

Software Engineering (14:332:452)

Group 3

**Programming Project: Blockchain-Based Safe Sharing of Population
Descriptors**

Submission Date: November 14, 2019

**Team Members: Sasan Hakimzadeh, Pradyumna Rao, Nithyasree Natarajan,
Jack Dulin, Shruthi Sureshkrishnan, Nithya Kandappan, Breanna Higgins,
Sean Kearns**

Table of Contents

Responsibility Matrix	3
Section 1: Diagrams	5
Fully Dressed Use Cases	5
b. System Sequence Diagrams	7
c. Interaction Diagrams	10
Section 2: Class Diagram and Interface Specification	13
Class Diagram	13
b. Data Types and Operation Signatures	14
c. Traceability Matrix	18
Section 3: System Architecture and System Design	19
Architectural Styles	19
Identifying Subsystems	20
Mapping Subsystems to Hardware	22
Persistent Data Storage	22
Network Protocol	22
Global Control Flow	23
Hardware Requirement	23
Section 4: Data Structures	26
Section 5: User Interface Design and Implementation	27
Section 6: Design of Tests	34
Unit Testing	34
Test Coverage	34
Integration Testing	34
Other Testing	35
Section 7: Project Management and Plan of Work	36
Issues Encountered	36
Project Coordination and Progress Report	36
Plan of Work	36
Breakdown of Responsibilities	36
References	37

Responsibility Matrix

[illegible]

Unit Testing	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Test Coverage	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Integration Testing	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Other Testing	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Issues Encountered	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Project Coordination and Progress Report	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Plan of Work	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Breakdown of Responsibilities	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
References	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%

All team members contributed equally.

Section 1: Diagrams

a. Fully Dressed Use Cases

UC-2	Data Input/Validation
Related Requirements	REQ-7, REQ-1, REQ-12, REQ-13, REQ-14, REQ-15, REQ-18
Initiating Actor	User, Database Management System (DBMS)
Participating Actors	Database Management System (DBMS)
Preconditions	The user is logged into the system.
Postconditions	There has been a successful or unsuccessful attempt of adding more data into the system.
Event Flow (Success)	<ol style="list-style-type: none">1. The user has tried to enter a reasonable value.2. The data point is entered into the system.
Event Flow (Failure)	<ol style="list-style-type: none">1. The user has entered an unreasonable value (e.g. it is above a certain threshold).2. The data point is rejected from the system.

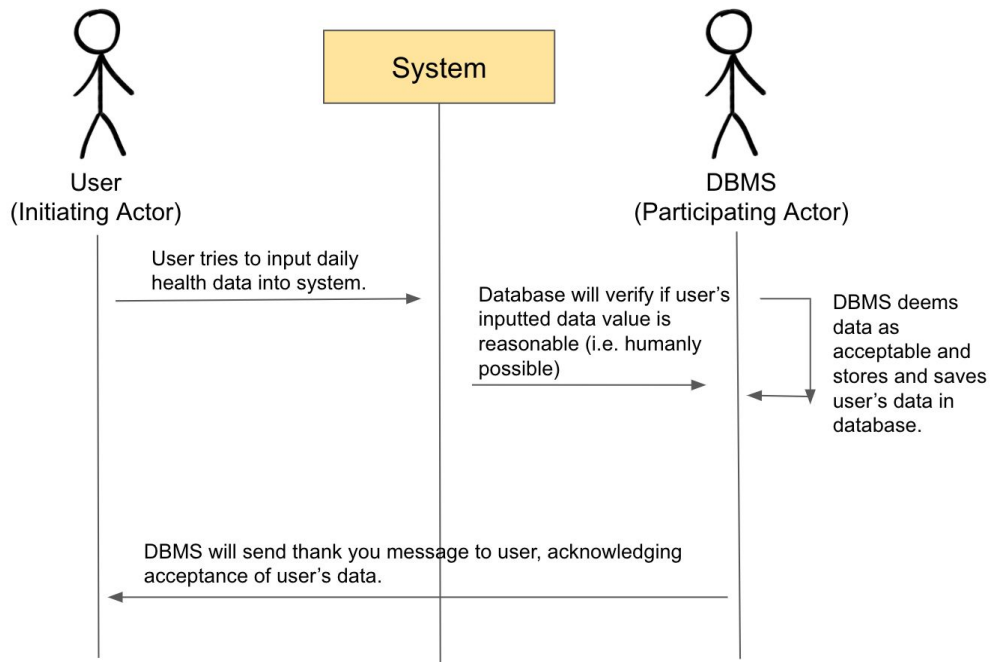
UC-4	View Visualizations
Related Requirements	REQ-6, REQ-20, REQ-5, REQ-12, REQ-14, REQ-18
Initiating Actor	User
Participating Actors	Data Visualization Software, DBMS
Preconditions	The user is logged into the system.
Postconditions	The user sees a collection of their data in the form of visuals.
Event Flow (Success)	<ol style="list-style-type: none">1. The user requests a visualization of their inputted data.

	<ol style="list-style-type: none"> The data visualization software shows the user their data in a presentable way, whether through a pie chart or a bar graph.
Event Flow (Failure)	<ol style="list-style-type: none"> The user requests a visual representation of their inputted data. The data visualization software is unable to accomodate the user's request.

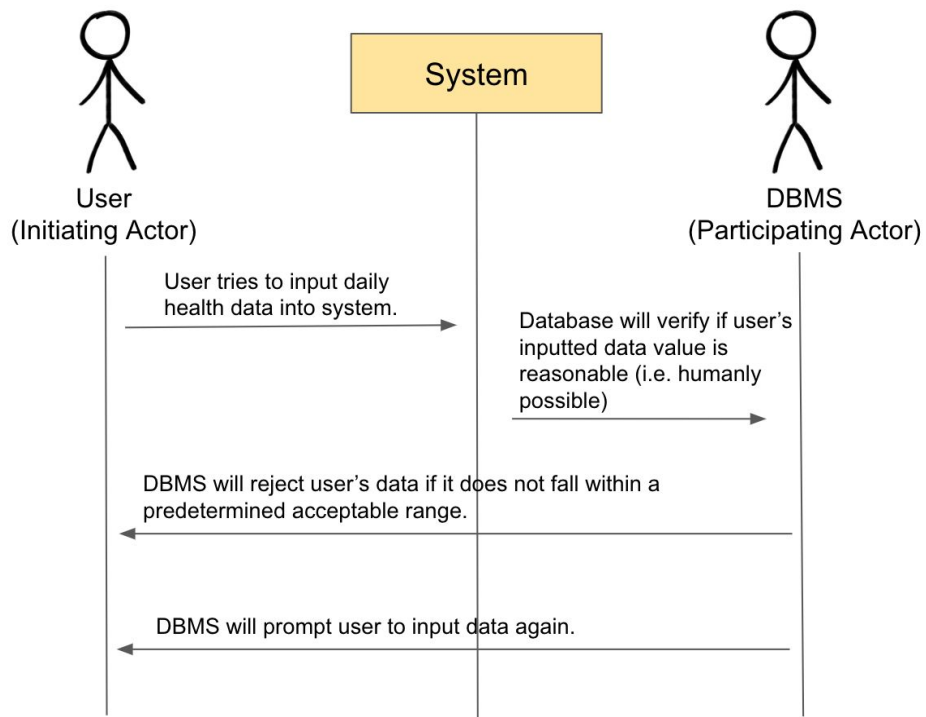
UC-8	Log In
Related Requirements	REQ-21, REQ-1, REQ-3, REQ-4, REQ-15
Initiating Actor	User
Participating Actors	Database Management System (DBMS)
Preconditions	The user is currently at the screen in which they are asked to enter his/her username and password.
Postconditions	There has been an attempt to get into the system through a correct or an incorrect username and password.
Event Flow (Success)	<ol style="list-style-type: none"> The user types in the correct username and password. The user is taken to the home page of our application.
Event Flow (Failure)	<ol style="list-style-type: none"> The user types in the incorrect username and/or password. They have another opportunity to type in the correct username and password. They are asked if they have forgotten their username or password.

b. System Sequence Diagrams

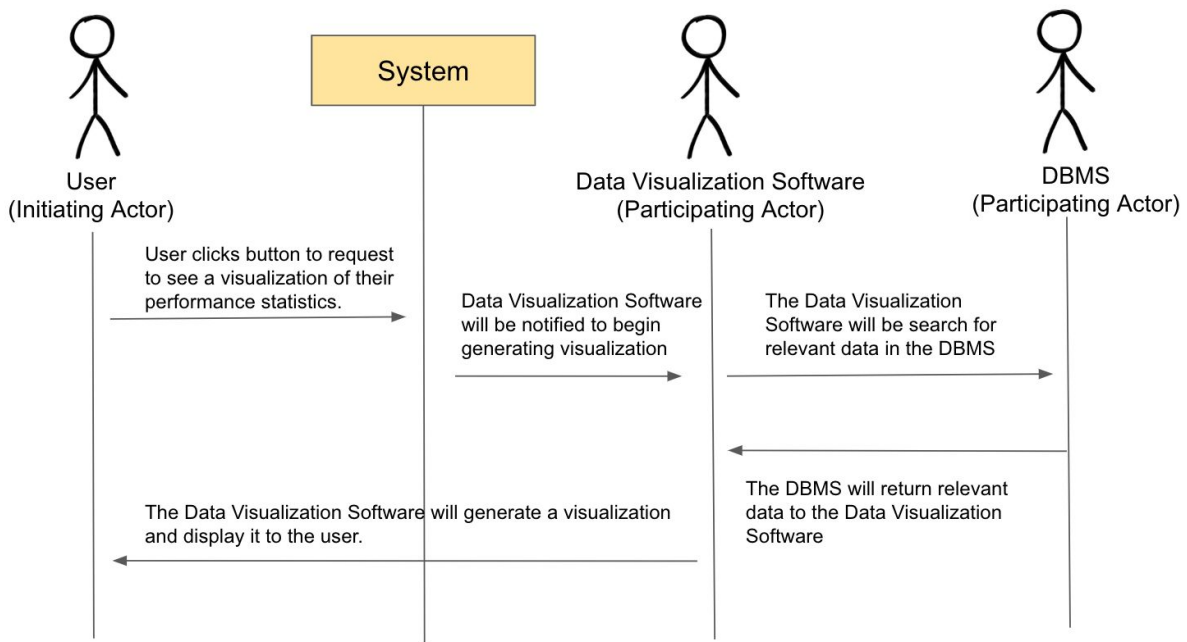
Use Case 2: Data Input / Validation (Success)



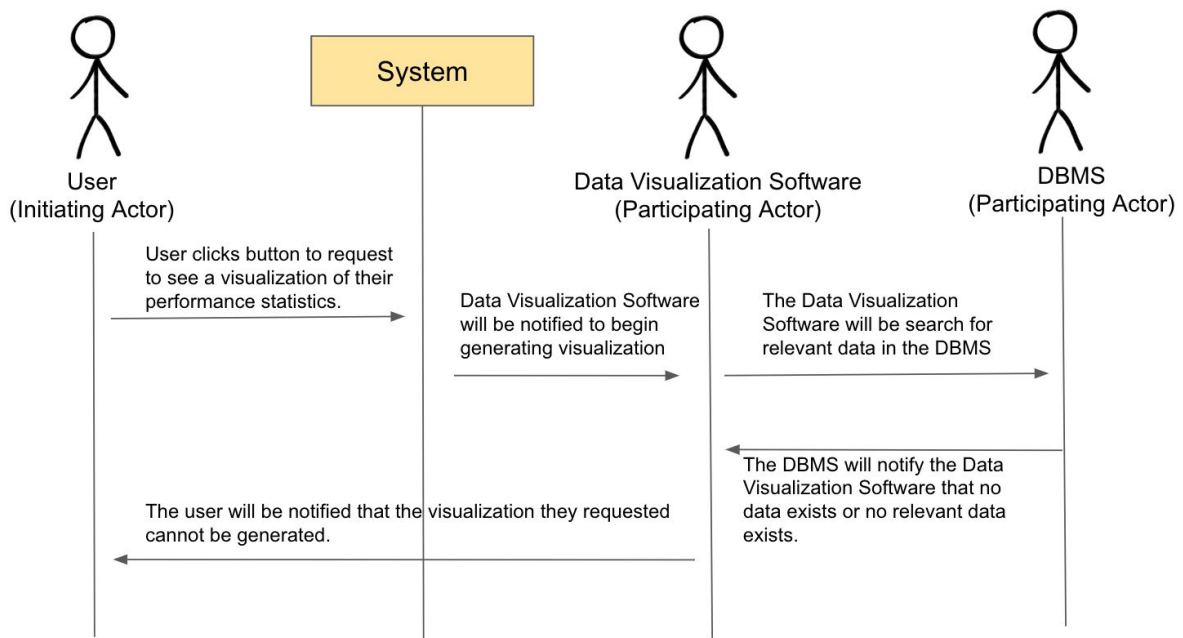
Use Case 2: Data Input / Validation (Failure)



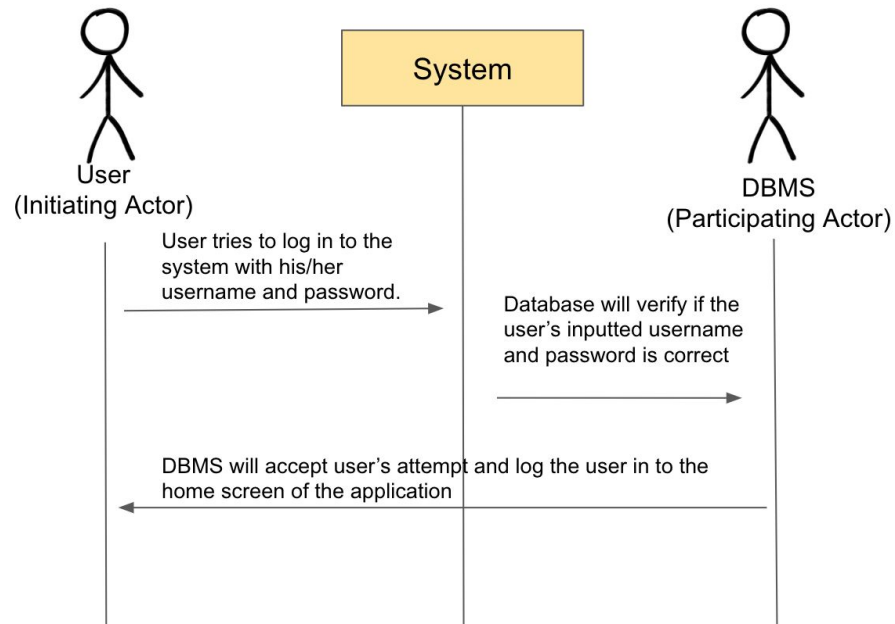
Use Case 4: View Visualizations (Success)



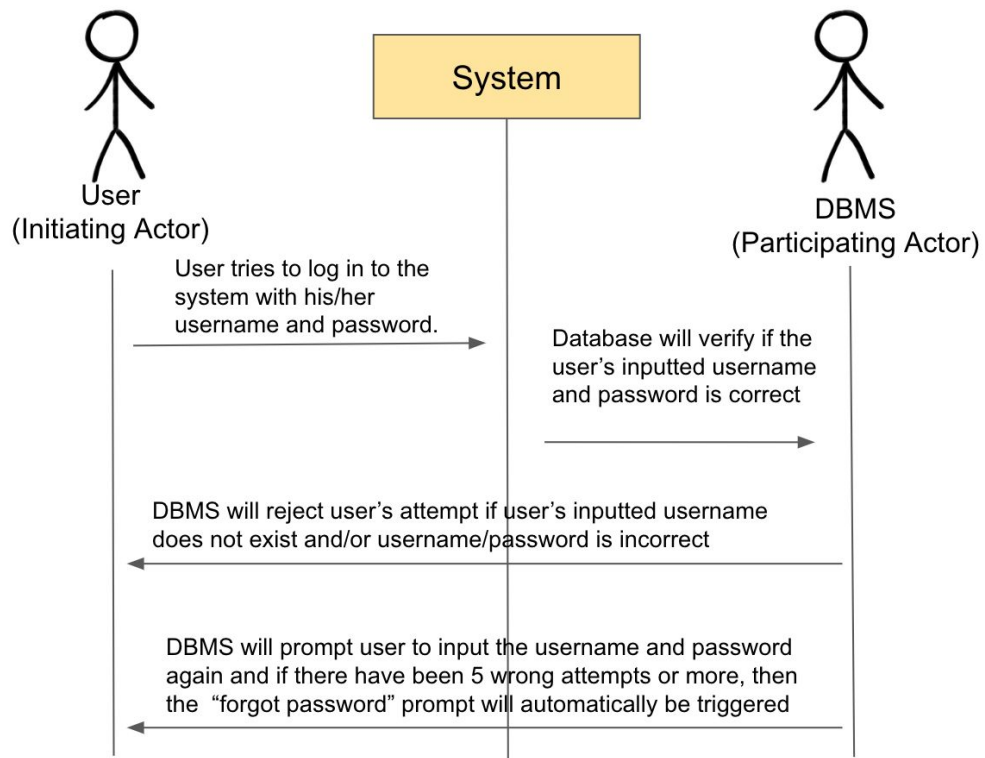
Use Case 4: View Visualizations (Failure)



Use Case 8: Log In (Success)



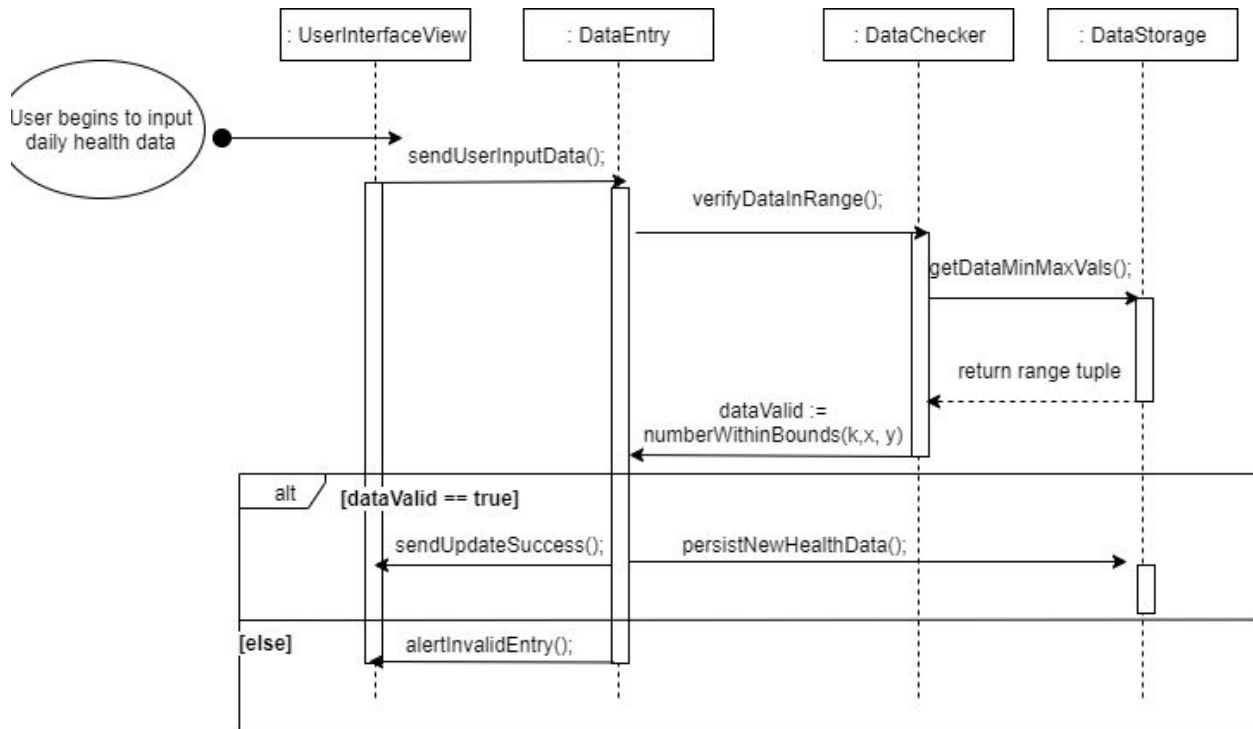
Use Case 8: Log In (Failure)



c. Interaction Diagrams

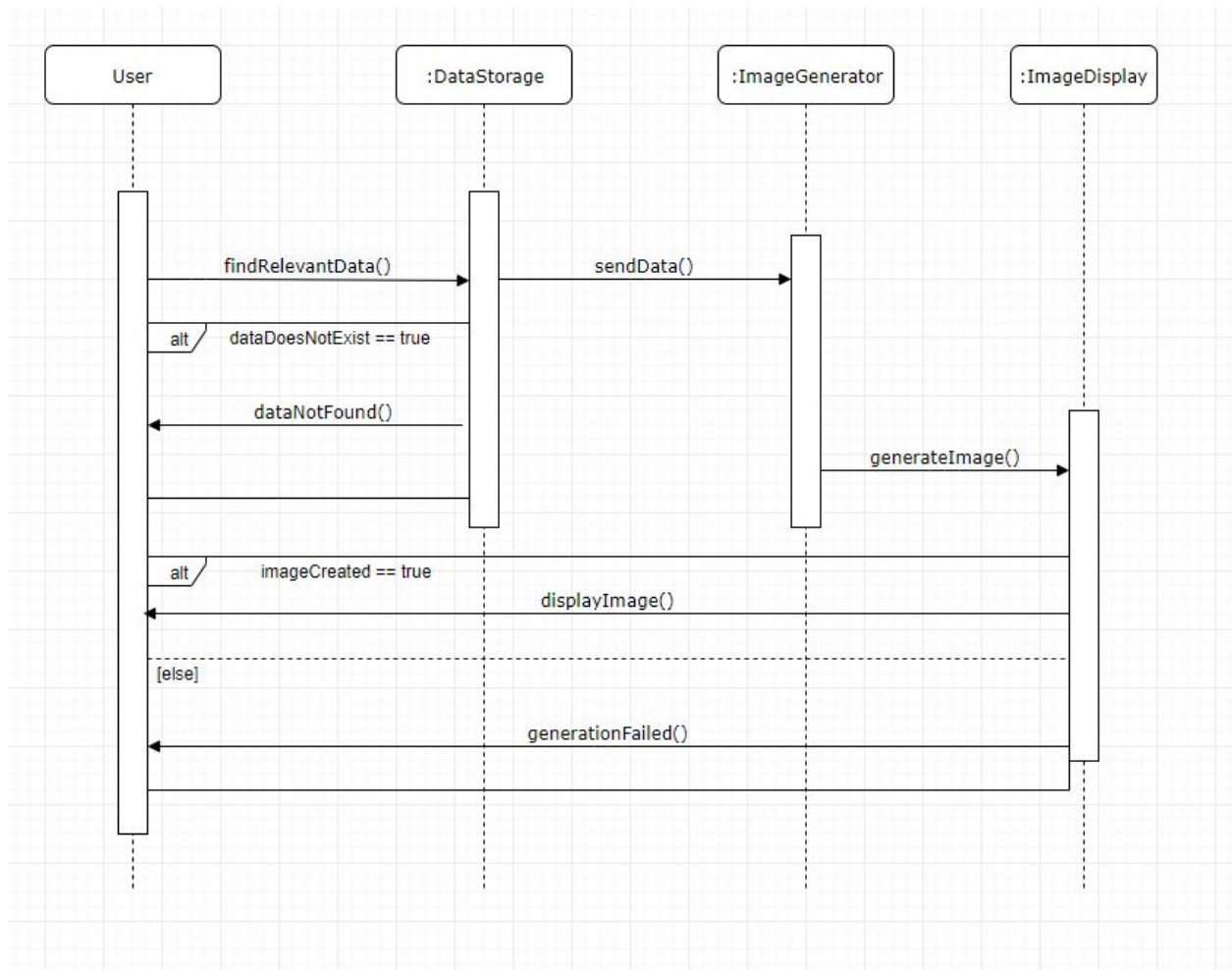
The following sections describe the main use cases of our application in terms of interaction diagrams.

Use Case 2:



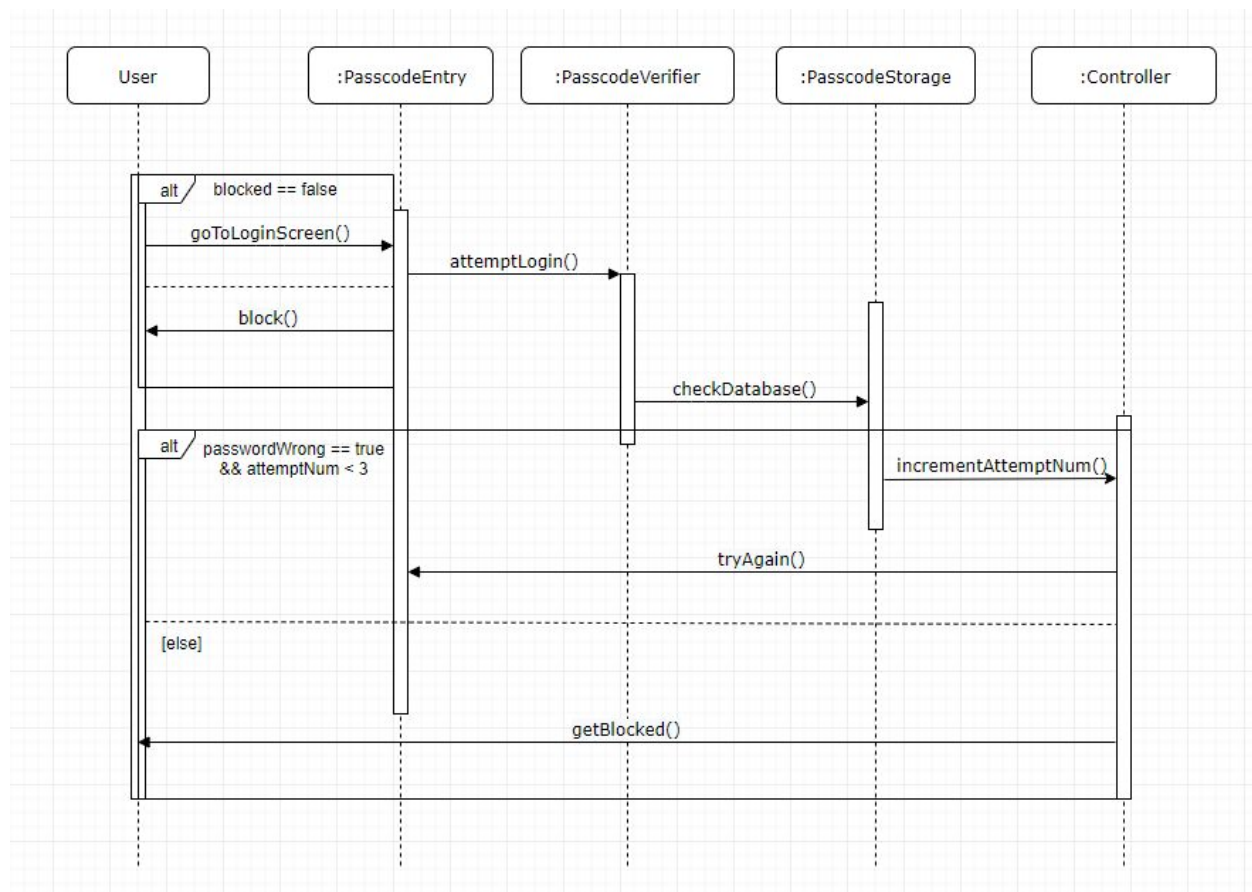
The above interaction diagram is derived from the system sequence diagrams to show the interaction between objects within the system. The design principle guiding this diagram was the main single functionality, low-coupling, for the responsibilities of each module.

Use Case 4:



The interaction diagram was motivated by the system sequence diagram. For each object, the design emphasized keeping the responsibilities (or number of modules) of a single object to a minimum to avoid too much dependency on one object in order to make future refactoring easier.

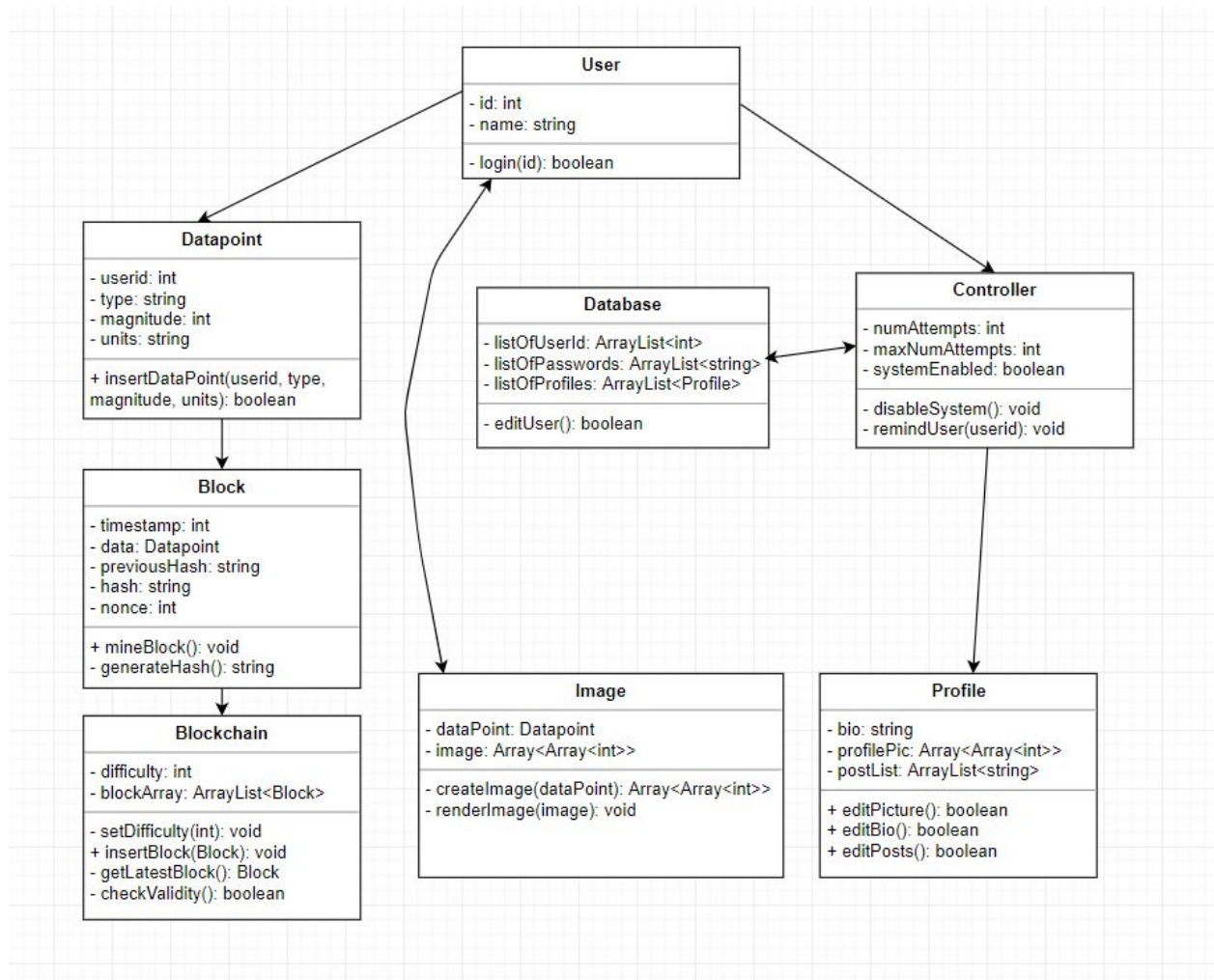
Use Case 8:



Like the other interaction diagrams, this tool was developed utilizing the system sequence diagrams from Report #1. There is a limited amount of functionality for each object, which results in a low amount of coupling. While this might take more effort upfront and require more modules, this will save effort in the long term because refactoring will be streamlined due to fewer dependencies on other objects.

Section 2: Class Diagram and Interface Specification

a. Class Diagram



b. Data Types and Operation Signatures

Profile Class:

Attributes:

-bio: string

- This is a string variable that refers to the bio of the user.

-profilePic: Array<Array<int>>

- Since pictures are stored using 2D arrays of integers that represent rgb values, the profile picture of the user is stored in an array of array of integers.

-postList: ArrayList<string>

- Since each post consists of a string, the complete list of posts that a user has is represented by an ArrayList of strings. It's stored in an ArrayList instead of an array so that the list of posts is mutable, making it easier to alter the list of posts that a user has.

Operations:

+editPicture(): boolean

- This is a boolean function that allows the user to upload or delete a profile picture. Upon failure, this function returns false. This could happen when the user uploads something that is not a picture.

+editBio(): boolean

- This is a boolean function that allows a user to upload a new bio, edit a pre-existing bio, or delete their bio. This function fails and returns false when the user tries to upload a bio that exceeds the character limit.

+editPosts(): boolean

- This is a boolean function that allows a user to upload a new post, edit a pre-existing post, or delete any post of their choosing. This function fails and returns false when the user tries to upload a post that exceeds the character limit.

Datapoint Class:

Attributes:

-userid: int

- This int parameter keeps track of the userid that this particular data point is linked to.

-type: string

- This string parameter tells you what type of parameter has been inserted (e.g. hours of sleep, miles walked, push-ups performed, etc.).

-magnitude: int

- This int parameter tells you how much of a certain action was performed (e.g. you slept for 8 hours last night).

-units: string

- This string parameter gives you the units of the particular data point you just entered (e.g. miles, hours).

Operations:

+ insertDataPoint(userid, type, magnitude, units): boolean

- This boolean method inserts the data point into the blockchain and returns true if the data point consists of a valid userid, the type is within a predetermined set of values, the magnitude is within

a reasonable range, and the units are acceptable. Upon failure of any of these conditions, it returns false.

Database Class:

Attributes:

-listOfUserId: ArrayList<int>

- This attribute stores the list of userid's in the system.

-listOfPasswords: ArrayList<string>

- This attribute stores the list of passwords in the system.

-listOfProfiles: ArrayList<Profile>

- This attribute stores the list of profiles in the system.

Operations:

-editUser(userid): boolean

- This boolean method alters the profile data linked to a particular userid, whether it is adding a new user or deleting a pre-existing user from the database and it returns true upon success. Otherwise, it will fail and return false.

Controller Class:

Attributes:

-numAttempts: int

- This int attribute gives the current number of failed consecutive attempts to get into the system.

-maxNumAttempts: int

- This int attribute gives the maximum amount of consecutive failed attempts that are allowed before the system is no longer enabled and the user has to change their password.

-systemEnabled: boolean

- This boolean attribute tells you whether the system is enabled or not.

Operations:

disableSystem(): void

- This method disables the system after an incorrect password has been entered three consecutive times in a row.

remindUser(userid): void

- If the user hasn't put in any data that day, this method will send a message to the user reminding them to input their data for that day.

Image Class:

Attributes:

-dataPoint: Datapoint

- This attribute gives the image class the information it needs to generate an image based on the data point that has been entered by the user.

-image: Array<Array<int>>>

- This attribute stores the image of the user's progress that will be shown to the user.

Operations:

+createImage(dataPoint): Array<Array<int>>>

- This method creates an image (a 2D array of integers that represent rgb values) using the data point that has been given as a parameter.

+renderImage(image): void

- This method shows the image stored in the image parameter to the user.

User class:

Attributes:

-id: int

- This attribute stored the userid associated with this user.

-name: string

- This attribute stores the name of the user.

Operations:

+login(userid): boolean

- This method logs the user into the system. This will return false if the user entered an incorrect password.

Block class:

Attributes:

-timestamp: int

- This int attribute tells you how many milliseconds have passed since January 1, 1970 up until the creation of this block.

-data: Datapoint

- This attribute is an instance of the Datapoint class that details the type of data that is within this particular block.

-previousHash: string

- This string attribute tells you the hash of the previous block that points to this current block. This is used to ensure that the blockchain retains its validity.

-hash: string

- This string attribute is a calculated value that depends on the value of every other attribute. If any of the attributes get tampered even slightly, there will be a dramatic change in this value and it will render the blockchain invalid.

-nonce: int

- This int attribute is used as a noise value that aids in the process of mining a block until it contains a hash that is prepended by a certain number of zeros. This number of required zeros is determined by the difficulty attribute in the Blockchain class.

Operations:

+mineBlock(): void

- This void method tries to find a hashcode that has a sufficient number of zeros prepended to it. Once it does that, the new block can now become a part of the blockchain.

-generateHash(): string

- Using all of the other data found within the block, this method generates a hashcode for this current block.

Blockchain class:

Attributes:

-difficulty: int

- This determines how many zeros should be at the beginning of a hash in order for it to be accepted as secure enough for secure use.

-blockArray: ArrayList<Block>

- This attribute is the current list of the blocks that are currently in the blockchain.

Operations:

-setDifficulty(int): void

- This method takes an int parameter that determines how many zeros a hash should have in the beginning in order for it to be deemed as secure enough for general use.

+insertBlock(Block): void

- This method inserts the block into the blockchain. There is no need for this method to be a boolean because the verification of the data occurred before the data enters the blockchain.

-getLatestBlock(): Block

- This method returns the block that has been placed into the blockchain the most recently.

-checkValidity(): boolean

- This boolean method tells the caller whether the blockchain is still valid or not. If a rogue agent has attempted to modify the blockchain, this method will return false.

c. Traceability Matrix

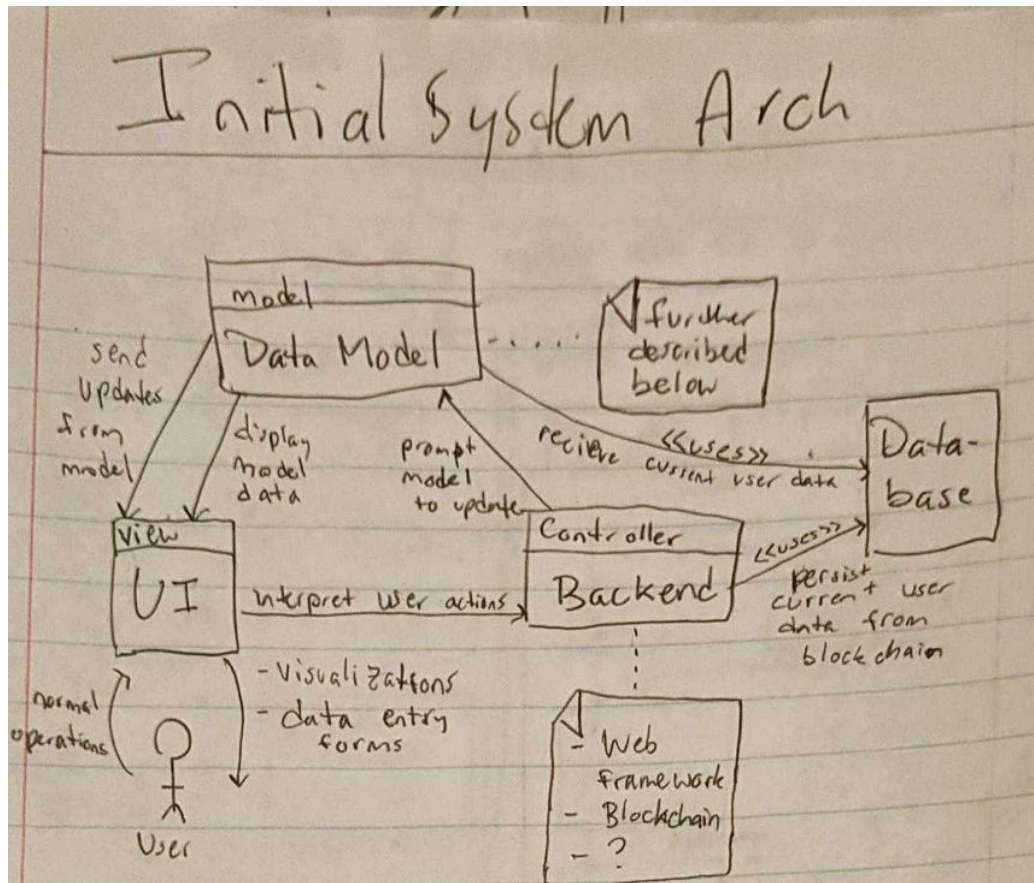
		Classes							
	Class	Profile	Datapoint	Database	System	Image	User	Block	Blockchain
Domain Concepts	Profile	x							
	ProfileChanger	x					x		
	ProfileStorage			x					
	DataEntry		x				x	x	x
	DataChecker		x						
	DataStorage		x				x	x	x
	TaskChecker				x				
	Reminder				x				
	ImageGenerator					x			
	ImageDisplay					x			
	EmailNotifier				x				
	PasswordChanger			x			x		
	GoalSetter			x			x		
	GoalStorage			x			x		
	ModifyFriends			x			x		
	ShareProgress						x		
	PasscodeEntry			x	x		x		
	PasscodeVerifier			x	x				
	PasscodeStorage			x					
	Controller				x				

Section 3: System Architecture and System Design

a. Architectural Styles

The architectural style of our application is mainly classified as a Multitier architecture in which the presentation of our application is separated from the actual processing component as well as the data management component. The presentation layer, or the UI/view layer is what the user will see and deal with. When users log in, view their personal data, view general population metrics, update their profiles, etc. it will all be done on this layer. This layer is largely built using Javascript/HTML to allow easy navigation throughout the application. The processing component of our application is the blockchain which securely stores user information/data metrics in blocks. Each user entry is a block which is associated to the respective user, and others will not be permitted to view anyone else's personal information or data. This layer is also built using javascript. The data management component is where all user profiles and data points are stored. For example, user credentials and personal information such as date of birth, gender, weight, etc. will be stored here. Our database management system is based on a SQL server which will be performing this storage.

b. Identifying Subsystems



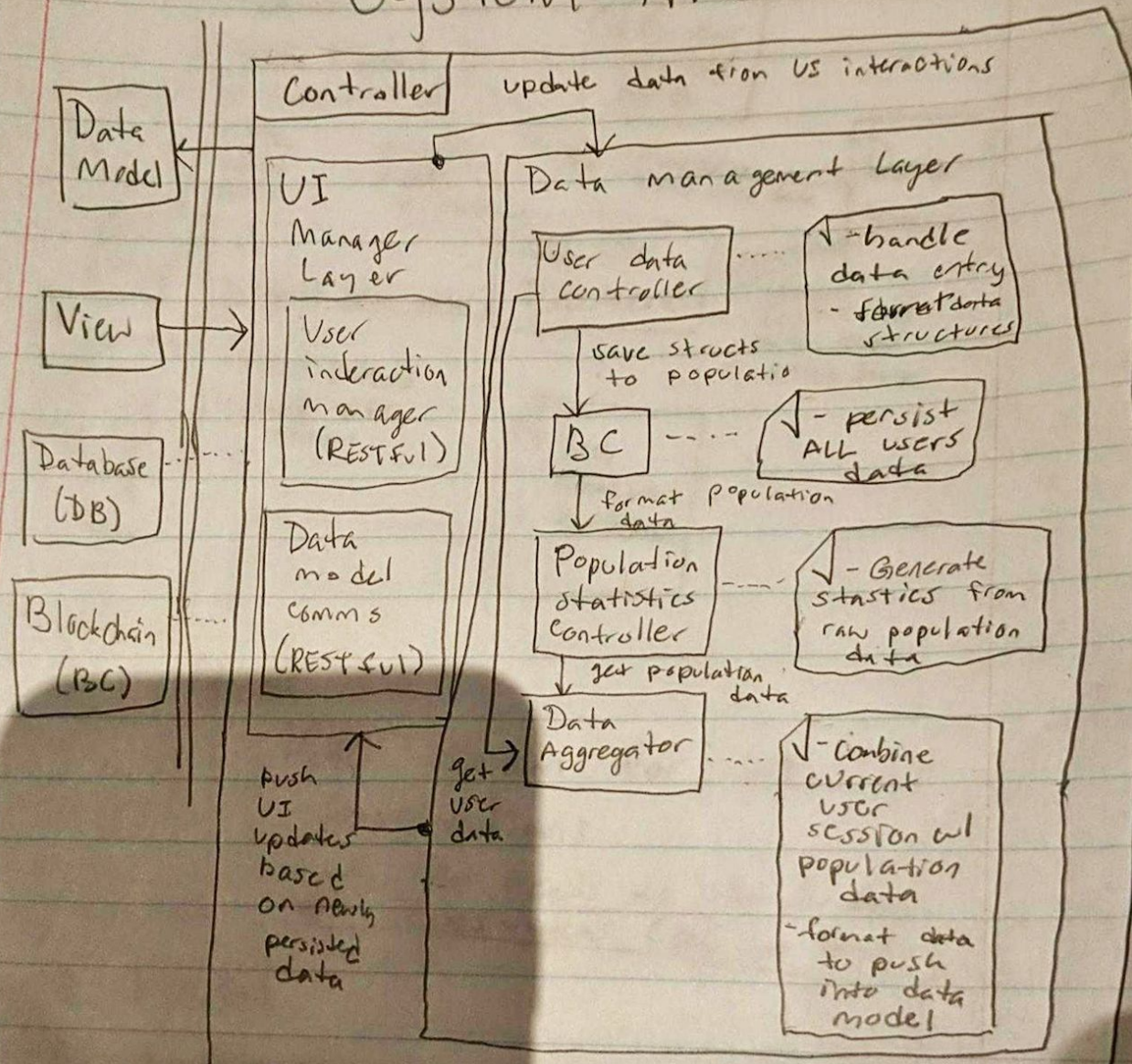
Notes:

- MVC system for UI interactions
- Controller needs to be broken into subsystems (consider layered arch)

Data Model

- User preferences
 - eg: active devices, reminder schedule
- User device data
 - eg: steps walked, weight, blood pressure
- ~~Generated~~ Generated Population statistics

Initial Controller System Arch



c. Mapping Subsystems to Hardware

Because our application is based on blockchain and the whole motivation behind it is for multi user functionality, it is correct that our system should run on multiple computers. One user who submits his/her metrics for that day should be taken into account on another user's computer in regards to the public population metric. For example, if a 30 year old female enters 2.6 miles walked that day, then this value should be taken into account for the "30-35 Year Old Female" bucket in regards to miles walked. This value should be added and averaged with all the other values in this field. Thus, when another user (let's say a 31 year old woman) accesses her data and would like to see population metrics, the value from the first user should have an effect on the average and thus display a new value which takes into account her value. Our client is currently a web browser which the user can access, and it is being hosted on a WAMP server. The web browser can be accessed from any computer, while the WAMP server is hosted independently and externally.

d. Persistent Data Storage

The application has two types of storage requirements which drive the design of the two persistent data storage methods implemented. The program needs to store the fitness data of each user, which must persist when the user closes the application. This data storage will be done through a blockchain system so that the data is secure. The program must also store common app data and user session information. This is performed in a common Relational Database System (RDBS). Through this dual-implement system, the application accesses the secure blockchain for sensitive health data but also maintains well-organized and easily queryable databases for user session data.

e. Network Protocol

The app is to be hosted on a website capable of performing user interactions client-side and handling data storage needs server-side. For client-to-server communications, HTTP is used to serve the site files to the user's web browser from the central server. For client-side behavior, multiple javascript files must be handled within the system. The communication between these multiple files is performed in a Single Page Application Router which uses REST APIs to load HTML and js data from multiple pages on a single page.

f. Global Control Flow

For interactions within the application, the execution orderliness of the app is event-based. All actions within the app, such as adding data, viewing statistical visualizations, modifying settings, or managing goals, must be initiated by clicking the button that brings that view to the main window. Retrieval of data to display a page is performed by the initiation brought forth by the button click. Further actions such as data validation or data persistence is once again, triggered by user interaction.

With regards to time dependency, there is a timer in the system that would alert the app user to input their data at the user's preferred time selected every night if they have not already done so.

g. Hardware Requirement

The application was designed to be cross-platform. As such, any device capable of accessing a web browser should be able to run the app. There are no resolution requirements as the bootstrap-framework-based UI is built for responsive design. In order to run the app, the browser used must be able to run JavaScript.

PART 3:

4. Algorithms and Data Structures (if applicable)

1. Algorithms

Describe the algorithms that implement mathematical models from your Report #1. Does your system use any other complex algorithms? For example, when computing a motion trajectory for an animate figure in a game, you may use some numerical or computer-graphics algorithms. Or, when assessing stock market movements, you may be using statistical algorithms.

If NO, skip to the next item;

If YES, describe your algorithms. For example, for the animate figure example above, will the path coordinates be precomputed and stored in a look-up table or will they be computed using a spline interpolation algorithm.

It is a good idea to use *activity diagrams* to describe the algorithm design.

2. Data Structures

Does your system use any complex data structures, such as arrays, linked lists, hash tables, or trees?

If NO, skip to the next item;

If YES, what criteria you used in deciding what data structure to use, e.g., performance vs. flexibility?

5. User Interface Design and Implementation

1. Describe whether and how you modified and *implemented* the initial screen mock-ups developed for [Report #1](#). Comment only on significant changes in your user interface, those that reduce (or increase) the [user effort](#). Changes of colors or styles are less important and should be omitted from your report.

2. The textbook does not deal much with the GUI design. Excellent guidelines for GUI design can be found here:

Sun Microsystems, Inc. *Java Look and Feel Design Guidelines*. Mountain View, CA, 1999. Available at: <http://java.sun.com/products/jlf/ed2/book/>

3. “Ease-of-use” is generally considered a key characteristic of user interface. “Ease-of-use” should not be confused with a flashy interface, with lots of colors, picture, graphics, etc. On the contrary, you should avoid flashy user interfaces. “Ease-of-use” means that interface is intuitive, easy to understand and operate, without having to ask many questions or read voluminous documentation. A minimal user interface that is well organized should be sufficient. You already considered the user effort as part of [Report #1](#), and here you should strive to minimize the user effort, thus maximizing the “ease-of-use”.

6. Design of Tests

Note that for this report you are just *designing* your tests; you will *program and run* those tests as part of work for your first demo, [see the list here](#).

1. List and describe the test cases that will be programmed and used for unit testing of your software.
2. Discuss the test coverage of your tests.
3. Describe your *Integration Testing* strategy and plans on how you will conduct it.
4. Describe also your plans for testing any algorithms, non-functional requirements, or user interface requirements that you might have stated in your Report #1.

7. Project Management and Plan of Work

1. Merging the Contributions from Individual Team Members

Compiling the final copy of the report from everyone's work, ensuring consistency, uniform formatting and appearance.

Describe what issues were encountered and how they were tackled.

2. Project Coordination and Progress Report

What use cases have been implemented?

What is already functional, what is currently being tackled?

List and describe other relevant [project management](#) activities.

3. Plan of Work

List the projected milestones and dates by which you plan to accomplish them.

Preferably, you should use [Gantt charts](#) for planning and scheduling your project.

4. Breakdown of Responsibilities

- List the names of modules and classes that each team member is currently responsible for developing, coding, and testing
- Who will coordinate the integration?
- Who will perform and integration testing? (The assumption is that the unit testing will be done for each unit by the student who developed that unit.)

8. References

The list of references should contain exact *references and URLs* of any material that is used in the project and does not come from the textbook.

Section 4: Data Structures

Arrays are being used to store the blockchain structure, and the main reasons an array was chosen was for simplicity, difficulty of removal, and ease of access of each element. Using arrays instead of linked lists makes it so that an element can be added to the chain by simply using the `push()` operation in JavaScript instead of manipulating pointers and having to possibly traverse the entire list in the case of linked lists. Also, a blockchain should be designed such that elements are difficult to remove after placing them into the blockchain. An array would be a better choice than a linked list because in a linked list, removal is constant time when you have access to the element you want to remove. However, arrays have $O(N)$ removal time if there are N elements in the array (or in this case, N blocks in the chain). Additionally, you would potentially have to shift all of the elements in the array instead of manipulating a few pointers like you would after removing a node from a linked list. Therefore, it would be more computationally expensive to remove a block in an array-based blockchain than it would be in a blockchain stored in a linked list, which is the desired behavior of a blockchain structure. Lastly, ease of access for certain elements is required in some instances. For example, we will need to access the last block in the chain to obtain the latest block that has been added to our blockchain. In a linked list, we could potentially use a dummy node to keep track of where the last element is in a linked list, but that would take up additional memory that wouldn't be necessary in an array. We will also need to sequentially access elements in our blockchain during the validation phase, and using an array facilitates this process.

Section 5: User Interface Design and Implementation

Use Case 2: Data Input/ Validation

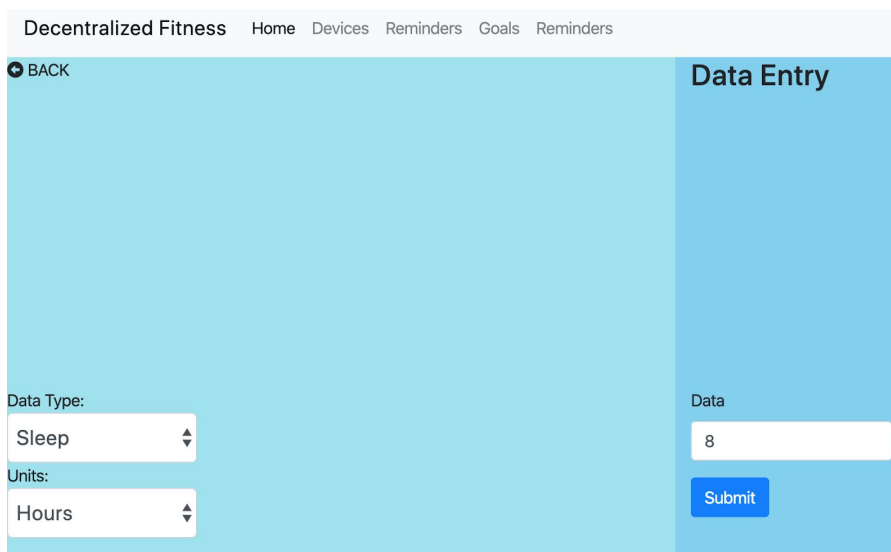
Initial Mockup

Data Entry Page



The initial mockup of the Data Entry page is divided into two main sections. The left section, with a dark teal background, contains input fields for 'Weight' and 'Blood Pressure', and an 'Add Data' button. The right section, with a light blue background, contains input fields for 'Calories burned' (500), 'Steps' (2000), 'Distance', and 'Sleep' (0:00), along with an 'Add Data' button and an 'Upload from device' button. A 'Back' button is located in the top left corner.

Current Implementation



The current implementation of the Data Entry page features a navigation bar at the top with links: 'Decentralized Fitness', 'Home', 'Devices', 'Reminders', 'Goals', and 'Reminders'. Below the navigation bar, there is a 'BACK' button. The page is split into two main sections. The left section, with a light blue background, contains a 'Data Type' dropdown menu set to 'Sleep' and a 'Units' dropdown menu set to 'Hours'. The right section, with a darker blue background, contains a 'Data' input field with the value '8' and a 'Submit' button.

We changed the screen mockups to make it easier to use for our customers. In the initial mockup, we split the “Data Entry” page into two sides, where one side was used to add the user’s weight and blood pressure and the other side of the page was used to add the number of calories burned, the amount of steps taken, the distance walked, and the hours slept. However, this was confusing for the user because there were two “Add Data” buttons, one on each side of the screen. If we kept the design the same, then it is possible for users to click on the wrong “Add Data” button, which will result in the user’s information not being saved. Therefore, to make it easier to use and less effort overall, we decided to have one “Submit” button, rather than two “Add Data” buttons.

The image displays three panels of a user interface mockup, each featuring a 'Data Type' dropdown menu and a 'Units' dropdown menu.

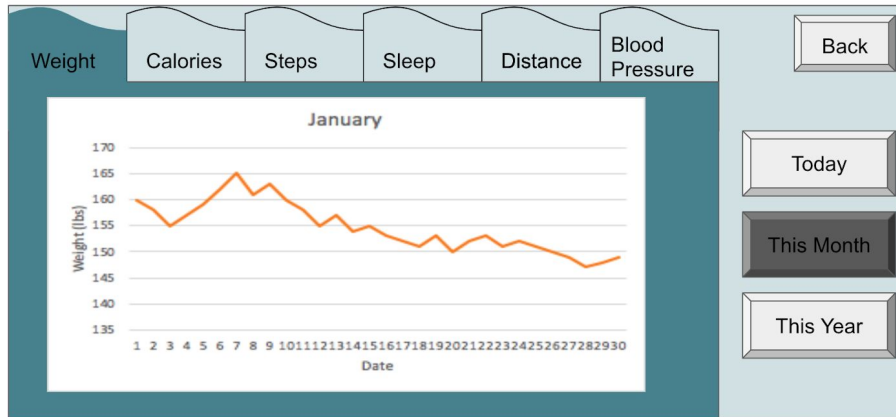
- Panel 1 (Left):** The 'Data Type' dropdown is set to 'Walking distance'. The 'Units' dropdown is set to 'Kilometers'.
- Panel 2 (Middle):** The 'Data Type' dropdown is open, showing a list of categories: 'Sleep', 'Food', 'Walking distance' (which has a checkmark), and 'Heart Rate'. The 'Units' dropdown is set to 'Kilometers'.
- Panel 3 (Right):** The 'Data Type' dropdown is open, showing a list of units: 'Hours', 'Calories', 'Steps', 'Kilometers' (which has a checkmark), 'Miles', and 'BPM'. The 'Units' dropdown is not visible in this panel.

We also noticed that our mockup did not take into account units for each of the categories. Therefore, if a user put in the value “2” for distance, it could be kilometers or miles or any other unit used for measuring distance, but the application would not know. Having customers put in the units themselves requires more effort from the users’ end. It can also lead to issues if the unit is spelled incorrectly, or if an abbreviation is used, such as km for kilometers. As a result, we changed the way the data would be inputted by the user. Instead of putting in all the data at once, which can be overwhelming and confusing for users, we created a drop down menu called “Data Type”, to allow users to select the data category they want add information for. Below that is another drop down menu called “Units”, which is used to select the appropriate unit for the data type chosen. Having a drop down menu is easier to operate because users will already know what units the application accepts, which results in less issues when adding their information. Users will also be able to save time by not having to type out this information themselves.

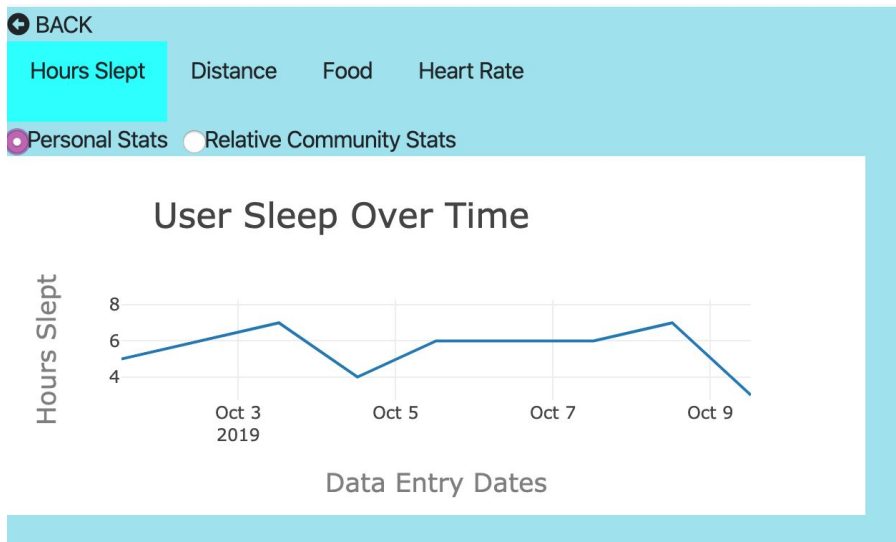
Use Case 4: Viewing Visualizations

Initial Mockup

User Statistics Page



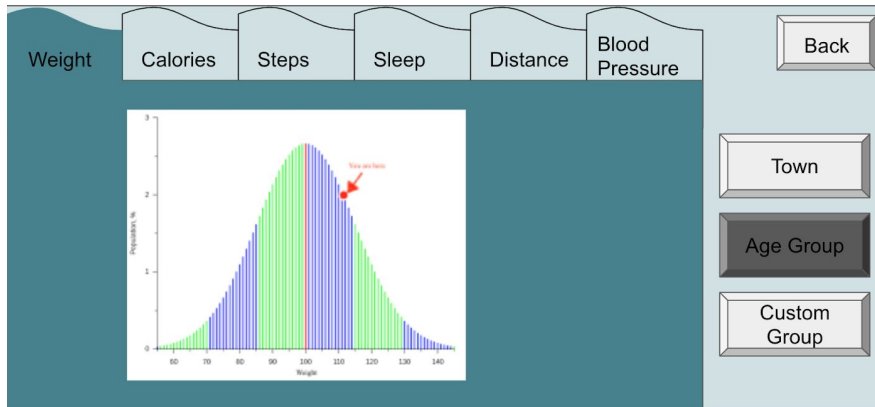
Current Implementation



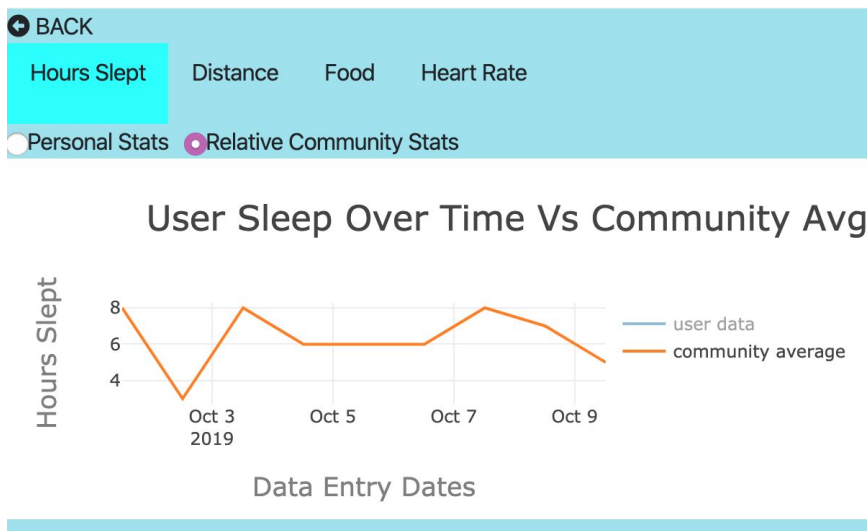
The visuals for the user's data are very similar to the original mockup. Although the implemented page may look a little different visually, the functionality remains the same. The user is able to view his/her personal information as a graph, which allows the user to see his/her data in a meaningful way.

Initial Mockup

Population Statistics Page



Current Implementation



The visuals for the community data are similar to the original mockup. Although the implemented page may look a little different visually, the functionality remains the same. The user is able to view community statistics from a specific population as a graph, which allows the user to see where the general population lies for a specific category.

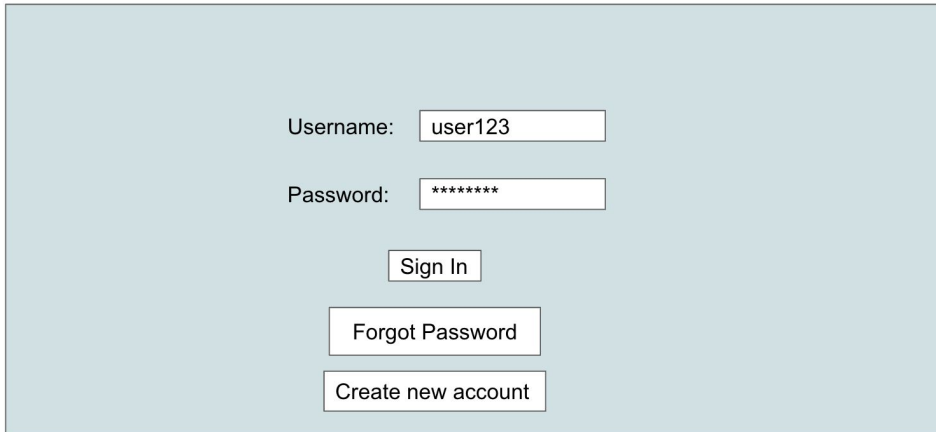


We also improved upon our mockup to make it easier for users to compare their data with the community data. This way, users will be able to see where they lie in relation to the general population. This will allow users to use these comparisons to make any necessary improvements or changes to their current lifestyle. Therefore, being able to look at the two graphs at once makes it easier and a more fluid transition between viewing only personal statistics and viewing performance in comparison to the general population.

Use Case 8: Log In

Initial Mockup

Login Page

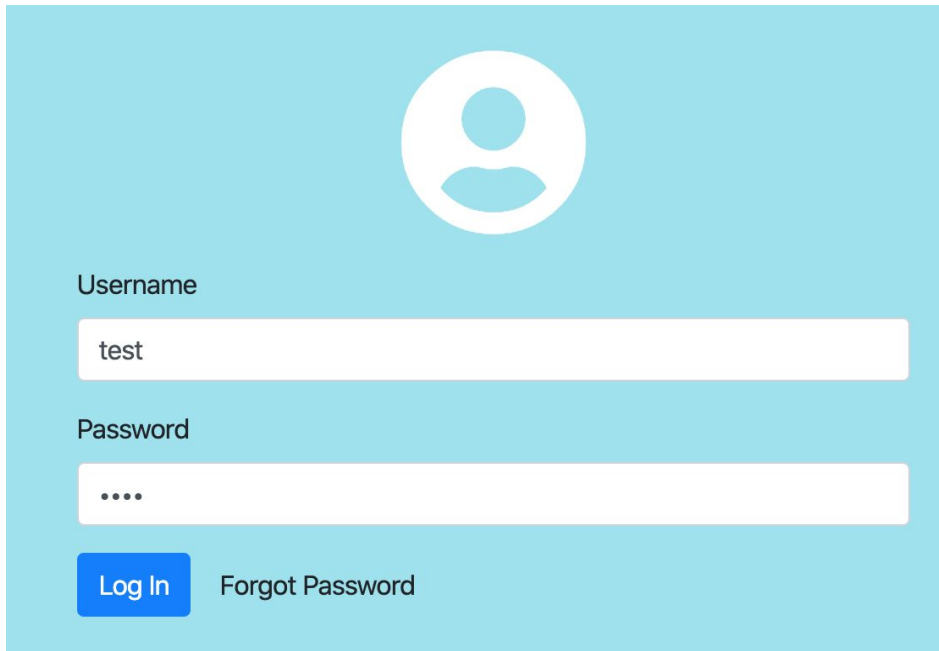


A light blue rectangular box representing a login page mockup. It contains the following elements from top to bottom: a label 'Username:' followed by a text input field containing 'user123'; a label 'Password:' followed by a password input field containing seven asterisks; a 'Sign In' button; a 'Forgot Password' button; and a 'Create new account' button.


Username:

Password:

Current Implementation



A light blue rectangular box representing the current implementation of a login page. It features a white circular user icon at the top center. Below the icon, the label 'Username' is followed by a text input field containing 'test'. Below that, the label 'Password' is followed by a password input field containing four dots. At the bottom left is a blue 'Log In' button, and to its right is a 'Forgot Password' link.



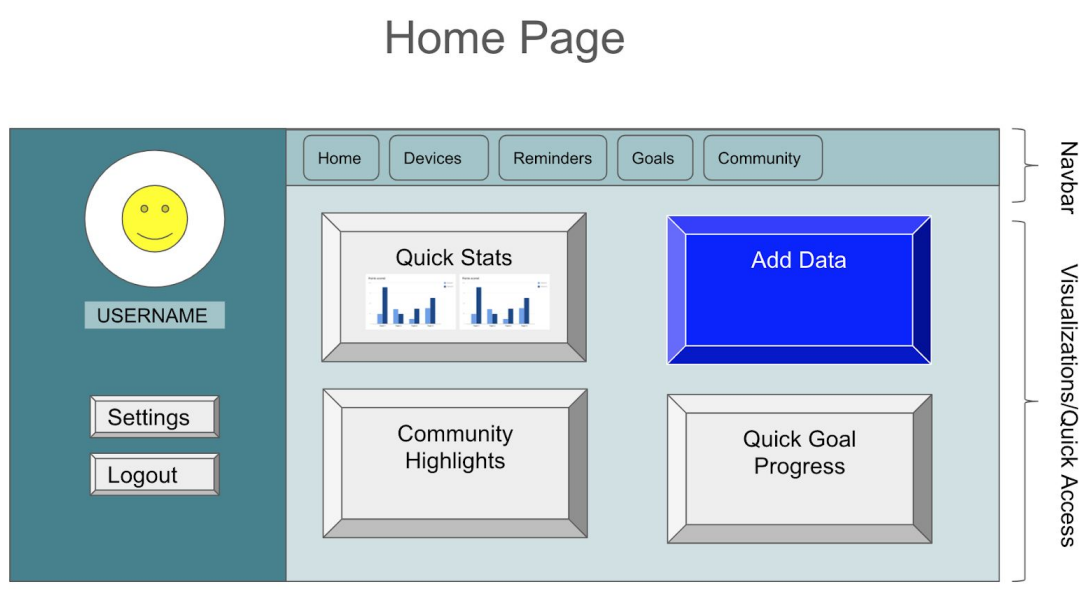
Username

Password

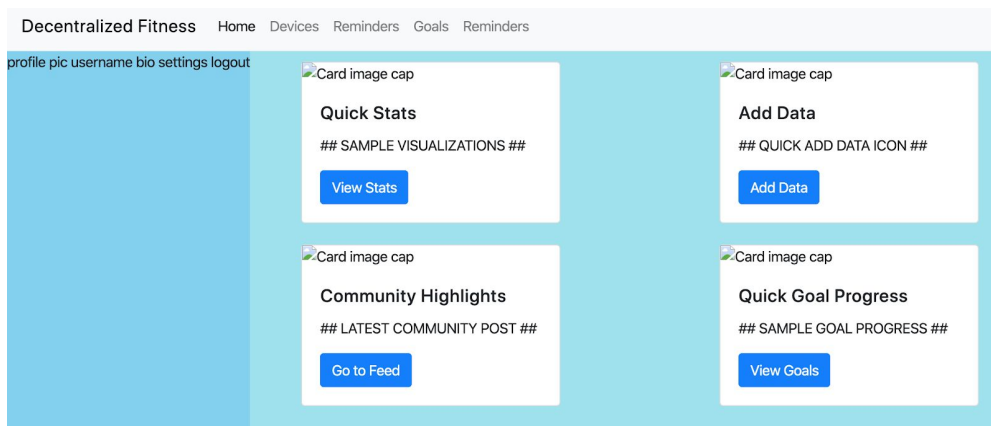
[Forgot Password](#)

The visuals for the login page are very similar to the original mockup. Although the implemented page may look a little different visually, the functionality remains the same. The user is able to log in with his/her correct username and password.

Initial Mockup



Current Implementation



The visuals for the home page are very similar to the original mockup. Although the implemented page may look a little different visually, the functionality remains the same. Once the user logs into his/her account, he/she will first see the home page and can access other pages from the home page.

Section 6: Design of Tests

a. Unit Testing

For the website portion of the application, the first step of testing is navigating to the following URL: <http://se-g3-decentralizedfitness.000webhostapp.com/>. Then, the next portion of testing involved navigating to the Data Entry section and then putting in data for various activities. Entering a data point should yield a pop-up message along with the data point saying that it has been successfully inputted.

With regards to verifying if the blockchain works correctly, two series of tests were run during the first demo. The first series of tests added two blocks of the blockchain, and this showed that the proof of work algorithm was effective because it took around 15 seconds for each block to be placed into the blockchain with a difficulty level of 5. (In reality, it should take around 10 minutes for a block to be placed into the blockchain, so the difficulty level should be set to a 6 or a 7 in reality. However, for demo purposes, the mining process was purposefully shortened.) It also verified that the previous hash attribute of one block actually matched up with the current hash of the previous block and that the data has been successfully inputted. The second battery of tests involved trying to alter the data in the blockchain and then checking the validity afterwards. During the demo, we checked the validity of the blockchain before altering the data. This yielded “true” (it was valid), as it should have done. However, we tried altering the data to another value and then we checked the validity again. This time, it returned “false,” which was the correct value.

b. Test Coverage

During the demo, we tested what would happen if you were to navigate to particular pages on the website and the response of the website whenever you inputted a new data point. We also ensured that the user was able to compare themselves to other users in the system using sample data. Checking for validity of the data points was not done during the first demo, but we will also test for invalid inputs in the next demo by making sure invalid data points don’t get placed into the blockchain and an error message pops up. In reference to the blockchain, we tested how long it would take for a block to be inserted into the chain given a certain difficulty and that the hashes were properly linked (i.e. the previous hash attribute of the current block lined up with the actual previous hash). We also ensured that the blockchain was no longer valid after an outside source tried to tamper with it, which will affirm that the user’s data is secure.

c. Integration Testing

In order to ensure that the blockchain implementation has been successfully integrated with the overall website, we will try inputting a certain data point in the Data Entry section of the website. But before we do this, we will check the state of the blockchain. After entering data, we will check the state of the blockchain again. If the blockchain now contains one more data point than before (the data point we put into the website), then we can say that the two modules have been successfully integrated.

d. Other Testing

In the future, we will test what happens when multiple users are on the system simultaneously and then we will ensure that all of their data successfully goes into the system, even when multiple requests are being handled at the same time. We will also implement rollback in the blockchain, which saves the latest valid version of the blockchain in the case of a rogue agent trying to place falsified data points into the system. That way, all of the data won't be wiped out in the case of a catastrophe. This will be tested by trying to alter a data point like we did in the first demo, but after the blockchain detects an invalid blockchain, it will simply fetch the latest version that was valid and then we will make sure it fetches the previous version that was valid instead of wiping out the entire blockchain.

Section 7: Project Management and Plan of Work

a. Issues Encountered

We have encountered no major issues so far that have impeded us from moving forward smoothly with our application.

b. Project Coordination and Progress Report

We have completed all seven of our functional requirements. However, for requirement #2 which states that the application must remind users on a daily-basis to input their new data, we would like to consider the option of sending a text notification to the user's phone in addition to our current implementation. For the nonfunctional requirements, we have completed req #8 and 12-17. Requirements #9, 10, and 11 are still being worked on in their current iteration. Finally, we have completed all UI requirements except req #22 which states that we must send a text reminder to the user's phone each night, as mentioned in the similar functional requirement #2. All use cases except social-networking on the application between users have been completed. Moving forward, we would like to refine our app for the second (final) demo. We have been able to meet almost all of the requirements we laid out for ourselves in the roadmap, but we would like to make some small quality of life and potentially some aesthetic changes to our application. Thus far, we have been focused on raw functionality, but for the final demo, we would like a polished version of our application such that the customers' usage is straightforward and user-friendly.

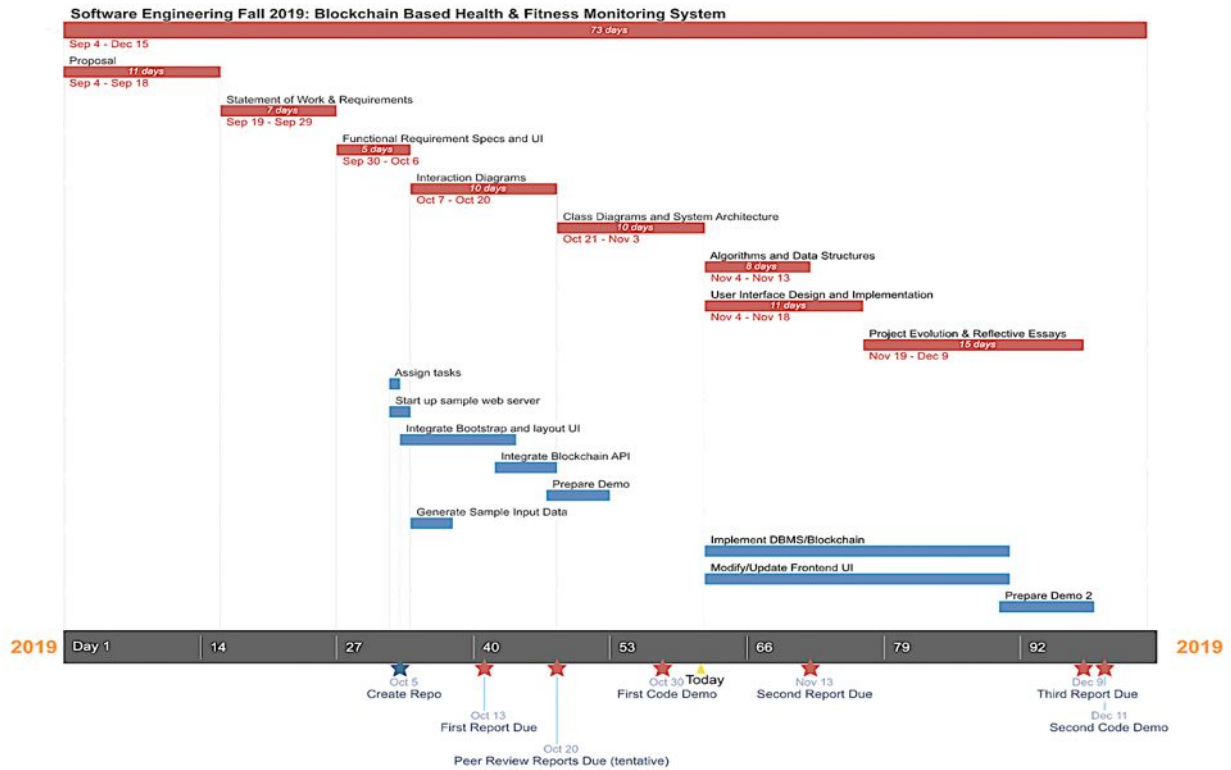
c. Plan of Work

Nov 15 - Dec 9

- Continue coding and debugging
- Continue testing the application to see what needs to be changed or added
- Prepare for the second demo
- Submit report 3

Dec 10-15

- Prepare and present demo 2
- Submit electronic archive



d. Breakdown of Responsibilities

Breanna and Jack will coordinate the integration testing because Breanna was responsible for the data entry portion on the website while Jack was responsible for creating the blockchain, and the data entry portion will be directly linked to the entry point of the blockchain.

References

Unified Modeling Language. (2019, August 25). Retrieved October 21, 2019, from https://en.wikipedia.org/wiki/Unified_Modeling_Language#Interaction_diagrams

Marsic, I. Retrieved October 21, 2019, from <https://www.ece.rutgers.edu/~marsic/books/SE/instructor/slides/>

Multitier Architecture. (2019, July 4). Retrieved November 2, 2019, from https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture

Software Architecture. (2019, October 13). Retrieved November 1, 2019, from https://en.wikipedia.org/wiki/Software_architecture#Examples_of_Architectural_Styles_.2F_Patterns

Class Diagram. Retrieved November 3, 2019 from <https://www.guru99.com/uml-class-diagram.html#10>