CSCI 370 Final Report

Lab Ratz

James Garrison
Jacob Wienecke
Scott Ruckel
Eliot Edwards

Revised June 15, 2022

CSCI 370 Summer 2022

Professor Bodeau

Table 1: Revision History

Revision	**Date**	**Comments**
New	5-17-2022	n/a
Rev – 2	5-18-2022	Initial requirements filled in; team profile edited
Rev – 3	5-19-2022	Team profile edited
Rev – 4	5-20-2022	Requirements revised
Rev – 5	5-23-2022	Risks revised
Rev – 6	5-27-2022	Added system architecture section
Rev – 7	6-03-2022	Added testing and ethical considerations sections
Rev – 8	6-06-2022	Results table added
Rev – 9	6-08-2022	Updated project completion status and lessons learned
Rev – 10	6-09-2022	Added future work section
Rev – 11	6-12-2022	Updated table of contents section
Rev – 12	6-13-2022	Added Lessons Learned section
Rev – 13	6-15-2022	Made revisions from peer group review

Table of Contents

I. Introduction

This project, brought to us by Table Mountain Innovation, is an interface designed to track and compare blood test data from multiple testing corporations over time and coagulate all of the data gathered into an interactive and visually pleasing graph. The purpose of the progressive web application that we are designing is to be able to clearly visualize all different types of blood tests in a singular place so the user can draw conclusions and see improvement or deterioration from their own test results.

Our client, Table Mountain Innovation, is run by Chris Crowley who is an innovator with nearly 40 years of industry experience. When discussing the project with Chris, it was clear he felt very strongly about the topic. Chris recently found out that he had an autoimmune disease after nearly 3 years with hundreds of blood tests taken. When parsing through hundreds of blood tests data reports, Chris pondered that there had to be a better way for users to track and compare data in one place in order to draw their own conclusions and not have to wait for a doctor who could potentially miss the diagnosis altogether. The overall goal, as described by Chris, is to eventually have doctors and users pay for a subscription to the web app and upload their data over time so it can be tracked and monitored.

The source of the data we collect comes from each user uploading their own individual test results in JSON form to our web app, which will in turn make the interactive graph for the user. This makes not only the user a stakeholder in their own data, but doctors and insurance companies as well who may want to interact and study a user's data over time, as long as that is granted through permissions given by the user. The software will be maintained over time through Table Mountain Innovation's software engineers and will follow industry standards closely.

II. Functional Requirements

The user must be able to upload many files into the website that contain the results from a blood test. After this the website will parse through the data file and collect all of the necessary data to display the blood test results in a line graph. The user will then have the ability to choose what components of the blood test they would like to display on the graph. The graph will display the components relative to the healthy range in a normalized value as opposed to in their respective units so we can fit them all on a singular logical y-axis. There will also need to be a hyperlink that is embedded for each test that takes you to the loinc.org webpage explaining the test results.

III. Non-Functional Requirements

The client was very adamant on having a clean presentation of the final design. Also the graph needs to be very customizable. The client wants to be able to look through his own blood test results so he needs many different features that alter the graph and change the data to be able to find trends/problems.

IV. Risks

The process of manipulating, analyzing, and visualizing an individual's medical data introduces several risks. Many of these risks emphasize the importance of the blood test data to be read and displayed accurately and precisely. In order to do this, our progressive web application (PWA) [Appendix A] will require each blood data file to be the supported file type, have the supported formatting, display an accurate graph, and be visually pleasing to the user.

Since our solution may eventually assist an individual in identifying or predicting medical patterns and conditions, it is extremely important for our tool to display accurate results to the user. Otherwise, among other things, the graphical display may suggest inaccurate medical implications. On the other hand, it is possible that certain blood test results were measured or recorded inaccurately. Although showing a user incorrect medical data can be detrimental, the lab that performed the incorrect test is responsible for the consequences that may come from this circumstance.

Ending with an accurate graphical representation of the data begins by having an effective method for reading in the data from laboratory-distributed blood test result files. To ensure that our application will not crash at the data-import stage, only JavaScript Object Notation (JSON) files will be accepted as input. For more details regarding JSON and JSON files, refer to Appendix A. Additionally, because the standard text format for laboratory blood test results is HL7/FHIR format, this is the format that our application will require and initially check the data files for [4].

The program will then check that the JSON file is in the format that our graphing algorithm can handle. Without this check, we risk the formatting of the input file not being properly supported by the program. This would result in the data being read incorrectly from the file, and the graphical representation of the blood test results to be inaccurate.

Although we realize that it is possible that the format considered as standard for blood test results will change in the future, we have no way of knowing what the differences will be compared to the current standard format. Despite this concern, we think refactoring our application to support new formatting types would be manageable, if necessary.

Also, as with any medical result data, we have to comply with the Health Insurance Portability and Accountability Act (HIPAA) [Appendix A], which requires national standards to protect sensitive patient health information [3]. For this project, the user's medical data will not be stored in memory. Although this means that the user will have to reupload all of their blood result files every time they want to reload the graph, it also means that we do not risk incompliance with the HIPAA. Users may be given the option to store their results locally, but the website will retain any user information in order to coincide with the policy.

V. Definition of Done

Our task is to create a PWA that takes blood test data files from the user as input and returns a visually-pleasing, interactive graph to the user. This tool will allow the user to compare data from their complete blood count test results in a way that is convenient and user friendly. Specifically, the graphing tool will allow users to select the individual results that they would like to compare against one another, plot these results on a singular graph, and see a visual representation of the corresponding normal ranges for each result. Additionally, the graphing tool will display doctor's comments for each test result if there are any present.

VI. System Architecture

To accomplish our task we will be using a couple different 3rd party elements to make a graph. In order to display a graph on our website it is necessary to import a JavaScript library. For this we choose ChartJS because it is free and intuitive to use. Also, our client suggested that we use Replit for our IDE. Replit is a fantastic online IDE that has real time multi-person editing like google drive which is ideal for a group project. Replit also features an interactive preview page for the website we are building so that the changes we make can be inspected and tested without leaving the browser tab. Since this is more of a proof of concept than a profitable product, Replit is perfect because of the features it offers, and we don't have to worry about sharing any profits.

We will be building a website so naturally we will be programming in HTML, CSS and most significantly JavaScript. When files are uploaded they will be passed from HTML into a JavaScript function where they will be parsed for the data that we care about. The data from every test will be put into a testResult object that stores the name of the molecule being tested for and all the numbers needed to put it on the graph. Since one blood draw is often tested for many different things, we will have an array of the testResults mapping to the date it was collected on. This is a very intuitive design because the x-axis will be the dates of every blood collection, or the key in our map, and the y-axis will be the results from that day, or the corresponding value in our map. All of the data manipulation will be happening in JavaScript where we will then create a graph and bind it into HTML. CSS will be used to make the website look pretty and put the ever so important background on the graph which conveys what test results are good and which ones were bad.

Figure 1 shows the fundamental wire frames for the website.

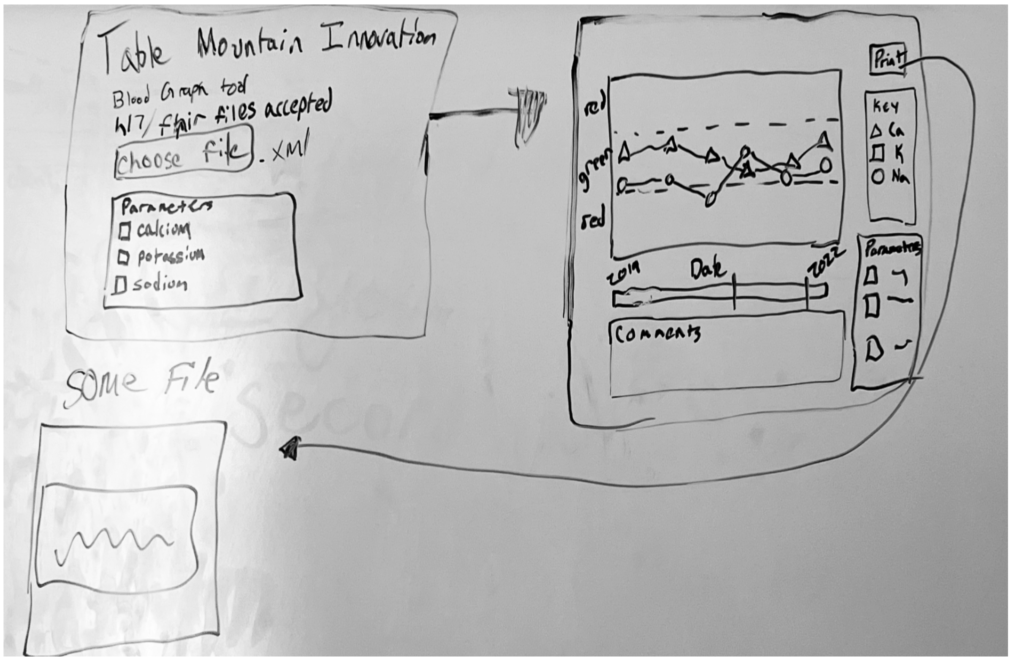


Figure 1: Wireframes for online WebApp

VII. Software Test and Quality

We will use the Unit.js library to import assertions to create unit tests for our backend. The parsing algorithm will be what is tested in most of our unit tests. This collection of tests will be used to ensure that the files a user uploads are parsed correctly and the data is stored into the data structure correctly. These tests will run the parsing algorithm with a collection of files that include all of our edge cases in an attempt to break the algorithm and also some regular files included to make sure we are generating the correct normalized graph value. The following are some examples of the tests we will use to test the edge cases of our parsing algorithm. This will include files with not enough information to ensure that no testResult object is created. It will also include numerous files that have abnormal ranges to certify that our code has the ability to create a normalized graph value for an assortment of different ranges and distinguish those from nonsense ranges. These ranges will include less than/greater than, positive/negative, and nonsense ranges, and the test will assert that all of these graph values are generated and stored correctly. We will also have a test that includes multiple files from the same day to see if the algorithm is able to store these values in the correct spot in the data structure. All of these tests will ensure that the parsing algorithm and the rest of the backend works as intended.

The front end will be much more difficult to have tangible tests for. Instead, we will rely heavily on manual tests that we as a group will perform to ensure that the website has good formatting and clean presentation. We want the graph and the rest of the elements on the page to look neat and professional. This will include testing the website on an assortment of devices to ensure that the alignment and styling looks good on different screen sizes. Also, there are many different ways to navigate through the website. We will make sure that navigating pages and pushing buttons in any order will not affect the underlying data structures holding all of the important information. Testing the front end will be difficult and mostly consist of using the website in ways that we know it should not be used to see if the back end is negatively impacted in any way.

Tables 2 and 3 display the results from unit testing and testing the UI WebApp.

Table 2: Results from UI WebApp Testing

Test	**Result**	**Comments**
UI manual tests	N/A	Everything works well except for the tooltip. Could be a problem with the way we override the tooltip or just a problem with ChartJS. This isn't a huge problem because the correct data is displayed and is easily fixable but definitely something to look into for future work.
Different browsers	Passed	We tested the website on most common up-to-date browsers and everything worked as intended. Browsers tested include: Safari, Firefox, Chrome, Edge.
UI testing	Failed: Bug found and fixed	When files are uploaded multiple times the website generates new checkboxes for every test that has been uploaded. This was fixed by deleting all current checkboxes when the generate checkboxes button is clicked. This way the website will generate checkboxes for every dataset included every time the button is clicked.
Tooltip	Failed	Occasionally when there has been multiple assortments of datasets previously selected, and certain tooltips are activated the graph will revert to previous datasets. This has really weird circumstances when it happens and must be related to the ChartJS library but we were not able to fix it. The graph can be fixed by clicking the graph again or will revert back if the first tooltip in a given dataset is hovered. Tooltip does display correct information but has weird interactions occasionally.
No data stored	Passed	When the website is closed or refreshed the data is erased and the website starts from fresh.
Multiple users	Passed	As long as there are not too many people using it at the same time, multiple users can be using the website locally simultaneously with no crossover of data.
Mobile	Passed	This test passed much better than expected. Website looks great on mobile and works perfectly. The tooltip didn't mess up the dataset and the graph and checkboxes were all generated correctly. We tested on iPhone and Android.
>5 components selected	Passed	To prevent a cluttered graph we made a limit to the amount of datasets that are allowed to be included in the graph. When more than 5 tests are clicked a popup is brought up and no graph is generated. If more color values are added we are able to generate more datasets onto the graph but this will become cluttered and hard to interpret.
Miscellaneous	Passed	This is here to say that many other small things were tested such as random button clicking that don't have formal ways of reporting. The UI has been thoroughly manually tested.

Table 3: Results from Simulated Unit Testing

Unit Test	**Result**	**Comments**
Standard Range	Passed	This tested a given range in the format “xx-XX unit”. All tests from Chris’s data and the files we generated passed. The unit test tested the normalized graph value and the lower and upper bounds, and everything was generated and stored correctly.
Less than	Passed	This tested a given range in the format “< xx unit”. All tests from Chris’s data and the files we generated passed. The unit test tested the normalized graph value and the lower and upper bounds, and everything was generated and stored correctly.
Greater than	Passed	This tested a given range in the format “> xx unit”. All tests from Chris’s data and the files we generated passed. The unit test tested the normalized graph value and the lower and upper bounds, and everything was generated and stored correctly.
Less than or equal to	Passed	This tested a given range in the format “> OR = xx unit”. All tests from Chris’s data and the files we generated passed. The unit test tested the normalized graph value and the lower and upper bounds, and everything was generated and stored correctly.
Positive/Negative	Passed	This tested a given range in the format “POSITIVE” or “NEGATIVE”. All tests from Chris’s data and the files we generated passed. The unit test tested the normalized graph value and the lower and upper bounds, and everything was generated and stored correctly, and everything was generated and stored correctly.
Multiple ranges	Passed	This tested one dataset with multiple different ranges in it. To try and edge case test this, the dataset included ranges from 0.00-1.00 to 100-1000. The normalized values were all generated correctly and without hovering over the tooltip there would be no way to tell that these values were as different as they are.
Upper bound = 0	Failed	This tested a range in which the upper bound is equal to zero. This included “< 0 unit” and “0.0-0.0 unit”. It failed because of a divide by 0 exception, however in practice this is highly impractical. If a blood test were to test for any concentration higher than 0 in a patient’s blood, this would most likely be performed as a positive/negative test and not a given range. If this were to be a problem in the future, we could simply treat it as a positive/negative test for graphing purposes.
Incomplete test	Passed	This tested many files that did not have sufficient data to create a testResult object. These files were parsed but did not include one or more of the necessary data needed
Incorrect file format	Passed	Files that are completely wrong (.txt/.pdf/wrong format) will not be interpreted correctly by the parsing algorithm at all and won’t generate a testResult object. These files are basically disregarded like insurance files that don’t include a test.

Unit testing was particularly difficult because we wanted to test our parsing algorithm with many edge cases, however our code is based on the data we were given and do not know what other ranges or file formats could be possible. Sufficient testing on the parsing algorithm will be possible when more data is provided, and when HIPAA is less of a concern.

VIII. Project Ethical Considerations

The process of handling, manipulating, and analyzing medical data introduces ethical concerns that should be addressed.

Firstly, a tool that is intended for medical analysis and/or predictions requires a high level of accuracy, it is particularly important to ensure that extra attention to detail is being paid wherever necessary. Association for Computing Machinery (ACM) [Appendix A] Software Engineering Principle 1.03 states that software must not diminish quality of life. If this product were to fail to report and graph the data accurately, the user could be harmed. This product could potentially become a tool that users and doctors rely on for medical diagnosis and treatment; it is imperative that the output data is entirely correct. As an end product, there should be proper disclaimers in place to ensure that the graphing software data is cross-checked with the original lab tests.

Furthermore, since the client hopes to eventually possess a complete and monetizable product, it is possible that additional ethical questions will be raised with succeeding projects. In general, the act of monetizing a product that may be used for medical analysis and/or predictions should be carefully considered. Because this tool specifically may be able to predict and/or explain an individual's health history, it is imperative that the target audience for the product is carefully selected. ACM [Appendix A] Software Engineering Principle 1.03 states that software should not diminish privacy. A product dealing with medical data needs to be strictly regulated so that confidentiality is maintained. The only people with access to one's medical data should be themselves and the doctor. The team developing this product should not be able to view the data either.

IX. Project Completion Status

The deliverable for this project was a Progressive Web Application that takes blood test data files as input and displays a visually-pleasing, interactive graph to the user. This tool is made up primarily by a parsing algorithm that extracts values from each blood test and normalizes them relative to the given healthy range for that test. By normalizing the data, the graph is able to accurately portray the trends of more than one test at a time. In addition, the graphing tool allows the user to hover over each data point on the graph and see the date, actual range, and results of that specific test. Because the graphing tool satisfies the requirements made by the client, the project is considered complete.

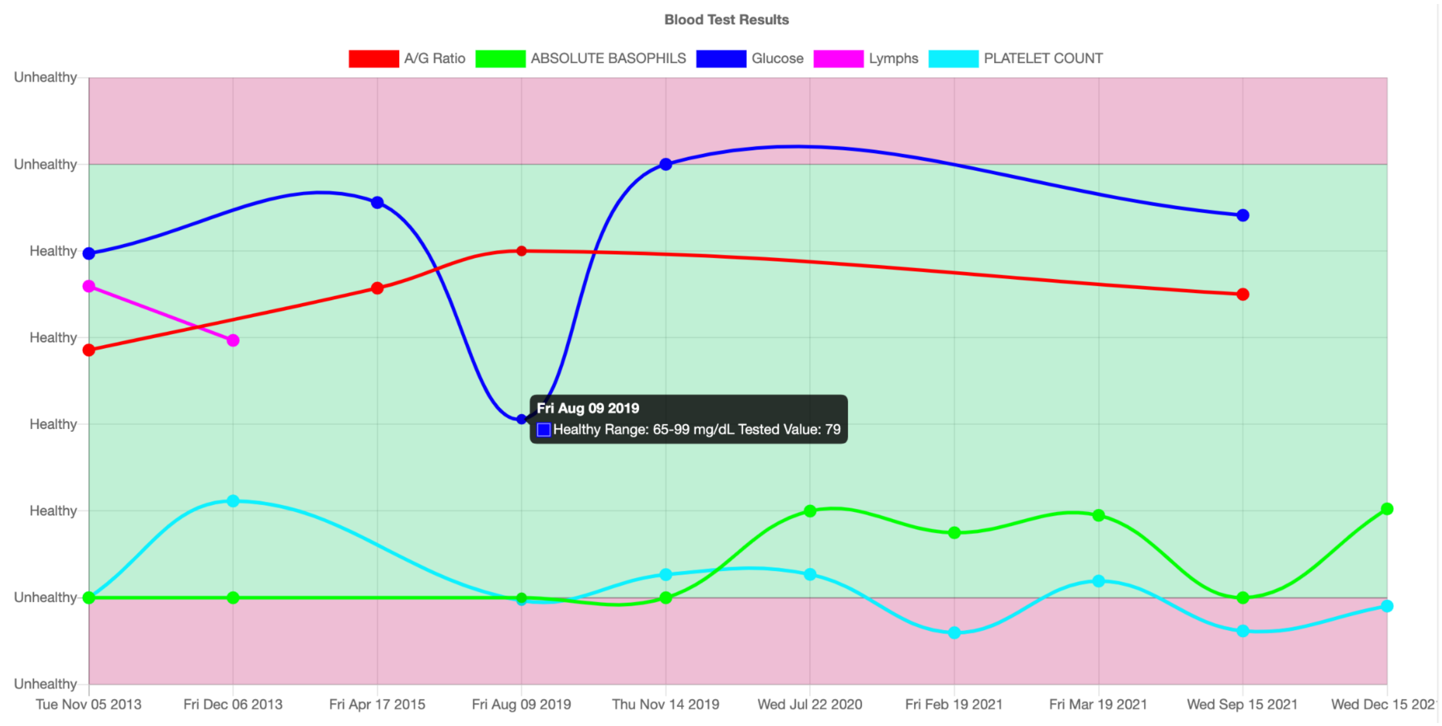


Figure 2: Final PWA Graphical Tool

X. Future Work

Future contributors to this project will have to be cognizant of HIPAA privacy and security rules. The current state of the project is a proof of concept. HIPAA rules have been considered, but have not been closely followed. The web application is currently hosted by a third party website, so a new host will have to be chosen to responsibly handle user data. An end goal for the client is to have user-made accounts so that users can revisit their data without having to re-upload blood test files. This implementation will require the design of a login interface. Additionally, the project currently features a basic, but functional, user interface. A future implementation may include improving the existing interface to add more functionality or otherwise make it more visually-pleasing. Future work may also involve adding features to make the application more user friendly. Adding tool-tips or intuitive buttons that are easy to spot may be a positive inclusion.

XI. Lessons Learned

This project was much more open ended than classwork that we have done at Mines in the past. This gave us an opportunity to work in a much different way than we typically do and thus had a fair amount of problems as we went through the project. Specifically, we learned the importance of team dynamics. We did not have too many problems in our team dynamic, however the reason we had success and were able to have fun throughout the project was due to our dynamic. Communication amongst the team is necessary. This had a little bit of a learning curve towards the beginning as our team had never worked together before and we had to get to know each other. However, once we delegated a project manager this conversation became much more targeted towards the project and we began to work better together. Having a good manager, keeping everyone on track while still embracing the fun aspect, is essential to having a group that communicates and works well.

It is also of utmost importance that the client is satisfied. This means creating a product that satisfies the clients requirements and vision for the product. It can be hard for a client to portray their vision accurately, so it is necessary to ask many questions that clarify what the client actually wants to see in the product. We asked these questions and put forth a product that our client liked, however this was after producing code that did not work well with the vision he had in mind. As a computer scientist, it is imperative that you understand every detail the client wants before you begin to code or you will end up redoing that code. Rewriting code that you already implemented incorrectly is a waste of time, and minimizing this optimizes the time that you are able to write code that performs as intended. In order to catch these mistakes early, sufficient testing is required. Testing allows a programmer to efficiently find out what edge cases are not implemented correctly and saves loads of time debugging. We tested our website for quality assurance before handing the product back to the client, however having the unit tests that we made while programming the product would have saved us a lot of time. The unit tests would have streamlined testing the parsing algorithm. In the future we will definitely make sure to have unit tests for any large software feature before jumping into the implementation. Our group worked well, but we definitely had weak spots that will be addressed in our future endea

XII. Team Profile

James Garrison (jgarrison@mines.edu)

Project Manager

James is a Senior at Colorado Mines studying Computer Science with a concentration in Business. Originally from Cave Creek, AZ, he came to Mines to be a member of the Colorado Mines Football Team and receive an excellent education. While he does not have prior work experience in the Computer Science field, he worked full-time at the Golden Mill located in downtown Golden, CO. He is passionate about making the CS industry ethical and innovative, and shares these passions with his group members and fellow colleagues.

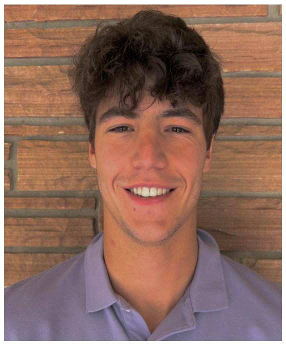
Jacob Wienecke (jacobwienecke@mines.edu)

Advisor POC

Jacob is a Senior at Colorado School of Mines pursuing a degree in computer science with a focus area in robotics and intelligent systems. He was drawn to Mines by the prestigiousness of the school and an offer to play for the Mines Football Team. The chance to play sports and pursue a STEM degree proved to be an offer he could not pass up. He does not have work experience corresponding to the Computer Science field, but he worked two Summers managing a team of lifeguards for a state park in the Austin, Texas area. As a type 1 diabetic, he hopes to use his degree to make life easier for those with disabilities through medical technology.

Scott Ruckel (scottruckel@mines.edu)

Client Liaison



Scott is an upcoming senior at Colorado School of Mines studying computer science. His journey to Mines started when he was just a little boy as he would frequently visit Golden and the Mines geology museum with his alumni grandfather. Since then he has worked hard to not just get to Mines but play football here as well. While in college he has held a couple of different jobs, most notably a Millwork Associate at Home Depot. He is jumping at the bit to get out into industry and start making a difference with his programming and engineering abilities.

Eliot Edwards (eedwards@mines.edu)

Scrum Master

Eliot is a senior at Colorado School of Mines, where she is currently finishing up her Bachelors degree in Computer Science. Inspired by her brothers, and in addition to her studies, Eliot grew up playing soccer competitively. Originally from South Denver, she was eager to continue her competitive soccer career at a high-quality STEM school not far from home. Moving forward, Eliot is passionate about furthering her personal, technical, and professional development through opportunities that will allow her to challenge herself emotionally, academically, and athletically.

References

[1] “About the ACM organization,” *Association for Computing Machinery*. [Online]. Available: https://www.acm.org/about-acm/about-the-acm-organization. [Accessed: 15-Jun-2022].

[2] Cleveland Clinic, “Complete Blood Count | Cleveland Clinic,” *Cleveland Clinic*, 2014. [Online]. Available: https://my.clevelandclinic.org/health/diagnostics/4053-complete-blood-count. [Accessed: 20-May-2022].

[3] “What Is a Progressive Web Application?,” *Codecademy News*, Sep. 17, 2021. [Online]. Available: https://www.codecademy.com/resources/blog/what-is-a-progressive-web-application/. [Accessed: 20-May-2022].

[4] “Health Insurance Portability and accountability act of 1996 (HIPAA),” *Centers for Disease Control and Prevention*, 14-Sep-2018. [Online]. Available: https://www.cdc.gov/phlp/publications/topic/hipaa.html#:~:text=The%20Health%20Insurance%20Portability%20and,the%20patient's%20consent%20or%20knowledge. [Accessed: 20-May-2022].

[5] “JSON File Extension - What is a .json file and how do I open it?,” *Fileinfo.com*, 2018. [Online]. Available: https://fileinfo.com/extension/json. [Accessed: 20-May-2022].

[6] ”What is HL7? Definition and Details,” *www.paessler.com*. [Online]. Available: https://www.paessler.com/it-explained/hl7. [Accessed: 20-May-2022].

[7] “What Is FHIR®?” [Online]. Available: https://www.healthit.gov/sites/default/files/2019-08/ONCFHIRFSWhatIsFHIR.pdf. [Accessed: 20-May-2022].

Appendix A – Key Terms

Term	Definition
Association for Computing Machinery (ACM)	*Group that outlines the requirements for ethical professional conduct [1].*
Complete Blood Count (CBC)	*A blood test that measures and counts blood cells to detect a range of disorders and conditions [2].*
Fast Healthcare Interoperability Resources (FHIR)	*A standard that defines how healthcare information can be exchanged between different computer systems regardless of how it is stored in those systems [7].*
Health Insurance Portability and Accountability Act (HIPAA)	*A federal law which required national standards to protect sensitive patient health information from being disclosed without the patient's consent or knowledge [4].*
Health Level 7 (HL7)	*A set of clinical standards and messaging formats that provide a framework for the management, integration, exchange, and retrieval of electronic information across different healthcare systems [6]. HL7 uses a collaborative approach to develop and upgrade FHIR [7].*
JavaScript Object Notation (JSON)	*A standard data interchange format that is primarily used for transmitting data between a web application and a server [5].*
JavaScript Object Notation (JSON) File	*A file that stores simple data structures and objects in JavaScript Object Notation [5].*
Progressive Web Application (PWA)	*A type of application software that is designed to work on standard web browsers [3].*
User Interface (UI)	*The user interface (UI) is the point of human-computer interaction and communication in a device.*