From Bodily Sensors to the Cloud and Back

Design Document · Group 26

Client – Goce Trajcevski
Advisors – Goce Trajcevski, Liang Dong, Sung Min Kang

Isaac Zahau – *Front End/UI* – isanga@iastate.edu
Justin Worely – *Cloud Engineer* – jmworely@iastate.edu
John Kivley – *Electrical Engineer* – jekivley@iastate.edu
Richa Patel – *Database Engineer* – rppatel@iastate.edu
Michael Lauderback – *Embedded Systems Engineer* – mlauderb@iastate.edu

Team Website
`https://sddec20-26.sd.ece.iastate.edu`

Executive Summary

Development Standards and Practices

- IEEE standards
- Waterfall Software Development
- Circuit, Block, and Use Case diagrams

Summary of Requirements

- At least 2 sensors
- Data stored on the cloud
- Transmit and receive data securely
- User-friendly user interface

Applicable Courses from Iowa State University Curriculum

- EE 201 – Electric Circuits
- EE 230 – Electron Circuits and Systems
- EE 285 – Problem Solving Methods and Tools for Electrical Engineering
- EE 321 – Communication Systems I
- CPRE 281 – Digital Logic
- CPRE 288 – Embedded Systems I: Introduction
- CPRE 388 – Embedded Systems II: Mobile Platforms
- COM S 227 – Object-oriented Programming
- COM S 228 – Introduction to Data Structures
- COM S 319 – Construction of User Interfaces
- COM S 309 – Software Development Practices
- SE 329 – Software Project Management
- SE 339 – Software Architecture and Design

- COM S 362 – Object-Oriented Analysis and Design
- COM S 363 - Introduction to Database Management Systems
- COM S 474 – Introduction to Machine Learning

New Skills or Acquired Knowledge

- Gained knowledge of industry tools used for hardware design such as KiCad.
- Gained knowledge and skill in designing electrical hardware starting from an idea to the final product.
- Gained knowledge and honed skills in troubleshooting and debugging hardware designs and learning multiple methods to conduct troubleshooting.
- Gained knowledge in project documentation practices.
- Gained knowledge in Amazon DynamoDB
- Gained knowledge on AWS infrastructure.
- Gained knowledge on features of Spark and Cassandra
- Gained knowledge on different types of databases.
- Gained knowledge in MVC

Contents

List of Figures

List of Tables

1 Introduction

1.1 Acknowledgement

We would like to acknowledge Goce Trajcevski, Liang Dong, and Sung Min Kang. Goce Trajcevski is our advisor. He has been very helpful, guiding the project in a successful direction. Sung Min Kang and Liang Dong have been there to offer great advice in regards to sensors for our project. We would also like to acknowledge Sarah Radke, Jessica Terrell, and John Stankovic for they were the sources of our market survey.

1.2 Project and Problem Statement

General Problem:
Doctors cannot monitor all of their patients at once. At a medical center or hospital, a doctor must allocate their time accordingly to the needs of their patients. Some patients require more attention than others due to the severity of their condition. In addition, some patients wish to not remain in a medical center or hospital; however, their condition requires a higher level of monitoring from a medical professional.

General Solution:
To solve these and other possible needs, we are using remote data storage using a cloud platform and sensors placed in different locations on the human body. We will incorporate different sensors placed in key locations that all talk to a master control unit (MCU). This MCU will then process the data from all of the sensors. If the mobile application is open, it will transmit live statistics to the mobile application. Along with the mobile application the MCU will also send the data that was collected to the cloud platform. Once there, that data will be stored and analyzed to provide the user with multiple ways to view their statistics, as well as their medical professional. This way, it is easier for doctors to monitor multiple patients, as well as give the patient the freedom to leave the hospital or medical center since the doctors can now monitor the patient remotely.

1.3 Operational Environment

The end product is expected to be operational under various environments whether inside a hospital or home, or outside in sunshine, snow or rain. In addition, it is also expected to operate properly, even when exposed to sweat, repeated motion, mild detergents, and various clothing material.

1.4 Requirements

1.4.1 Engineering Constraints

- Use at least two on-body sensors that will measure particular vitals of the human body to create an IoT system from monitoring bodily functions.
- Sensors must have low power consumption, be wearable, and have a dedicated power supply to ensure the device is portable. Furthermore, it needs to be reliable to ensure the product can be used on a consistent basis collecting information accurately.
- The sensors must be able to transmit data to an MCU using digital signals where MCU relays the data to the user's phone. The MCU relays digital signals directly and it converts received analog signals to digital signals using an ADC before relaying the data.
- The mobile application must be able to pull data from the database.

1.4.2 Functional Requirements

- All electric hardware must be protected from outside elements when being worn on the body. The hardware cannot be affected by the rain, snow or sweat.
- The UI must be easy to navigate while providing details from every sensor. The visual data should be easy to view and manipulate.
- All transmissions to and from the cloud need to be encrypted.
- The database must be cloud-based to store and access data in real time.

1.5 Intended Users and Uses

Our team conversed with multiple medical professionals and researched the work of John Stankovic, a professor at the University of Virginia whose research focuses on sensor-based health care. The end product will be wearable and easy to use. Furthermore, it will provide information regarding heart rate, temperature, and body weight all in real time.

1.6 Assumptions and Limitations

Below is an itemized list of assumptions driving the project.

- The maximum number of users will greatly vary over any given time.
- The product is currently limited to the United States of America.

- The users will agree to an information disclosure allowing bodily readings to be stored in a remote location.
- There will be a monthly subscription charge to access and store bodily data.
- The main body sensors will monitor temperature, heart rate, and weight.
- The end product will consist of multiple wearable items, each embedded with sensors for monitoring vital functions.
- Design and build our own MCU board to receive and relay the data from the sensors.
- The MCU will relay sensor data to the user's phone which will relay the data to the cloud.

Below is an itemized list of limitations driving the project.

- The cost of the final physical product should not exceed $100.
- Due to funding and limited time, less than ten products will be available for testing.
- Majority of testing will be with simulated users and devices.
- Due to the limited amount of users, scalability will not be able to be tested as effectively.
- The system must have a power supply consisting of batteries that will power the device for at least the majority of the day.
- The system must operate on 5.5VDC or less.
- Due to the nature of the data being collected, all data transferred must be encrypted.

1.7 Expected End Product and Deliverables

The final product is a system that integrates a wearable device that includes multiple on-body sensors, communication protocols, data management, and mobile app. The sensors will measure the mobility, heart rate, and more of the user. This product will be portable enough for the users to take away from home and to use during everyday events. The product is also able to connect to a mobile phone to display live data when WiFi is not available. The device will use the bluetooth functionality to connect to the mobile phone and display the live data in a clean user interface. If WiFi is available, the mobile application will be able to pull in processed data from the cloud and display the data onto the application.

2 Specifications and Analysis

2.1 Proposed Approach

Our system will utilize sensor(s) to track different vitals. Based on the readings from one or more sensors, more data will be collected or a notification will be sent to the user, medical provider, or both. To accomplish this we will have one or more sensors collecting data and send that data to a local MCU. The local MCU will then compile that data and send the data to a user's mobile device. A user will utilize our mobile application to visualize and transmit the data received from the MCU to the cloud database.

When data is received by the database it will also be sent to a cloud server where data analytics will determine if there are any abnormalities in the sensor readings. If any abnormalities are found, the server will request more data from the MCU through the mobile application. The MCU will increase sampling rate from active sensors, request data from inactive sensors, or both. Once more data is available for analysis the server will notify the user or medical provider about the possibility of a medical concern if necessary. If the server did not identify a medical concern it will notify the MCU through the mobile application to resume prior sensor monitoring.

2.2 Design Analysis

We discussed and researched databases, sensors, and package designs. We decided to focus on getting one sensor, a pulse sensor, fully integrated into the system. The pulse sensor is very easy to set up since it only has three connections. We also discussed using an ESP board for bluetooth and WiFi connectivity, but ultimately decided to use the STM32WB55, an ARM chip with integrated wireless support. The primary reason for this was because the ESP module is quite large and would be cumbersome in our sensor application. The STM chip also has development kits called nucleo boards which are well supported. The STM chip would also be easier to integrate into a custom PCB since it is one component, not two. We decided that due to time constraints, it was unfeasible to design and order a custom PCB. Mentions of a custom MCU design in the remainder of this document are academic, reflecting an overall plan without execution.

We have decided to use AWS DynamoDB as our database as it enables easier specifications for triggers and lands itself naturally to AWS cloud services which, in turn, enables our project to execute in distributed settings. We have also set up a test application to verify the connection between the database and the mobile application.

2.3 Development Process

We are using the Agile development process because the system consisted of heterogeneous components and it was beneficial to enable various integration/interface

testing along the partial implementations throughout the project development.

2.4 Conceptual Sketch

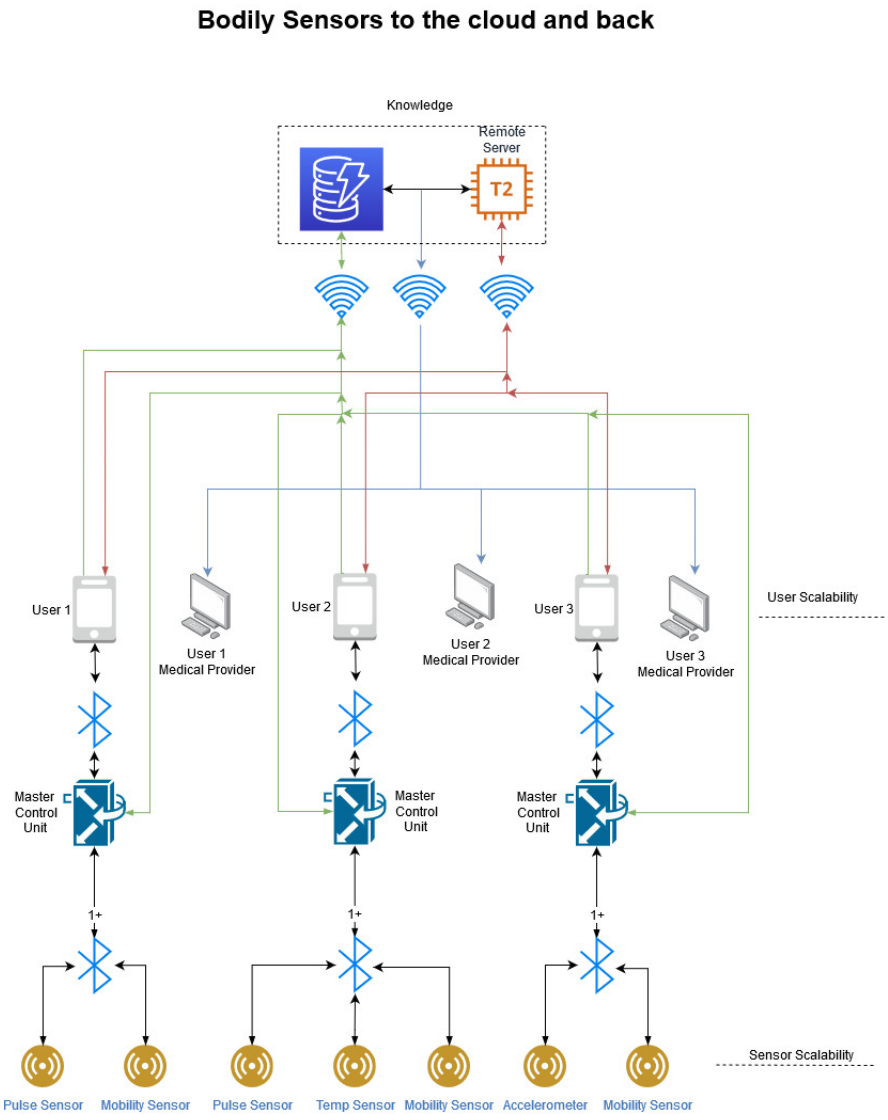

Figure 1: System Diagram

Figure 1 represents our entire system for the first three users. Users can have as many sensors as they choose, and the system is capable of adding more users.

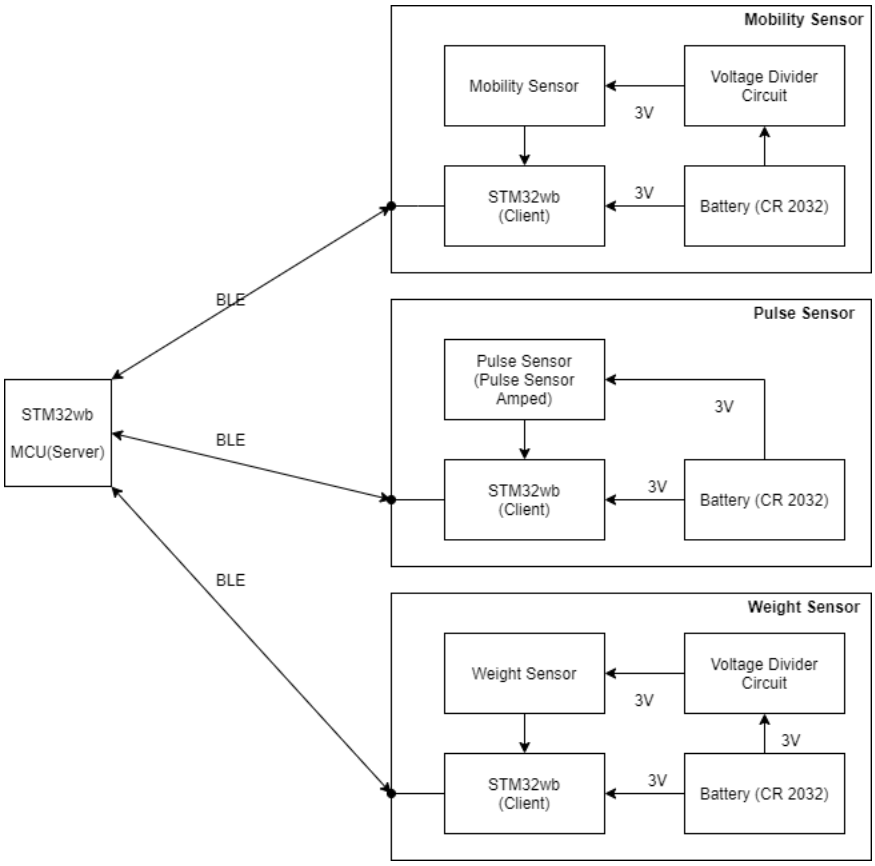

Figure 2: IoT Sensor Network

Figure 2 represents a sensor IoT network for a single user. Each user will have their own MCU (left side of the figure) which is equipped with an STM32WB chip to communicate with the sensors via bluetooth low energy. More information on the STM32WB chip can be found in section 3.2. The MCU will be collecting data from the sensor to relay it to the user's mobile phone and the cloud. This user has three sensors: one to measure their body temperature, one to measure their pulse, and one to measure their weight. Each sensor has its own battery power supply as well as an STM32WB.

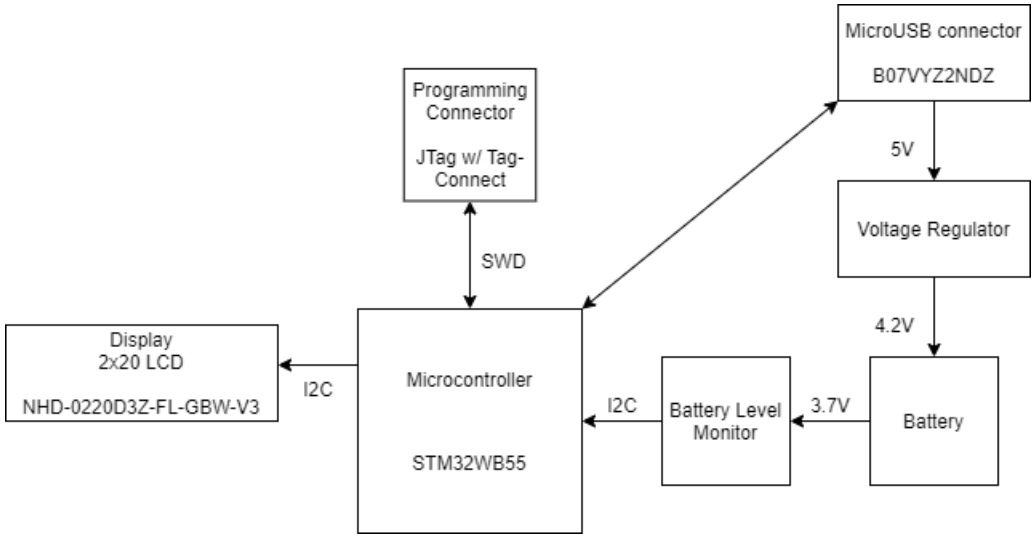

Figure 3: MCU Architecture

Figure 3 shows an outline of a single MCU. Each unit is equipped with its own battery supply. In addition, it will have its own battery level monitor to determine the state of charge in the batteries, allowing the user to keep track of how much charge remains, and determine if the system needs to be recharged by plugging in external power to the micro usb connector. The display shows MCU-specific information to the end user like battery level, paired sensors, and network status. The programming connector is used to load code to the MCU and debug.

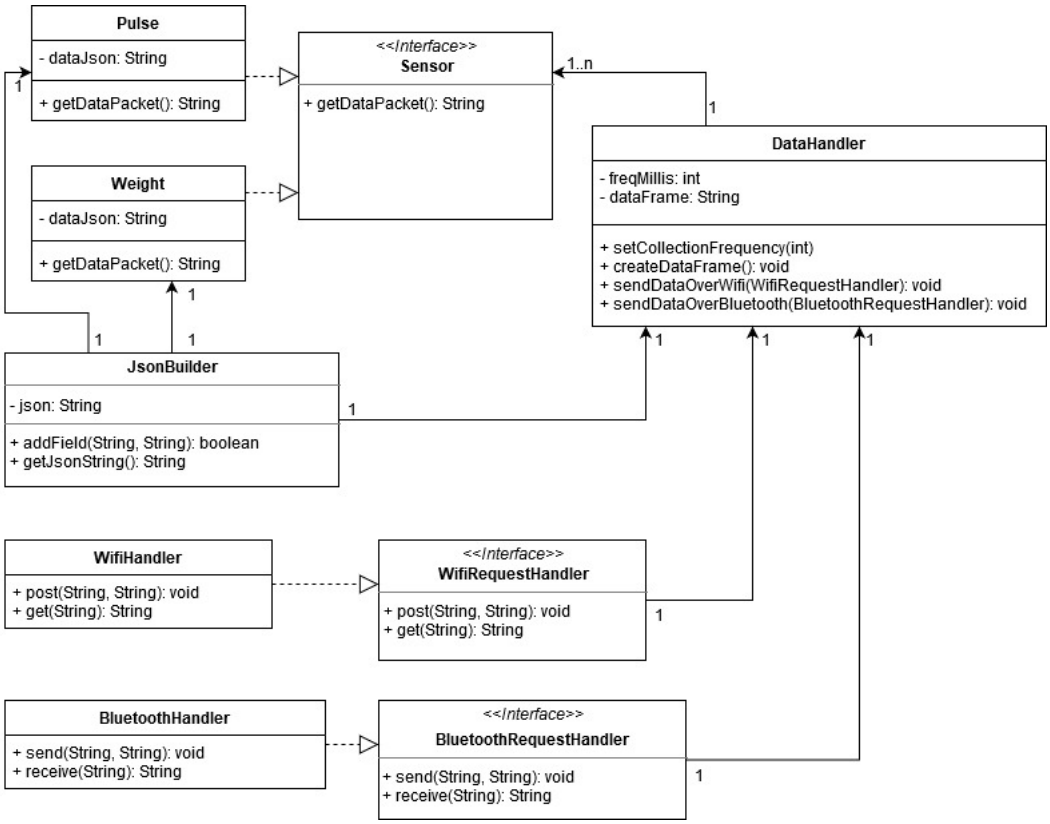

Figure 4: MCU Connectivity and Scalability

Figure 4 shows a class diagram of the MCU, illustrating how it keeps sensors scaleable and how it handles network requests for WiFi and bluetooth. The sensors shown are pulse and weight sensors.

To transfer data between sensors, the MCU, and the user's phone, we designed an encoding standard (data frame) which can be easily built and decoded. This data frame assumes the following pattern:

Type Code	**Data Type Code**	**Return Type Code**	**Data**	**End**
1 byte	1 byte	1 byte	n bytes	1 byte

Figure 5: Encoding Standard Overview

The type code tells us where the information comes from. The data type code tells us what the data type of "Data" is. The return type code tells us what the purpose of the data is. The data section is a single unit of the data type specified. This constrains n to the number of bytes of the specified data type. Finally, the end byte is set to 0. Below is a tabulation of type code values and their uses cases.

Type Code	Use Case
0x01	ACK. signal from MCU → Server
0x10	Message from Server → MCU
0x02	MCU → Server about respirations
0x03	MCU → Server about heartbeats
0x04	MCU → Server about weight

Table 1: Type Code Use Cases

The data type section is one unified byte containing a code telling us in what data type the data is formatted. A value of 1 means the data is an integer, a value of 2 means the data is a double, and a value of 3 means the data is a string. The return type tells the receiving end what the purpose of the data is. Below is a tabulation of return type codes and their uses.

Return Type Code	Signal Type	Use Case
0x00	SENT	Used for sending sensor data to server
0x01	ACK	Used to acknowledge sensor data received
0x02	MOD2	Modify respiration collection frequency
0x03	MOD3	Modify heartbeat collection frequency
0x04	MOD4	Modify weight collection frequency
0x11	ACK	Acknowledge and ACK signal
0x21	ACK	Acknowledge MOD2 signal
0x31	ACK	Acknowledge MOD3 signal
0x41	ACK	Acknowledge MOD4 signal

Table 2: Return Type Code Use Cases

Figure 6 illustrates a communication protocol that the MCU uses to interface with the cloud and the user's phone. All sent and received communications are to use the encoding standard mentioned in the previous paragraph. The purpose of designing the communication this way is to ensure that the MCU, cloud, and user's phone are all synchronized such that each entity knows what is being collected and how.

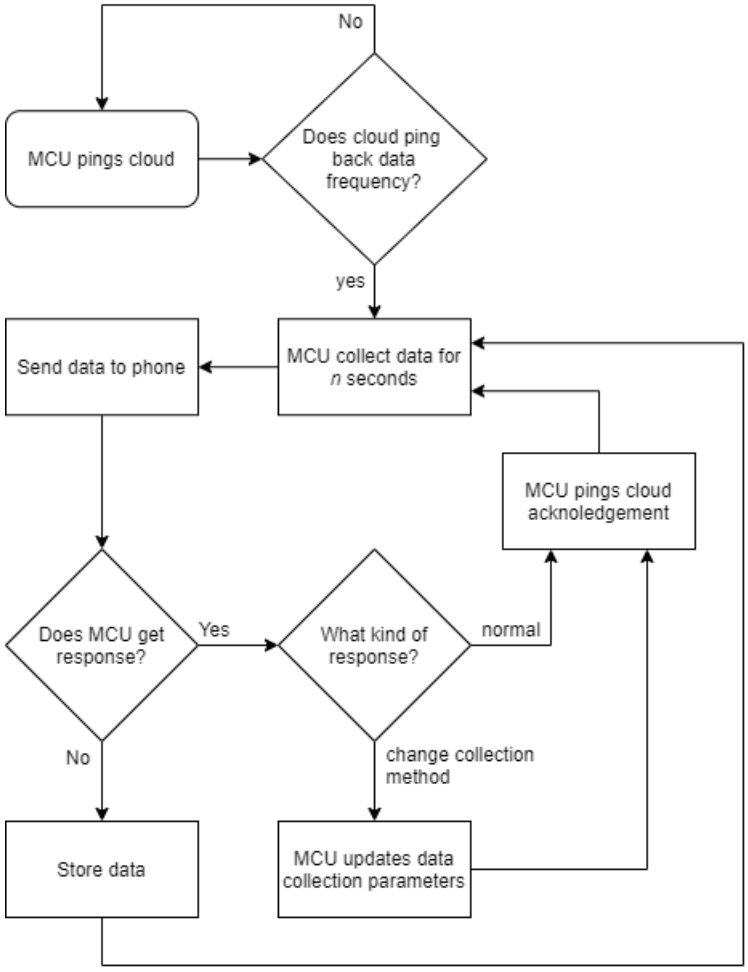

Figure 6: MCU-Phone Communication Flowchart

3 Statement of Work

3.1 Previous Work and Literature

John Stankovic is a professor in Computer Science at the University of Virginia. His research consists of a smart home system for patients with various health problems such as dementia, depression, obesity, and epilepsy. Different sensors such as a trip sensor, pressure sensor, floor sensor and many other sensors are placed in a living space. These sensors will detect any irregular activity and will notify the patient's medical provider if needed[1].

Our product will be similar to Professor Stankovic's research with the exception of creating a smart home device. This product places its focus heavily on the on-body sensors and uses different sensors to detect any irregular patterns with the patient.

Measurements from the sensors will be sent to the cloud, the cloud will process the data, and send a message back to our device if any other measurement is needed. Our web application will be used by the medical provider to keep track of the patient's health.

3.2 Technology Considerations

Technology considerations are split up into seven groups: hardware/embedded, connection board options, pulse sensor options, battery options, user interface, and cloud services data analysis/databases.

3.2.1 Hardware/Embedded

To keep sensors small and provide the end user with a central hub for all sensors, we will need some kind of MCU. After consulting with ETG about options for the MCU, we narrowed our focus to 3: the Teensy board, Raspberry Pi Zero W, and a custom design [2].

The Teensy board is a small arduino board with limited GPIO. It has two analog-digital converters which can be useful for displaying waveforms, and because it is an arduino, it is comparably easy to program and test. The teensy board is also very cheap. The two largest drawbacks that make the teensy board an unviable option are that it does not have built in bluetooth or WiFi. To implement both bluetooth and WiFi, we would need a riser connecting the GPIO holes which makes the finished unit too bulky.

The Raspberry Pi Zero W is a very small form factor raspberry pi with built in WiFi. Like the teensy board, it is quite cheap and easy to test [2]. The Pi Zero W also has a small, prebuilt battery pack, keeping to our form factor restrictions and eliminating the need to design battery technology. Unlike the teensy board, it does not have analog-digital converters, so clients will not be able to view raw waveforms and would have to rely on other tools to do so. The Pi Zero W does not have bluetooth, so, like the teensy board, we will have to use a riser to implement it. Perhaps the biggest drawback to the Pi Zero W is that it is overkill. To run any program, some version of linux needs to be installed. For our application, we do not need the HDMI port, nor the USB C ports. All we need are C libraries to run the MCU code, so using something like Linux From Scratch would be optimal.

The final option is a custom design. The greatest benefit of using a custom design is that we can implement everything exactly as needed, resulting in no extra hardware and no risers compromising form factor. Designing a custom system from scratch has its drawbacks as well. The system will be more difficult and expensive to test, and will be more time consuming to develop [3].

Given the three aforementioned options, we have chosen to build a custom design. The Teensy board is too bare-bones, requiring risers to include all of the functionality we need which hurts form factor. The Pi Zero W, while small and in some ways

convenient, was out of phase with what we are really looking for: not containing some hardware we need while simultaneously containing some hardware we don't. The best option that gives us the form factor we are looking for and the functionality is a custom design. Although we were not able to fabricate a custom design for the final presentation, we developed the project with intent of doing so, using the necessary development kits as a proxy.

3.2.2 Connection board options

The sensors will need a way to send the data they collect. In determining the connection options for the sensors, we spoke with ETG for advice on specific technology that will allow the sensors to relay data to and from our MCU. ETG pointed us in the direction of the ESP 8266 which is used in the roombas for the course CPRE 288 which all team members have taken. Taking a further look, the ESP 8266 enables the sensors to communicate via WiFi and WiFi only. It is also inexpensive, small in size, and has very low power consumption [4]. We found plenty of development kits available on the ESP website which allow us to easily test it's functionality.

When further exploring other ESP products we came across the ESP 32. The ESP 32 allows the sensors and MCU to communicate via WiFi and bluetooth [5]. This alone would allow the MCU to utilize bluetooth to communicate with the sensors and relay the data to the cloud via WiFi, unlike the ESP 8266. Furthermore, the ESP 32 is only slightly larger and more expensive than the 8266, however, it has stronger processing power, built-in SRAM and flash memory, and has development kits to allow us to test it's functionality on a breadboard [5]. In the end, we felt the ESP 32 was the best option because of it's dual capability of communicating via bluetooth and WiFi as well as its relatively cheap price, built-in memory, and the development kits available to help us test.

3.2.3 Connection Board Options

The sensors needed a way to send the data they collect. In determining the connection options for the sensors, we spoke with ETG for advice on specific technology that will allow the sensors to relay data to and from our MCU. ETG pointed us in the direction of the ESP 8266 which is used in the roombas for the course CPRE 288 which all team members have taken. Taking a further look, the ESP 8266 enables the sensors to communicate via WiFi and WiFi only. It is also inexpensive, small in size, and has very low power consumption[4]. We found plenty of development kits available on the ESP website which allow us to easily test it's functionality.

When further exploring other ESP products we came across the ESP 32. The ESP 32 allows the sensors and MCU to communicate via WiFi and bluetooth [5]. This alone would allow the MCU to utilize bluetooth to communicate with the sensors and relay the data to the cloud via WiFi, unlike the ESP 8266. Furthermore, the ESP

32 is only slightly larger and more expensive than the 8266, however, it has stronger processing power, built-in SRAM and flash memory, and has development kits to allow us to test it's functionality on a breadboard [5]. Finally, we looked one step further and found STM's WB lineup of ARM microcontrollers. The WB55 variant had a very attractive package at only 7x7 mm, and integrated Bluetooth and WiFi. Being STM, there is a lot of support for it, and a development kit [6]. In the end, we felt the STM32WB was the best option because of its small package which integrates all the wireless technologies we need.

3.2.4 Pulse Sensor options

The first collection of sensors that we explored consisted of pulse sensors. Specifically we looked at two sensors, the MAXREFDES117 and the Pulse Sensor Amped. We chose these sensors because when we approached ETG for guidance, they recommended the Pulse Sensor Amped. After exploring many more sensors we found the MAXREFDES117 to be most appealing due to the fact that it's wearable, small in size, cheap, has low power requirements, and is provided with free algorithms for functionality with arduino microcontrollers. The downsides were that it may require 5.5 VDC for improved accuracy, and it is only compatible with arduino and mbed platforms [7].

When researching more into the pulse sensor amped, we found it to have many similar pros to the MAXREFDES sensor. It is small, wearable, and requires very little power. Furthermore, it comes with processing visualization software to help us test the sensor, it is compatible with many embedded platforms including arduino, and only needs a power, common, and data connection in the hardware design [8]. In the end, we determined the Pulse Sensor Amped to be the most viable option due its similar pros to the MAXREFDES in addition to its simplicity in design, its capability with many platforms, and we can easily consult with ETG for any questions we have since they have experience working with the sensor as well.

For the design we have researched many types of sensors like pulse, weight, and mobility; however, to save time we chose to specifically focus on getting one sensor, the pulse sensor, functioning properly so that when we implement other sensors, the process will be much smoother and quicker.

3.2.5 Battery Options

Each individual sensor and master control unit needs to have its own battery power supply. We decided that it would be best to have replaceable coin cell batteries power the sensors and MCU so that when either runs out of charge, there is minimal time of data loss since the user can replace the used batteries with new ones. In addition, we aim to keep the sensor circuit as simple as possible and having a rechargeable battery would require adding a charging port and battery level monitor.

In the end, we decided to use the CR 2032 coin cells to power the hardware. These coin cell batteries are very popular as you can easily buy them in an electronics store or order them from various online stores. The development kits for the STM32WB55 have a specific coin cell battery slot fit for the CR 2032s. In addition, they are relatively small with a 20mm diameter and height of 3.2mm, provide enough voltage for our circuits(3V) and have a large current capacity of 235 mAh [9].

3.2.6 User Interface

Users will need to read measurements from the sensors and review processed data from the cloud. Our solution involves creating a mobile application and a web application for patients and medical providers. For the web application, we have done some research on frameworks and libraries including React and Vue. React has many advantages over Vue such as having a large support community, having good server performance due to its Server Side Rendering technique, and the non-requirement for extra packages to function [10][11]. For these reasons, we ultimately decided to go with React.

For our mobile application, we have two options: Android [12] or IOS [13]. Ideally, it would be great to have support for the two operating systems but due to time constraint, we decided to go with Android only. Both systems have extensive documentation on creating applications, but our team has much more experience with creating Android applications. Android also has a much larger development community which makes debugging easier.

3.2.7 Cloud Services

Organizations can buy and maintain servers in a physical location. However, in the current era it is much more common to use a cloud provider to outsource these resources. This not only alleviates the burden of having to do normal maintenance, but also provides the benefits of having updated technology at no extra cost. To this end, we will use cloud services to host a remote server.

After doing research into both Microsoft's Azure [14] and Amazon's AWS(Amazon Web Services) [15] we selected AWS. The reasons behind this choice were motivated by three main areas: Scalability, Cost, and Services. Our research showed that AWS offers much wider scalability when compared to Azure. While Azure might work better for smaller projects, AWS has a much greater ability to scale up with the project. For cost, we found that while Azure does offer relatively low costs, AWS's costs were even lower. AWS even offers a limited free tier that fits our teams needs for this project. When looking at services, we found that Azure is starting to invest more into different and unique services. However, AWS has a wider variety of services with easy intercommunication.

3.2.8 Data Analysis/Databases

While researching databases [16], we decided that Amazon DynamoDB would be the best NoSQL database. This is because it gives the ability to store high quantities of data in a flexible way, and NoSQL databases tend to be faster than SQL databases.

We decided on Amazon DynamoDB because it is an excellent NoSQL database, and is serverless. DynamoDB is designed to be highly scalable with relatively low access times. It also eases backups which are completed without performance reduction. The negative side to DynamoDB is that it can only be deployed within AWS, there is an additional storage cost for each item, and it is unable to do complex queries. We did some research on Cassandra but chose not to use it because the access times are slower, and querying does not have joins or subquery support. We also did research on MongoDB and chose not to use it due to its difficulty to secure properly, and it does not have relational database capabilities.

For the data analysis, we decided to use DynamoDB Streams. We believe this is a good option because DynamoDB Streams has the ability to set up triggers when new data is added, existing data is changed or deleted. We can then use this trigger to analyze each data using our data processing code which will be deployed using AWS Lambda.

3.3 Task Decomposition

The following table is a work breakdown summary, assigning task numbers and dependencies to tasks.

Task No.	Task Name	Contributors	Description	Dependencies
1	Hardware	John, Michael	Every hardware-based aspect of the project	
1.1	Sensor IoT Network	John	Scaleable bluetooth network of sensors	
1.1.1	Design Network	John	Create a block diagram for network architecture	
1.1.2	Order parts for network	John	Order sensors and testing dev kit	1.1.1

1.1.3	Assemble network	John	Assemble self-contained sensors	1.1.2
1.1.4	Test IoT network	John	Develop testing code to use as a template on the MCU	1.1.3
1.2	Custom MCU	Michael	Custom circuit that connects sensors to the cloud	
1.2.1	Design MCU architecture	Michael	Create a component diagram and schematic for the MCU	
1.2.2	Order components for MCU	Michael	Order components for the MCU including custom PCB	1.2.1
1.2.3	Assemble MCU components	Michael, John	Solder components to PCB	1.2.2
1.3	Integrate sensor network with custom MCU	Michael, John	Write code for MCU connecting sensor network	
1.3.1	Assemble sensors with MCU	Michael	Write MCU code	1.1, 1.2
1.3.2	Test sensor network with MCU	Michael	Refine Code	1.3.1
1.3.3	Calibrate sensors	Michael, John	Calibrate sensors so they generate accurate measurements	1.3.2
2	Database/Cloud	Richa, Justin	Cloud based aspects of the project	

2.1	AWS	Justin	Amazon Web Services	
2.1.1	Set up AWS server	Justin	Set up an AWS server	
2.1.2	Set up AWS DynamoDB	Richa	Set up AWS DynamoDB database	
2.2	Data Analytics	Richa	Develop AI code to interpret data	2.1
3	User Interface	Isaac, Richa, Michael, Justin	All UI aspects of the project	
3.1	Mobile Application	Isaac	Every mobile-based aspect of the project	
3.1.1	Integrate mobile application with database	Isaac, Richa	Connect mobile app to the AWS database	2.1.2
3.1.2	Integrate MCU with mobile application	Michael, Isaac	Connect mobile app to the MCU	1
3.2	Web Application	Justin, Isaac	Every web-based aspect of the project	2.1.2
3.2.1	Integrate web application with database	Isaac, Justin, Richa	Connect web app to the AWS database	2.1.2
3.3	Test Integration	Isaac, Justin	Test data flow from MCU to database	3.1, 3.2
4	Testing	Team	Test system as a whole, looking for issues in functionality and design	1,2,3

Table 3: Task Decomposition

Table 3 shows how the tasks for our project will be decomposed. Each task is assigned to a team member(s) who will be working on it. The dependencies of each task are listed in the far right column.

3.4 Possible Risks and Risk Management

Risk	**Severity**	**Mitigation Strategy**
High Expenses	High	Use of existing alert system(s) within AWS for notification when a service is reaching a repset usage threshold.
Lack of experience/knowledge	High	When faced with area's within the project where we were lacking experience, we consulted with experts in that area and would do extensive research into that area.
Injury from misuse	Medium	To minimize the injury from misuse, we took extra steps within the design to address possible causes of injury.
Malfunctioning system from normal usage over time	Low	We designed the system to be able to handle different environments and included tests for reliability into our design.
Data security	High	With AWS DynamoDB we were able to utilize the secure connection provided through AWS's user and secrets keys.

Table 4: Risk Mitigation

Table 4 shows the risks that are involved with creating and using our product. Each risk has its own severity and our strategy for dealing with each risk is listed under Mitigation Strategy.

3.5 Project Proposed Milestones and Evaluation Criteria

The main activities and milestones throughout the evolution of this project are illustrated in Figure 7 (Spring 2020 semester) and Figure 8 (Fall 2020 semester).

3.6 Project Tracking Procedures

We used Trello as a method for tracking progression on each task that needed to be accomplished in order to create the final product. This method of SCRUM also documented the progression with time. We also used GitLab to keep track of changes in our code. To easily track changes to master, we used a git webhook in the team discord server to send real-time notifications.

3.7 Expected Results and Validation

Our desired outcome is to have a product that is accurate and secure. The final product needs to be able to gain users' trust the moment it is being presented.

Although we focused on pulse sensors, our testing ensures that we can straightforwardly incorporate other sensors, such as temperature, movement, and body weight. Our main validation consisted of comparing the values detected and displayed by our system with the "brute-force" use of external sensor readings.

4 Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

For our timeline we have split the team into software and hardware. Software is split into three sub-groups: user interface, server, and data.

4.2 Feasibility Assessment

Designing a microcontroller requires a lot of troubleshooting and debugging which can lead to unforeseen delays. Our approach to work around this is to test all sensor functionality with a prebuilt MCU (a STM32WB55 nucleoboard) to ensure the IoT sensor network is functioning properly when we have to test our custom MCU.

4.3 Personnel Effort Requirements

Task	Estimated Time per Week
Cloud application	1 hours
Hardware application	9 hours
Database application	3 hours
Web application	5 hours
Mobile application	6 hours
Testing	25 hours

Table 5: Personnel Effort Requirements

4.4 Financial Requirements and Other Resource Requirement

For this project, we are assuming a budget of $500. Table 6 shows project costs at the end of the Spring 2020 semester as part of our Design Document.

Project Task/Tools	Quantity	Unit Cost ($)	Total Est. Cost ($)	% of Budget
ESP32 dev kits	4	10.99	43.96	8.792
Pulse Sensor	1	24.95	24.95	4.99
Weight Sensor	1	9.00	9.00	1.8
Tag Connect Cable	1	39.99	39.99	7.998
STM32 Microcontroller	2	3.49	6.98	1.396
Custom PCB	2	10	20	4
Micro-B USB connector	2	0.87	1.74	0.348
100 nF caps	10	0.163	1.63	0.326
10 nF caps	10	0.05	0.50	0.1
4.7 uF caps	10	0.206	2.06	0.412
AWS subscription[1]	9 months	0-30/month	0-270	0-54

Total	**190.88 - 460.88**	**38.176 - 92.176**

Table 6: Expected Expenses

We planned to spend at most 92.176% of the budget which leaves us $39.12 to continue adding sensors or eat up unforeseen costs. Because we are not valuing our labor, the surplus in budget would likely go to fund extra hardware in the event that original hardware is broken or becomes unusable.

However, during the implementation phases throughout the Fall 2020 semester, we had adjustments in the estimated costs, as shown in Table X:

Project Task/Tools	**Quantity**	**Unit Cost ($)**	**Total Est. Cost ($)**	**% of Budget**
ESP32 dev kits	4	10.99	43.96	8.792
STM32WB Nucleoboards	2	43.75	87.50	17.5
Pulse Sensor	1	24.95	24.95	4.99
Mobility Sensor[2]	1	0	0	0
AWS subscription[3]	9 months	0-30/month	0-270	0-54
Total			**156.41 - 426.41**	**31.3 - 85.3**

Table 7: Final Expenses

[1]AWS subscription ranges from $0-$30 per month because of the free trial tier. For the scope of this project, we are trying to keep in the free tier. The $30 per month is representative of a small scale deployment with multiple users.

[3]Mobility Sensor was provided free of charge by Iowa State University as part of a collaboration between our group and a graduate research group.

[3]Total AWS feed summed to $30-$35 consuming 6%-7% of the budget (the final month is still in progress).

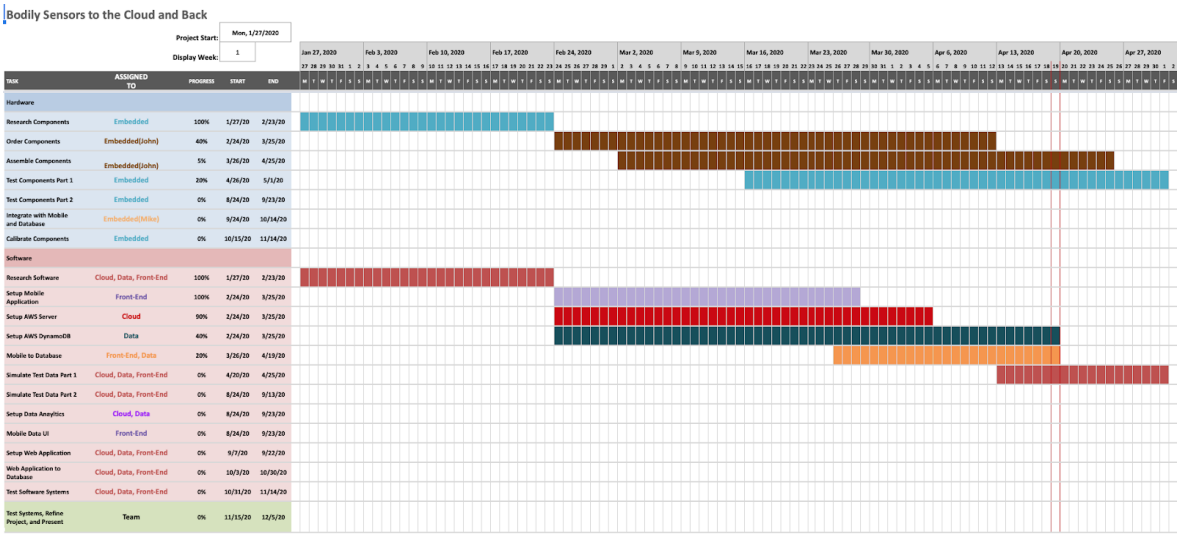

Figure 7: First Semester Gantt Chart

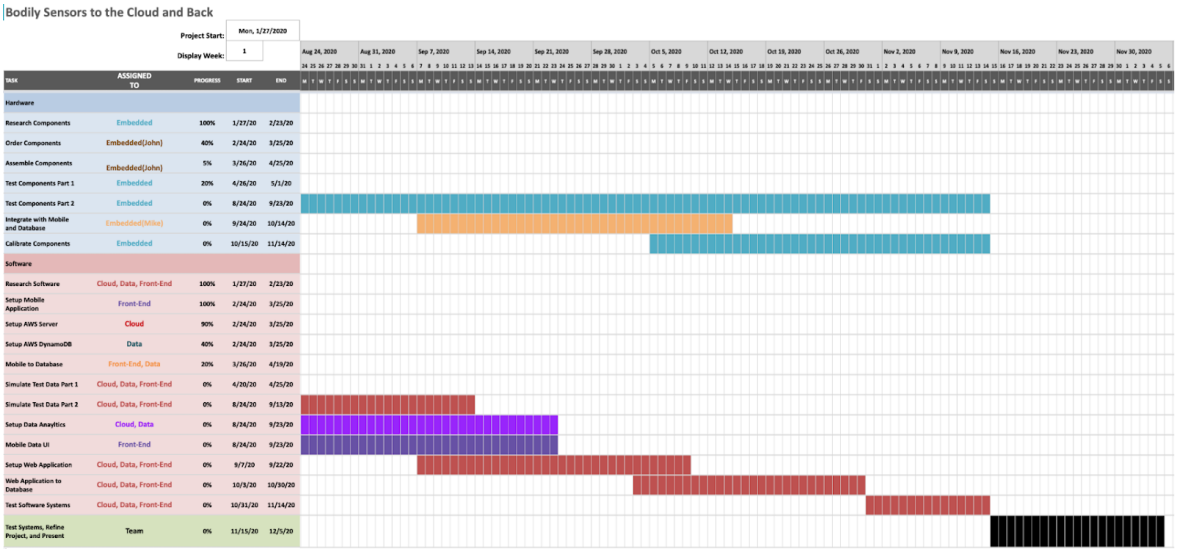

Figure 8: Second Semester Gantt Chart

5 Testing and Implementation

5.1 Interface Specifications

- Functionality between sensors and MCU (John)
- Functionality between MCU and mobile application (Michael & Isaac)
- Connectivity between mobile and database (Isaac & Richa)

- Connectivity between MCU and database (Michael & Richa)
- Functionality between database and cloud (Richa & Justin)
- Functionality between cloud and web application (Justin & Isaac)

5.2 Hardware and software

The following is an itemized list of software tools used in the project.

- Mocha
  Mocha is a library that is used to test JavaScript code. This library can be used to test the functionalities of React. Although Mocha was our chosen library for testing the web app, we did no get far enough with our web app to write automated tests.
- JUnit
  JUnit is used to test Java applications. This unit testing tool will be used to test the functionalities of the mobile application.
- KiCad
  KiCad is a CAD tool for printed circuit boards (PCBs). This tool will be used to design the IoT circuits as well as the MCU PCB.
- STM32CubeMX/Arduino IDE
  The STM32CubeMX and Arduino IDEs are embedded development environments for programming the nucleoboards.

5.3 Functional Testing

- Test sensitivity of the sensors using an oscilloscope. Make sure the sensitivity is not too high where it inhibits data collection and analysis. Adjust filtering or sampling frequency to clean up noisy signals. (John & Michael)
- Test ohmmeter circuit with resistive sensors to make sure the data collection is consistent and contains minimal noise. (John)
- Test the current consumption when sensors are operating to determine battery life of the CR 2032 cells. (John)
- Test bluetooth low energy mesh connections between sensors and MCU using serial messaging applications on our own cell phones and then testing. (John)
- Test to see if the mobile and web applications pull correct data from the database (Isaac & Richa)

- Test if measured data can be displayed on the mobile application live (Isaac)
- Test if the mobile application can push data to the database (Isaac & Richa)
- Test if new users can be added to the database (Isaac & Richa)

5.4 Non-Functional Testing

- Test wear-ability and sensor connection distance to make sure the sensors can connect within at least a 10 foot range but not further than 15 feet. (Entire team)
- Test how long it takes to pull data and display it onto the front end (Entire team)
- Test how fast users can login. The process should take no more than 30 seconds. (Entire team)
- Test the responsiveness of the web application. Visual lagging should be minimal. (Isaac, Richa, & Justin)
- Pen-testing for user data (Isaac & Justin)

5.5 Results

5.6 Hardware

Due to the COVID-19 pandemic causing the temporary closure of all labs on the Iowa State University campus our first semester and limiting the availability during our second semester, we had a limited window to utilize the labs and the equipment provided on campus. During our available time on campus, we tested the Amped pulse sensor and found promising results. The sensor sends an analog signal and we were able to observe our own heartbeats on the oscilloscope at an operating voltage of 5V and 3.3V DC. We then ended up testing the pulse sensor connected to the STM32WB55 nucleo development kit running an arduino analog read program and received the signal shown in Figure 9 which we deemed optimal for sending. The green line represents the user's pulse.

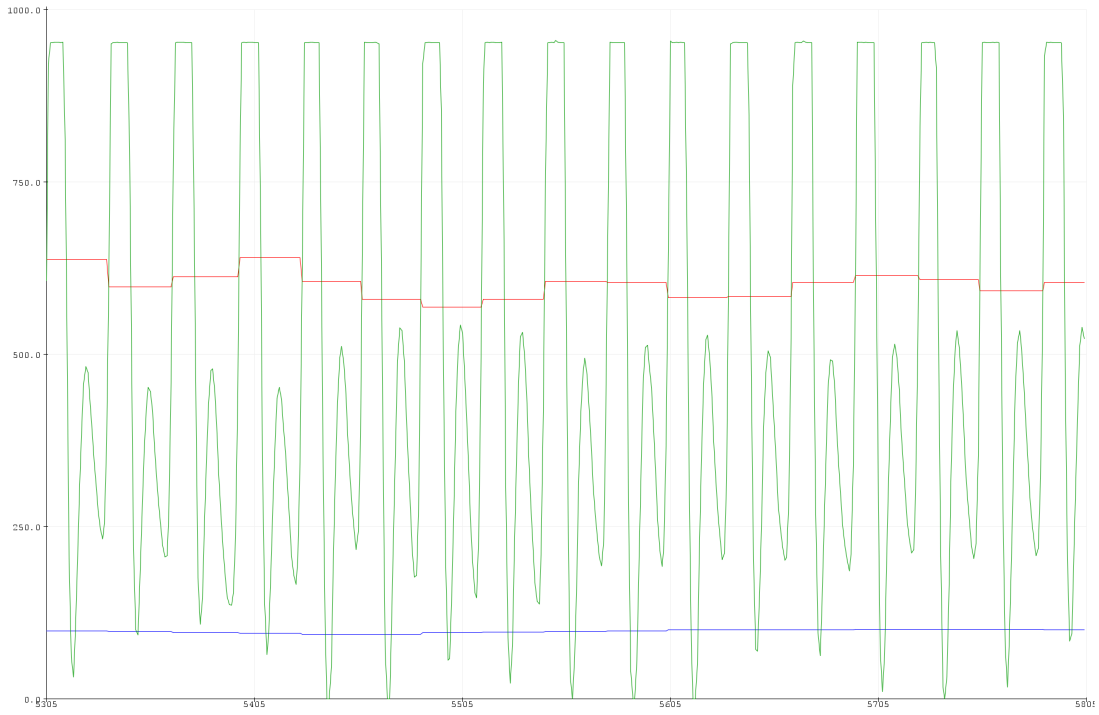

Figure 9: Heart Rate Signal

Furthermore, we conducted testing for battery life by measuring the input current for the operating circuit and dividing by the current capacity of the CR2032 battery cells(235 mAh). Once we had our circuit operating on battery power alone, we used a multimeter to determine the input current to the circuit. We conducted five attempts for averaging the input current to determine the battery life.

Test Attempt	1	2	3	4	5
Current (mA)	11.78	11.76	11.73	11.75	11.72

Table 8: Current Measurements for Pulse Sensor

We then performed the following calculation to determine how long the CR2032 could power the heart rate sensor circuit. We calculated the average current to power the circuit ($\bar{I}$), and then took the current capacity of the CR2032 and divided that by the average current to get the battery life (t_{bat}) in hours.

$$\bar{I} = \frac{11.78 + 11.76 + 11.73 + 11.75 + 11.72}{5} = 11.75 \text{ mA}$$

$$t_{bat} = \frac{235}{11.75} = 20.00 \text{ hours}$$

We conducted the same testing for the mobility sensor circuit. The way the mobility sensor behaves is that as the sensor material stretches and relaxes, the resistance increases and decreases respectively. So the data we gather from the mobility sensor will be the changes in resistance. Our mobility sensor resistance at normal length without stretching was about 5 MΩ so the circuit consisted of our sensor in series with a 10.7 MΩ resistance to conduct voltage dividing calculations that determine the resistance changes. We first set up the STM32WB55 development kit with an Arduino analog read program and connected the mobility sensor to the user's finger, and we were able to get the following signal out from finger movements:

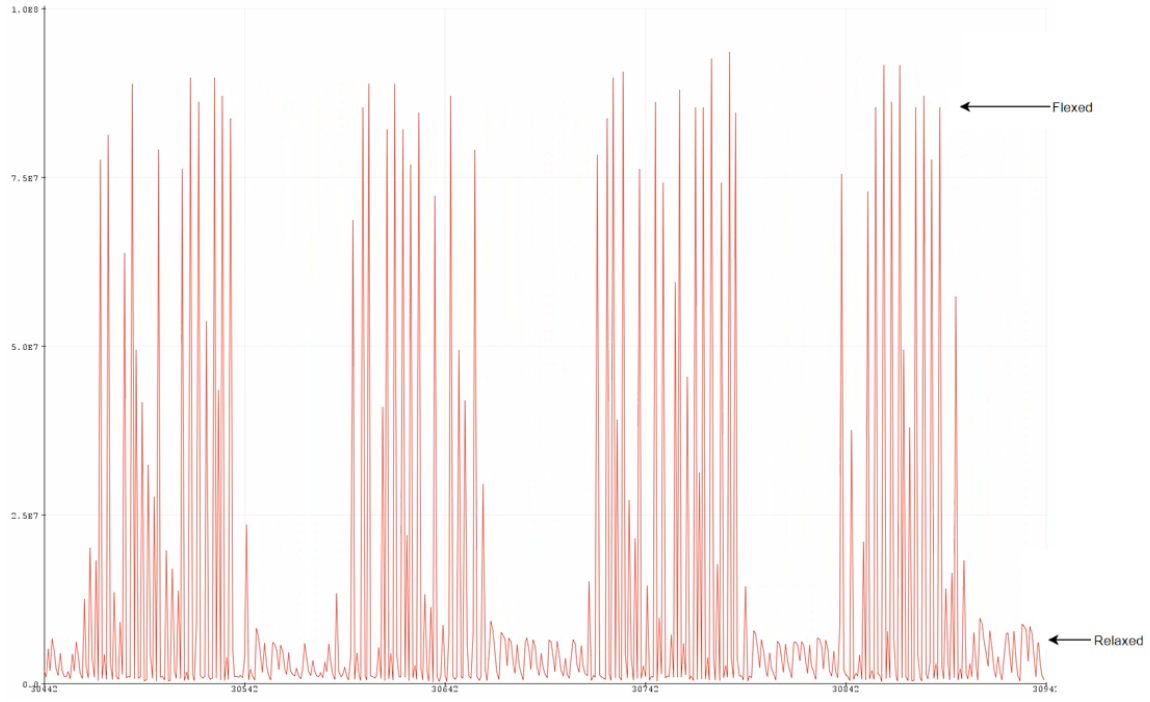

Figure 10: Mobility Sensor Signal

Moreover, we then conducted the same current tests as the heart rate sensor to determine how long a single CR2032 coin cell could power the circuit. We gathered the following current readings from the multi-meter connected at the battery feeding into the circuit during standard operation:

Test Attempt	1	2	3	4	5
Current (mA)	11.30	11.28	11.26	11.29	11.27

Table 9: Current Measurements for Mobility Sensor

These are the battery life calculations for determining the battery life for the mobility sensor circuit. We calculated the average current ($\bar{I}$) to power the circuit, and then took the current capacity of the CR2032 and divided that by the average current to get the battery life (t_{bat}) in hours.

$$\bar{I} = \frac{11.30 + 11.28 + 11.26 + 11.29 + 11.27}{5} = 11.28 \text{ mA}$$

$$t_{bat} = \frac{235}{11.28} = 20.83 \text{ hours}$$

When we tested the pulse and mobility sensor and moved our hand simultaneously, the signal experienced quite a bit of distortion (pulse sensor more than the mobility sensor). The signals will have to be filtered either using software or hardware techniques.

5.7 Mobile and Web Applications

During the development of our mobile and web applications, we tested the functional and non-functional requirements using Manual Testing. Due to time constraints, we were unable to write automated tests.

Tables 10 and 11 show the results of functional and non-functional manual tests.

Requirement	**Test Plan**	**Result**
When selecting a patient on the web app, the web app should pull the correct patient data	Create a set of test patients and users within the database. Sign in with different users to verify the results	The web app displays the correct data for a selected patient
When the MCU sends sensor data to the mobile app over bluetooth, the mobile app should correctly display the sensor data	Create a list of sample sensor data and use the LightBlue app to send the data to the mobile app	Mobile app correctly decodes the sensor data and displays the correct values on the UI

When the mobile app receives sensor data from the MCU, the mobile app should push the correct sensor data to the database	Use the LightBlue app to send sensor data to the mobile app and manually check DynamoDB for each sensor data	Each sensor data is correctly stored in the correct table in DynamoDB
When a user signs up using the mobile app, the correct user information should be stored in the database	Create multiple users using the mobile app and manually check DynamoDB for each user	Each user information is correctly stored in the correct table in DynamoDB

Table 10: Functional Tests

Requirement	**Test Plan**	**Result**
When selecting a patient on the web app, it should not take more than 30 seconds for it to display the patient's data	Create different patients within the database with varying amounts of sensor data. Manually verify that the loading times are within viable ranges	Each user's data is displayed in less than 5 seconds
When users login on the web app, it should not take longer than 30 seconds to login	Login into the web app from multiple browsers with different users	Logging into the web app multiple times all took less that 5 seconds to access the main screen
When users login on the mobile app, it should not take longer than 30 seconds to login	Login to the mobile app multiple times using different accounts and check how long it takes to get to the home screen	Logged in to the mobile app multiple times using different accounts and each login takes less than 2 seconds

Table 11: Non-Functional Tests

5.8 Integration Testing

Due to COVID, we were unable to do integration tests between our MCU and our mobile app. Instead, we have been using LightBlue to test our mobile app as mentioned in the Functional table above.

6 Conclusion

We have designed the MCU/sensor IoT network with the STM32WB55 nucleo development kits. The goal of this design is to interface the MCU with our sensors in an IoT environment and have the MCU send data from the sensors to the cloud and a mobile application where the user can then view the data and analytics from a mobile or web application.

To accomplish these goals in the time frame we had, our development groups worked concurrently. While the hardware team designed the MCU and sensor network, the cloud team set up infrastructure with AWS and implemented stubs to ensure communication between the cloud and the web application is running smoothly. Developing in parallel reduced the time cost of this project and gave us more time for testing.

Our end product allows medical professionals to analyze many vital functions of multiple patients at once with ease using our sensor network design. The information will be stored and interpreted using the cloud services we set up with our mobile and web applications. This will make the lives of medical professionals much less stressful when monitoring their patients. In addition, our product will give more patients the freedom to leave a medical center or hospital while receiving care if they wish to do so, and if they are given permission by their doctors, since our product is completely portable.

References

[1] "John A. Stankovic," *University of Virginia School of Engineering and Applied Science*, 06-Dec-2019. [Online]. Available: `https://engineering.virginia.edu/faculty/john-stankovic`. [Accessed: 25-Apr-2020].

[2] Sam Burnett - ETG, personal communication, Feb. 2020

[3] Teel, John, "How to Design Your Own Custom Microcontroller Board", *Predictable Designs*, 15-May-2018. [Online]. Available: `https://predictabledesigns.com/tutorial-how-to-design-your-own-custom-microcontroller-board-video-part1/`. [Accessed: 12-Feb-2020].

[4] Espressif Systems "ESP8266EX" ESP8266EX datasheet, Dec. 2015 [Revised Apr. 2020].

[5] Espressif Systems, "ESP32 Series," ESP32 datasheet, Aug. 2016 [Revised Jan. 2020].

[6] ST Microelectronics, "Datasheet - STM32WB55xx," STM32WB55xx datasheet, Oct. 2018 [Revised Jul. 2020].

[7] Maxim Integrated, "MAXREFDES117#: Heart-Rate and Pulse-Oximetry Monitor," System Board 6300 datasheet, Aug. 2016.

[8] Sparkfun Electronics, "Pulse Sensor," *Sparkfun Electronics*, SEN-11574. [Online]. Available: `https://www.sparkfun.com/products/11574`. [Accessed: 5-Feb-2020].

[9] Energizer Product Data Sheet "Energizer CR2032" Energizer CR2032 datasheet, Jun. 2018 [Revised Apr. 2020].

[10] Mahmood, Hamza. "Advantages of Developing Modern Web Apps with React.js" *Medium*, Medium, 7 Sept. 2018 [Accessed: 19-Apr-2020].

[11] "Overview" *Overview - vue.js*. [Online]. Available `https://v1.vuejs.org/guide/overview.html`. [Accessed: 19-Apr-2020].

[12] "Meet Android Studio: Android Developers," *Android Developers*. [Online]. Available: `https://developer.android.com/studio/intro`. [Accessed: 19-Apr-2020].

[13] "Start Developing iOS Apps (Swift): Jump Right In", 08-Dec-2016. [Online]. Available: `https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/`. [Accessed: 23-Apr-2020].

[14] "Cloud Computing Services: Microsoft Azure," *Cloud Computing Services | Microsoft Azure*. [Online]. Available: `https://azure.microsoft.com/en-us/`. [Accessed: 19-Apr-2020].

[15] "Cloud Computing Services: Amazon Web Services," *Amazon Web Services (AWS) - Cloud Computing Services*.[Online]. Available: `https://aws.amazon.com/`. [Accessed: 19-Apr-2020].

[16] Amazon Web Services, Inc 2020. *Amazon Dynamodb - Overview*. [online] Available at: `https://aws.amazon.com/dynamodb/` [Accessed 19 April 2020].

[17] Kumar, P., 2020. *Amazon Dynamodb Tutorial - A Complete Guide | Edureka Blog*. [online] Edureka. Available at: `https://www.edureka.co/blog/amazon-dynamodb-tutorial#What-Is-DynamoDB?` [Accessed 19 April 2020].

Appendix A

Market Survey

John A. Stankovic - BP America Professor at the University of Virginia
Sarah A Radke - Nurse Practitioner Specialist in Whitefish Bay, WI
Jessica Terrell - Registered Nurse at Life Care Center of Jacksonville

MCU Programming References

The GNOME Project, *GTK+ 3*, (2020). Github Repository: `https://github.com/GNOME/gtk`. [Accessed: 09-Apr-2020].

The GNOME Project, *Glib*, (2020). Github Repository: `https://github.com/GNOME/glib`. [Accessed: 09-Apr-2020].

Boyini, Karthikeya, "Enum in C," *tutorialspoint.com*, 05-Nov-2018. [Online]. Available: `https://www.tutorialspoint.com/enum-in-c`. [Accessed: 09-Apr-2020].

Carnegie Mellon University, "C Coding Standards," *Carnegie Mellon University*, 27-Apr-2004. [Online]. Available: `https://users.ece.cmu.edu/~eno/coding/CCodingStandard.html`. [Accessed: 16-Apr-2020].

Appendix B

Operations Manual

Sensors and MCU

1. Take your sensor circuit assembly and make sure the wiring connections are secure, and apply the sensor.
   (a) For the pulse sensor, make sure the side with the white heart figure is facing towards the tip of your finger directly on top of your finger print and secure it (lightly but enough to remain fastened to your finger) with either tape or velcro.
   (b) For the mobility sensor, make sure whichever joint the sensor is applied to does not come in direct contact with your skin as the sensor has powdered graphene and may cause irritation, and secure one end of the sensor above the joint and one below. When your joint is straight, the sensor should have a little slack to it.
2. Take your sensor assembly and place a CR2032 coin cell battery in the black battery holder and close the holder.
3. On the battery holder there is a switch near the wiring, the switch position closest to the wires is on, and further away from the wires is off. Flip the switch in the on position and you will see a red LED turn on on the circuit.
4. Take your MCU phone attachment and place a CR2032 cell battery in the battery holder and flip the switch to the same position and you will see a red LED turn on on the MCU.
5. You will now see the devices BLE connections via the mobile app.

Mobile Application

A mobile phone with an Android operating system is required to use the application. Since we were not able to do integration testing with our MCU, we recommend using the LightBlue app to test the full functionality of the mobile app. Android Studio is also required since we could not release a proper APK of the mobile app.

1. Clone our project repository to your Windows or Mac machine.
   `https://git.ece.iastate.edu/sd/sddec20-26`
2. Open Android Studio and import the folder *BodilySensorBLE*.
3. Open the LightBlue app and create a virtual BLE device (e.g Blood Pressure).

(a) Click on the new device that you just created.

(b) Scroll down until you see the device.

i. Click the blue icon.

ii. Copy the 4-digit *Service UUID*.

(c) Click *Option* and add a *Characteristic*.

i. Select the option below *Device Information*.

(d) Select the *characteristic* that you just created.

(e) Copy the characteristic *UUID*.

(f) Under *Property*, make sure *Indicate* is selected.

4. In Android Studio, navigate to DetailActivity.java.

(a) There, you'll find a variable named *SERVICE_ UUID*. Type in the following UUID and replace with the 4-digit *Service UUID* that you copied.

i. 0000-0000-1000-8000-00805f9b34fb

(b) You'll also find a variable named *READ_ UUID*. Type in the characteristic *UUID* that you copied.

(c) Connect your Android device to your machine and run the app.

5. On the mobile app, create and account and login.

(a) On the home screen, you can scan for nearby BLE devices. Click *Scan*.

(b) From the list, find the virtual BLE device that you created and click *Connect*.

(c) Click on the connected device.

6. Using our custom data frame, create some sample data and send it using the LightBlue app.

(a) Here is a 60BPM sample data
`030200404e00000000000000`

Web Application

The web application uses React-Bootstrap with different dependencies including, but not limited to, firebase, react-router-dom, aws-sdk, and react . You will need to have Node.js and NPM, which stands for node package manager, installed.To edit the web app you will need and IDE of your choice and access to the git repository. Once you have gathered all those resources you will need to follow the steps given below.

1. Navigate to the directory where you clone the repository from your terminal and run 'npm install' to install the project's dependencies.

2. After all the dependencies have finished installing you just need to run "npm start" from your terminal to launch the application. Once the server is running locally it will automatically open a new browser window or a new tab within an already running browser.

3. Navigate to the new tab. From there you will either need to sign-in, assuming you have already signed-up, or sign-up for the first time.

4. Once a provider is signed-in there would be a list of their patients on the Home screen. They would be able to navigate by clicking on the patient's name to a detailed page containing the patients sensor details.

Setup Manual

Sensors and MCU

1. The sensor assembly has three components:
   (a) The battery holder
   (b) The STM32WB55 board
   (c) The sensor

2. For setting up the pulse sensor:
   (a) Connect the red wire to the 5 Volt slot on the STM32WB55 board.
   (b) Connect the Black wire to the GND slot on the STM32WB55 board.
   (c) Connect the purple wire to the A0 slot on the STM32WB55 board.

3. For setting up the mobility sensor:
   (a) Connect one end of the mobility sensor to 5V slot on the STM32WB55 board.
   (b) Connect the other end of the mobility sensor to the voltage divider circuit.
   (c) Connect the free end of the voltage divider circuit to GND.
   (d) Where the sensor connects to the voltage divider circuit, take the free wire and connect that to the A0 slot on the STM32WB55 board.

4. The MCU assembly has two components:
   (a) The battery holder
   (b) The STM32WB55 board

5. Take the battery holder and connect the black wire to the GND slot on the STM32WB55 board and the red wire to the 3V slot.

Appendix C

Other Considerations

The biggest thing we learned from working on this project is that communication within a team is key. At the beginning of this project we were able to meet in person at different locations around campus. This changed drastically as with the start of the COVID-19 pandemic in mid March. Since then we have learned to work as a team from a distance. We have experienced the difficulty of sharing hardware between members when in isolation. We had members exposed to the virus and were obliged to remain in quarantine to prevent the spread which greatly hindered progress during the testing phases that required meetings face-to-face.Furthermore, we learned many new technologies we have not worked with before, so it was quite a learning experience.

Appendix D

Below is a URL for our project repository.
`https://git.ece.iastate.edu/sd/sddec20-26`