assignment Software Patterns (or Principles). The controller is defined as the first object beyond the UI layer that receives and coordinates ("controls") a system operation. So the controller module was assigned responsibilities which dealt with coordination and control processes in the system e.g. Assigning responsibility R22 (Select Recommendation to User) to the controller. Using a different module or combining more than one module for R22 would have increased the number of steps as the responsibility based on controller pattern should be included in the controller module for efficiency.

Another design principle we used for assigning responsibilities is high cohesion. High cohesion means that the responsibilities of a given element are strongly related and highly focused. For example while assigning responsibilities to interface module, responsibilities R1, R2, R3 and R4 are closely related as they deal with the display of information on the screen. The responsibilities are highly related and therefore assigning these responsibilities to the interface module is feasible. If closely related responsibilities are assigned to different modules such as R1(Display data and user information) which is included in interface module and R2(Display forms for user interaction) then the design of the system will be flawed as there will be a waste of resources as two different modules will be called to do similar actions. In addition, UI could also take the responsibilities of the TextReader and the checker, yet since it has already taken R1, R2,R3 and R4, adding the reader's responsibilities would have to lower the cohesion of the UI and that is why we have set up the checker and TextReader modules.

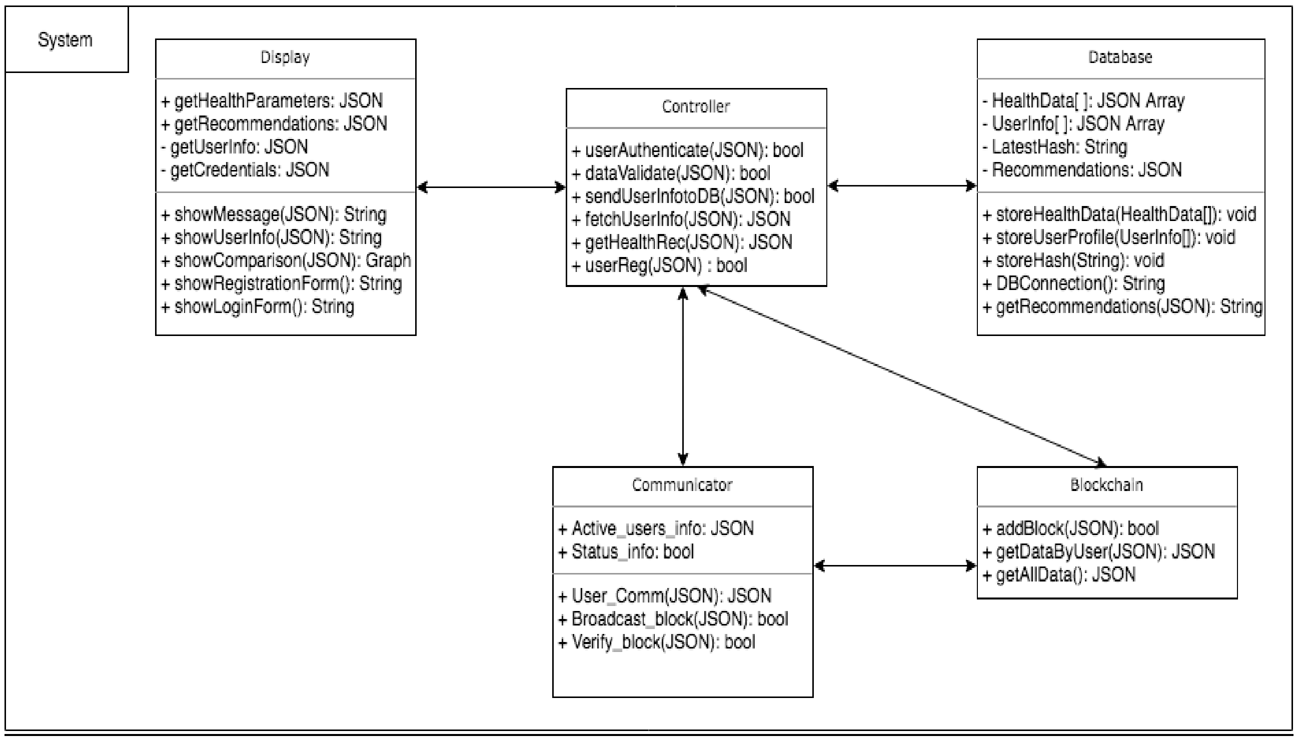
Regarding that there are so many responsibilities that the controller needs to fulfill, while there are still more responsibilities we have to reach out. As the communication responsibilities and data storage responsibilities pop up, we must set the communicator modules and the DB module.

Responsibilities are assigned to different modules depending on the above two design principles. Due to this fulfillment of a responsibility often requires information that is spread across different modules. This implies that there are many "partial experts" who will collaborate in the task. This is included under Information Expert (Expert) design principle.

SECTION 2

3. Class Diagram and Interface Specification

3.1 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

3.2 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.

3.3. 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

4. System Architecture and System Design

4.1. Architectural Style

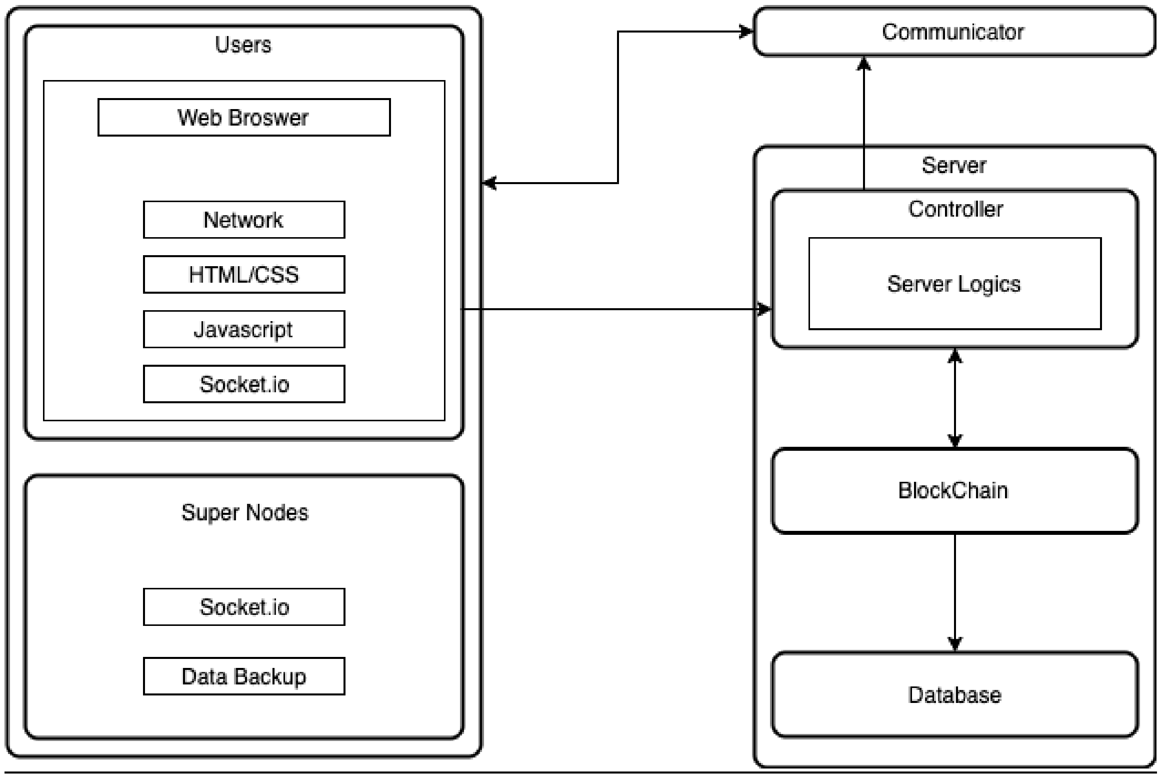
Our system will not follow the architectural model of common existing blockchain practices. To be fair, bitcoin-like architecture is kind of silly and has a lot of defects. Ideally, every user will store the transaction information from all the other users. But terabytes of that information are still 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.

4.2 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.

4.3 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.

4.4 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.

4.5 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.

4.6 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.

4.7 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.

SECTION 3:

5. Design of Tests

5.1 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

6. Project Management and Plan of Work

6.1 Project Coordination and Progress Report

Use cases that are completed so far are:

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.

Use cases being currently developed are:

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

Use cases to be attempted are:

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.

6.2 Plan of Work

In the first phase of our project, we are going to implement the basic blockchain-based health system. This phase is meant to form the most basic and essential functions of our system: adding data block to the main and local block-chains, communicating with the database and simple user interface. We will add three sample users to the network. One of the three users can add their data to the blockchain, and others can verify this data. If any outliers exist in the data entered by the user, the user will not be allowed access to the system. The data validator of the system will 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 are using the JavaScript and MongoDB, for the creation of the blockchain and backend part, where we can save the user's data and share it among other users. We will be finishing this phase by October 29th, 2019.

A super node is made available within this network, which is used to store the blockchain data of all the users and serves as the database for the users. This will be a relational database using SQL.

In the second phase of the project, we are going to finish the user interface to facilitate users to register and login to our system. While registering with the system, the user is given an option to share his personal data with other users and only if he consents, the data is broadcast to the other users who can compare the data to verify it. Once the user logs into the system, his credentials are saved onto the system and he can update his health data. If the user adds new data, all the other users on the network will be notified. Users can also generate historical reports, which gives details of his past health data and they can also compare their data with the data of the other users. The second phase will include provisions for the data analysis that will be done on the data stored in the blockchain. Necessary upgrades will be made to the user interface. We will be using open-source library like Electron to create this sort of desktop application.

This phase will be completed by November 20th, 2019. By this time we will have the application that will be able to perform most main functionalities that are desired from the application.

In the third phase, we will be addressing the scalability issues related to the network we have created. In other words, we will build the system to accommodate many users on the network. We will also provide recommendations to users based on their present health conditions. Each metric will have its own priority depending upon the seriousness of the health conditions. For example: heart rate and blood pressure will have importance compared to sleep patterns. A forum for the users to communicate with other users will also be created during this stage.

If the user is facing any problems with logging into the system, a support system will be used to address the issues. A support system also addresses issues related to login, data updating, health parameter comparison etc.

6.3 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 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
- Swapnil Kamate: User interface, documentation
- Justin Wei: Documentation, blockchain infrastructure.

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

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

7. References

I. *Software Engineering I Lecture Slides*

II. *Fitbit Health Monitoring Analytics*
https://www.ece.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2014-g5-report3.pdf

III. *GRASP (General Responsibility Assignment Software Patterns)*
https://whatis.techtarget.com/definition/GRASP-General-Responsibility-Assignment-Software-Patterns

IV. *Cohesion_wikipedia*
https://en.wikipedia.org/wiki/Cohesion_(computer_science)

V. *Monitoring the health of web page analytics code*
https://patents.google.com/patent/US20110035486