University of Manitoba
Department of Electrical & Computer Engineering

ECE 4600 Group Design Project

Final Project Report

Design of a Framework for an Adaptable Body Sensor Network

by

Group 06

Victor Anzala  
David Homeniuk  
Sean Iftody  
Michael Miner  
Luther Ng  
Parvinder Pawar

Academic Supervisor
Dr. Ken Ferens

Industry Supervisor
Dr. Paul Card – Canadian Tire Co.

Date of Submission
March 4, 2015

Copyright © 2015 Victor Anzala, David Homeniuk, Sean Iftody, Michael Miner, Luther Ng, Parvinder Pawar
Abstract

The purpose of this project was to design and implement a framework for a Body Sensor Network (BSN). Creating a BSN framework involves multiple engineering challenges and requires development of the following three components: sensors, a relay device, and a storage system. By designing a BSN framework, we are providing a base for future developers to build upon for their specific needs. Our BSN framework allows for the implementation of a hardware application, Phone Application and a server. Our framework is designed to be used in environments in which an individual may need to be constantly monitored. Our framework is robust, adaptable, and upgradeable.
## Contributions

<table>
<thead>
<tr>
<th></th>
<th>Victor Anzala</th>
<th>David Homeniuk</th>
<th>Sean Ifody</th>
<th>Michael Miner</th>
<th>Luther Ng</th>
<th>Parvinder Pawar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Microcontroller Software</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>PCB Design</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Phone Interface</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Phone Bluetooth</td>
<td>○</td>
<td></td>
<td></td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Phone Data Transmission</td>
<td>○</td>
<td></td>
<td></td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Database</td>
<td>○</td>
<td></td>
<td></td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Web Portal</td>
<td>○</td>
<td>○</td>
<td></td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Phone Listener</td>
<td>○</td>
<td></td>
<td></td>
<td>○</td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>BLE Integration</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td>○</td>
</tr>
<tr>
<td>Phone-Server Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
</tr>
</tbody>
</table>

Legend:  ● Lead task  ○ Contributed
Acknowledgements

Throughout our project, there have been many individuals that aided us by providing information, and helping us construct hardware components. We would like to thank the following individuals for their help in the completion of our project:

- Dr. Ferens for creating the project and continually providing feedback over the course of our project.

- Zoran Trajkoski for his insight in the design of our demonstration board, and for his help soldering some the components onto our demonstration board.

- Sinsia Janjic for ordering our parts and giving us insight into our components.

- Shamir Nazir Muki for allowing Michael to have fridays off to work on our project, and for allowing Michael to use time and resources at work to gain the insight to solve some of our hardest problems.

- Stack Overflow Community for their advice and insight into developing and troubleshooting Android applications.

- Bluegiga Technologies Ltd. for helping troubleshoot our BLE112 module application incorporated into our design.

- Aidan Topping, for her help reviewing and editing our reports.
# Table of Contents

Abstract ......................................................... i
Contributions ................................................... ii
Acknowledgements ............................................... iii
List of Figures ................................................... vii
List of Tables ................................................... ix
Glossary .......................................................... x

1 Introduction ..................................................... 1
   1.1 Project Overview ......................................... 1

2 Hardware Application ........................................... 3
   2.1 Requirements ............................................. 3
      2.1.1 Microcontroller ...................................... 3
      2.1.2 Sensors ............................................... 4
      2.1.3 Bluetooth ............................................ 4
      2.1.4 Power Management .................................... 4
   2.2 Design and Component Selection ......................... 4
      2.2.1 Microcontroller ...................................... 5
      2.2.2 Sensors ............................................... 5
      2.2.3 Bluetooth ............................................ 6
      2.2.4 Power Management .................................... 7
# Table of Contents

## 2.2.5 PCB Design

- 2.2.5.1 Prototype PCB .................................................. 10
- 2.2.5.2 Demonstration PCB ............................................. 10

## 2.3 Implementation and Integration

- 2.3.1 Microcontroller .................................................... 11
- 2.3.2 Sensors .............................................................. 19
- 2.3.3 Bluetooth ........................................................... 20
- 2.3.4 Power Management ................................................. 21

## 3 Phone Application

- 3.1 Design Decisions ...................................................... 22
  - 3.1.1 Development Platform .......................................... 22
  - 3.1.2 Development Environment ..................................... 23
- 3.2 Implementation ....................................................... 24
  - 3.2.1 Bluetooth .......................................................... 24
  - 3.2.2 Interface ........................................................... 27
  - 3.2.3 Server .............................................................. 35
- 3.3 Integration ............................................................ 36
  - 3.3.1 Hardware Application .......................................... 36
  - 3.3.2 Application Server ............................................. 38

## 4 Server

- 4.1 Requirements .......................................................... 40
- 4.2 Design Decisions ...................................................... 40
  - 4.2.1 Database ........................................................... 41
  - 4.2.2 Web Application .................................................. 42
  - 4.2.3 Phone Listener .................................................... 42
# Table of Contents

4.3 Implementation ................................................................. 43

  4.3.1 Database ................................................................. 43

  4.3.2 Web Application ............................................................ 48

  4.3.3 Phone Listener ............................................................. 60

  4.3.4 Remote Server ............................................................. 62

4.4 Integration ................................................................. 63

5 Conclusion ................................................................. 65

References ................................................................. 67

Appendix A Budget ................................................................. 70

Appendix B Prototype PCB Layout ................................................................. 72

Appendix C Prototype PCB Schematic ................................................................. 73

Appendix D Demonstration PCB Layout ................................................................. 74

Appendix E Demonstration PCB Schematic ................................................................. 75

Appendix F Curriculum Vitae ................................................................. 77
List of Figures

1.1 Abstract overview of the developed platform framework for a Body Sensor Network. ........................................ 2
2.1 Steps executed in setup when the program begins to run. ................................................................. 12
2.2 Steps executed when initializing the BLE112 module. ............................................................................. 14
2.3 Steps executed when update in the LM74 library is called. ................................................................. 15
2.4 Code executed when initializing the MMA8452Q accelerometer. ......................................................... 16
2.5 Behavior of loop in our application example. ....................................................................................... 17
2.6 Flow of function calls required to get accelerometer data. ................................................................. 18
2.7 Behavior of BLE112 response handler. ................................................................................................. 19
3.1 Relationship of the Phone Application with the framework. ............................................................... 27
3.2 Relationship between the Phone Application Activities pages. ......................................................... 28
3.3 Configuration of credentials for the Phone Application. ..................................................................... 29
3.4 Phone Application Bluetooth device selection screen. ....................................................................... 30
3.5 Phone Application settings screen. ......................................................................................................... 31
3.6 Phone Application home screen. ........................................................................................................... 32
4.1 An overview of the server with details on communication between its multiple sub-systems. ................................. 39
4.2 An Overview of the Database illustrating the schema, the four tables, and the columns they contain. ................................................................. 44
4.3 A Flowchart illustrating the logic of the Web Application when permitting logins. ............................... 45
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>A Flowchart illustrating the logic of the Web Application when retrieving sensor readings from the SensorData table.</td>
</tr>
<tr>
<td>4.5</td>
<td>An overview of the Web Application detailing the contents of: the Web Server and its Packages, the Web Site and its pages, and communications to and from internal and external components.</td>
</tr>
<tr>
<td>4.6</td>
<td>A Flowchart illustrating the logic of the UserBean for user login and logout.</td>
</tr>
<tr>
<td>4.7</td>
<td>A Flowchart illustrating the logic of the SensorBean for Sensor Reading retrieval.</td>
</tr>
<tr>
<td>4.8</td>
<td>A Flowchart illustrating the logic of the GrootBean and AdminBean.</td>
</tr>
<tr>
<td>4.9</td>
<td>A Flowchart illustrating the logic of the Web Application during startup.</td>
</tr>
<tr>
<td>4.10</td>
<td>An Overview of the Phone Listener detailing the contents of the Phone Listener Server and its Package, and communications to and from external components.</td>
</tr>
<tr>
<td>4.11</td>
<td>An illustration of the format of a Sensor Reading when transmitted by the Phone Application to the Phone Listener.</td>
</tr>
<tr>
<td>4.12</td>
<td>Phone Application/Server Communication Standard.</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.I</td>
<td>Maximum current and power consumption of major components.</td>
<td>8</td>
</tr>
<tr>
<td>4.I</td>
<td>A breakdown of the different Web Application features available to each tier.</td>
<td>51</td>
</tr>
</tbody>
</table>
Glossary

**Activity**: A screen in Android that allows user interaction

**Admin**: An administrative user

**AdminBean**: A Bean used to accept an administrators on page inputs

**ADT**: Android Development Tools

**API**: Application Program Interface

**Bean**: A java class which parses input and output for the user

**BGAPI**: Bluegiga Application Program Interface

**BGScript**: Bluegiga Script

**BLE**: Bluetooth Low Energy

**BSN**: Body Sensor Network

**CSS**: Cascading Style Sheets

**DAO**: Data Access Object

**EAGLE**: Easily Applicable Graphical Layout Editor

**Fragment**: A piece of an on screen interface

**GATT**: Generic Attribute Profile

**Groot**: The top level system administrator

**GrootBean**: A bean used to accept a groot users inputs on page

**HTML**: HyperText Markup Language

**HTTP**: Hyper Text Transfer Protocol
HTTPS: Hyper Text Transfer Protocol Secure
I²C: Inter-Integrated Circuit
IDE: Integrated Development Environment
JDBC: Java DataBase Connectivity
JSF: JavaServer Faces
JSP: JavaServer Pages
JTAG: Joint Test Action Group
LED: Light Emitting Diode
LoggedDAO: A Data Access Object used to manipulate a datatable storing logged in users
LoggedIn: A datatable in the database tracking users logged in
MAC: Media Access Control
MD5: Message-Digest 5
PHP: PHP: Hypertext Processor
Phone Application: A program running on a mobile phone
Phone Listener: A Tomcat 8 server running the WebSocket API
RPC: Remote Procedure Calls
RX: Receive
SensorBean: A Bean used to show sensor data on the webpage
SensorDAO: A Data Access Object used to manipulate sensor data
SensorData: Data received from the phone application
SensorThreshold: A data table consisting of set safe sensor thresholds
SensorThresholdDAO: A Data Access Object used to manipulate sensor thresholds
SPI: Serial Peripheral Interface
SRAM: Static Random Access Memory
TWI: Two-Wire Interface
TX: Transmit
UART: Universal Asynchronous Receiver/Transmitter

UI: User Interface

URL: Uniform Resource Locator

UserBean: A bean used to process user inputs, handles a users session

UserDAO: A Data Access Object used to manipulate user data

User: Someone registered with the system

Users: A data table consisting of all registered users

UUID: Universally Unique Identifier

WAMP: Web Application Messaging Protocol

Web Application: A basic website allowing user interaction with the database

Wi-Fi: Wireless Fidelity
1. Introduction

With advancements in technology, BSNs have become beneficial in many environments. For example, in the medical industry, the use of BSNs for the monitoring of a patient’s well being is crucial to the facilitation of care. Patients are given body sensors to wear to monitor conditions such as heart failure, diabetes, and asthma. These body sensors then access a network and transmit the data they have collected to a server, for permanent storage and presentation to a qualified individual upon request. This improves doctor-patient interaction, allowing the doctor to have a precise record of the patient’s history, and as a result, give better advice specifically suited to the needs of that patient. BSNs are also beneficial to oversee the well-being of individuals in a hazardous setting, such as firefighters and construction workers. In these hazardous settings there may be events where temperatures can reach dangerous levels, or volatile gases may unknowingly be present. Individuals that are equipped with body sensors can be promptly informed of such events and take measures accordingly.

There is a growing emphasis on the use of BSNs, and for these BSNs to perform in an efficient and productive manner, rational development is needed. Developers of BSNs are confronted with many engineering challenges such as energy efficiency, cost, and ease of development. Furthermore, developers need to design and implement the same common components for their BSN such as: sensors, a relay device, and a storage system. Our project has produced a framework that a developer can use to create a custom BSN.

1.1 Project Overview

Our project was to create a BSN framework composed of a server, Phone Application, and microcontroller to monitor data received from body sensors, as seen in Figure 1.1
We have developed a microcontroller that reads and transmits data from body sensors via Bluetooth Low Energy (BLE) to the Phone Application. Depending on the availability of a network connection, the Phone Application will either store the data locally and transmit the data when a connection is available, or continually push the data to the server at predefined intervals. The rate of data transmission is left up to the developer to suit the developers specific application needs. Once the server receives the data, it stores and displays the data in a user friendly manner.
2. Hardware Application

The core purpose of our project is to create a framework which will act as a foundation for developers. We designed and built a hardware application as an example to show the functionality of our framework. The hardware application consists of two portions: the printed circuit board (PCB), and the software application.

The purpose of this hardware application is to show that the framework functions as intended, and to show that the components selected for the hardware application are interchangeable. We want developers to be able to select components of their choosing, while still being able to fully integrate selected hardware components with our framework. We will begin our discussion of the hardware application by outlining our requirements, followed by the design decisions, implementation, and finally integration of the major hardware and software components.

2.1 Requirements

In this section, we will discuss the requirements which were set for the microcontroller, sensors, Bluetooth, and power management components of the hardware application.

2.1.1 Microcontroller

The microcontroller has two main requirements. The first requirement is that the microcontroller must have at least one Inter-Integrated Circuit (I²C), and one Serial Peripheral Interface (SPI) port, to allow support for any sensors that a developer may use. The second requirement is that the microcontroller must be able to support Bluetooth communication.
2.1.2 Sensors

For the purposes of our hardware application, we must demonstrate the functionality of our sensors using the I²C and SPI protocols. Due to this constraint, we were required to select only sensors which communicated through the I²C and SPI protocols.

2.1.3 Bluetooth

The two main requirements we established for selecting the type of Bluetooth communication are: the maximum range of the Bluetooth connection, and the power consumption required to transmit data. An auxiliary requirement is that the Bluetooth must function at a minimum range of 5 meters.

2.1.4 Power Management

A battery is required to power the demonstration PCB. The battery must be able to continuously power the demonstration PCB for approximately 8 hours, the length of an average working day. Acquiring a battery that operates for exactly 8 hours is not crucial. The time of operation for the hardware application is application specific, and will vary based on what the hardware application is being used for. The battery must be small enough so that it does not increase the size of the demonstration PCB. In the event that the battery fails or has no charge left, and a replacement battery is not available, a secondary power source is required to power the demonstration PCB.

2.2 Design and Component Selection

In this section, we outline the design decisions made during the selection of the microcontroller, sensors, Bluetooth, and power management components for the hardware application.
2.2.1 Microcontroller

The selected microcontroller must be compatible with the Arduino Framework. The Arduino Framework provides us completed and abstracted low level code for certain microprocessors manufactured by Atmel, as well as hardware designed by the Arduino community. A microcontroller compatible with the Arduino Framework allows us to put more focus towards developing a proof of concept for the functionality of our developed framework. Our initial decision was to use the ATSAM3X8E, a 32-bit ARM based microcontroller, which is currently being used on the Arduino Due. We already had access to an Arduino Due, so it was a cost effective decision to use the Due for the development of our hardware application. However, we had to switch to another microcontroller due to issues with the ATSAM3X8E microcontroller being out of stock and lead times being indefinite. We decided to use the ATSAM3X4E. The ATSAM3X4E has the same internal architecture as the ATSAM3X8E, with the exception that the ATSAM3X4E has half the amount of available flash memory and static random access memory (SRAM) \[2\]. For our purposes, the drop in memory between the ATSAM3X8E and the ATSAM3X4E had no consequences.

The Arduino Framework also gives users a free software development platform to write, compile, and upload programs to supported microcontrollers, and is even supported on different free Integrated Development Environments (IDE). Utilizing the Arduino Framework cuts costs, as many development tools for embedded systems, such as debuggers and licenses for IDEs, are expensive.

2.2.2 Sensors

The decision to demonstrate the functionality of the I\(^2\)C and SPI protocols with our application limited us to the selection of sensors which would use those protocols. We chose an accelerometer and a temperature sensor because they are simple to demonstrate. The accelerometer allows us to provide some motion or g-forces to the demonstration PCB to demonstrate its operation. The temperature sensor is simple to demonstrate because its readings can be compared to the room temperature, or a finger can be placed on top of the sensor and, as a result, the readings of the
temperature sensor will increase.

Our initial selection for an accelerometer was a fully constructed breakout board housing the LIS331. The LIS331 was selected because it came soldered and implemented on a PCB. This was initially thought to save time because we would not have to map out the connections of the accelerometer onto a preliminary PCB for testing and development. The LIS331 also allowed for communication through either the use of I\(^2\)C or SPI protocols, so it fit our constraints. However, due to the price of the fully assembled breakout board, $34.96 CAD, we decided to look into purchasing the sensor separately. The price of the sensor was expensive, at $7.08 CAD; as a result we found another sensor, the MMA8452Q, with similar functionality to the LIS331 but at a lower cost. The MMA8452Q accelerometer was $1.72 CAD. The only noticeable difference between the LIS331 and the MMA8452Q is that the MMA8452Q only communicates over the I\(^2\)C protocol.

The initial selection for a temperature sensor was the TMP102, which came fully implemented on a breakout board at a cost of $7.44 CAD. The TMP102 communicates over I\(^2\)C, and while we were able to use the TMP102 alongside the LIS331, since the LIS331 communicates over either I\(^2\)C or SPI, we were not able to use the TMP102 alongside the MMA8452Q because the TMP102 only communicates over I\(^2\)C. Due to this incompatibility, we chose the LM74 as the temperature sensor because it communicates over SPI, and is of a reasonable cost at $2.12 CAD.

### 2.2.3 Bluetooth

Since the Bluetooth module will be sending information to the Phone Application, we assumed that the phone would be in close proximity to the individual that the hardware application is placed on. Due to this reason, we were required to choose from one of the three classes of devices for Bluetooth: class one, which has a range of roughly one hundred meters; class two, which has a range of roughly ten meters; and class three, which has a range of roughly one meter [3]. Class one was immediately eliminated due to the range being longer than required. This long range also meant that the power required to transmit would be much larger. Class three was eliminated due
to the range being too short. This left us with class two which had the optimal range, allowing the phone to be on or near the user of the hardware application. This intermediate range also meant that the Bluetooth would not consume unnecessary amounts of power when transmitting to the Phone Application.

As Bluetooth power consumption was an important concern, we decided that we would use the BLE protocol. The major difference between the classic version of Bluetooth and BLE is that BLE consumes much less power, and as a result, the battery life of the hardware application can last much longer [4].

Due to the SPI and I²C ports being reserved for other sensors that can be added to the hardware application, it was determined that the BLE module should also be able to communicate to the microcontroller through the Universal Asynchronous Receiver/Transmitter (UART) protocol. The BLE112 fits our needs, as it is a class two Bluetooth device, supports the BLE protocol, and communicates with external processors via UART [5].

### 2.2.4 Power Management

After initial research, it was determined that all of the components on the demonstration PCB would require a voltage of 3.3 volts to operate. However, the demonstration PCB was also determined to have components which required a voltage larger than 3.3 volts, such as Light Emitting Diodes (LED) which require a voltage of approximately 5 volts for proper operation. This required us to have two different levels of voltage on our demonstration PCB. Therefore, we required a voltage regulator to regulate the higher voltage level down to 3.3 volts. We chose a small coin cell battery to power the demonstration PCB. The size of the demonstration PCB is important because it is placed on a user’s body. Therefore, a coin cell battery would best fit the requirement of size, as well as powering the PCB. The coin cell battery must have a capacity large enough to power our demonstration PCB for a period of 8 hours. The calculation for the battery’s capacity was done using the following equation:
\[ \text{Capacity} (Ah) = \frac{\text{DeviceWattage}(W) \times \text{OperationTime}(\text{Hours})}{\text{Voltage}(V)} \]  \hspace{1cm} (2.1)

Table 2.1 shows the maximum current consumption of the major components on our demonstration PCB as well as the consumed power.

Table 2.1: Maximum current and power consumption of major components.

<table>
<thead>
<tr>
<th>Component</th>
<th>Max Current Consumption (mA)</th>
<th>Supply Voltage(Volts)</th>
<th>Power used (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE112</td>
<td>36</td>
<td>3.3</td>
<td>118.8</td>
</tr>
<tr>
<td>LM74</td>
<td>0.52</td>
<td>3.3</td>
<td>1.716</td>
</tr>
<tr>
<td>MMA8452Q</td>
<td>0.165</td>
<td>3.3</td>
<td>0.5445</td>
</tr>
<tr>
<td>ATSAM3X4E</td>
<td>0.7</td>
<td>3.3</td>
<td>2.31</td>
</tr>
</tbody>
</table>

These values demonstrate a capacity of approximately 300 mAh as needed for the required 8 hour operation time. These calculations allowed us to have a general idea of the capacity required for our coin cell battery. We were unable to find a coin cell battery that supplied 5 volts. We were able to find a coin cell battery that supplied 6 volts, but it did not have the required capacity to operate our demonstration PCB for the required amount of time. We then began to look at smaller voltage coin cell batteries and decided to use two 3 volt coin cell batteries in series, to supply a combined 6 volts to the demonstration PCB. We were able to locate a coin cell battery that has a voltage of 3 volts with a capacity of 1 Ah. This capacity of 1 Ah allows us to power our demonstration PCB for a period of approximately 24 hours; however, the cost of just one battery was too expensive for our purposes, at $3.81 CAD. Due to the cost, we decided to use a coin cell battery with a smaller capacity of 240 mAh, which will allow our demonstration PCB to run for about 7 hours, which is close to the initial time of operation we required of 8 hours. Achieving an operation time of 8 hours is not crucial, as operation time is application specific, and for our purposes 8 hours was selected as a demonstration. Also, we were able to purchase 10 batteries for $3.90 CAD, due to the smaller cost of this coin cell battery.

For a secondary power source, we decided to use a micro Universal Serial Bus (USB) port since
it is a minor component which was already intended to be placed on the demonstration PCB. The use of a secondary power source is not required in all applications. We are using it for our purposes just in case we run out of batteries and need another source to power the demonstration PCB.

The initial selection of the voltage regulator was to go with the LM317. This is a simple voltage regulator that uses external resistors to adjust the output voltage in relation to the applied input voltage. The LM317 allowed us to have an adjustable output voltage in the range of 1.25 volts to 32 volts, by adjusting external resistors [6]. This is a very wide output voltage range, so any miscalculations in selecting the external resistor values, or variances in the input voltage would cause problems on our demonstration PCB.

We also had the problem of needing to account for two different input voltages. When the demonstration PCB is powered by the micro USB the input voltage to the voltage regulator would be 5 volts, and when the demonstration PCB is powered by the coin cell battery, the input voltage to the voltage regulator would be 6 volts. This means that the external resistors would need to be different for a 5 volt input and a 6 volt input, to properly regulate the input voltage down to the required 3.3 volts. Therefore, we decided to use a voltage regulator that will consistently regulate the input voltage down to 3.3 volts when given a range of input voltages. We decided to go with the MIC5245 voltage regulator. The MIC5245 allows for a range in input voltages of 0-7 volts, and regulates it down to 3.3 volts at the output. The MIC5245 also has very low noise at the output, meaning that there will be less variance in the voltage which is going to power our major components, and as a result has a lower risk of damaging the other components. Another added benefit of the MIC5245 is that it has a very low internal voltage drop, around 150 mV with a maximum output current of 150 mA [7].

2.2.5 PCB Design

The design of all PCBs in our project was done through the use of software by CadSoft known as Easily Applicable Graphical Layout Editor (EAGLE). EAGLE has a free version which can be used
for non-commercial purposes, which we employed for the design of the Prototype and Demonstration PCBs.

2.2.5.1 Prototype PCB

Our initial PCB design was a prototype PCB. The prototype PCB houses the accelerometer and temperature sensors which allows us an easy way to map out the connections between the microcontroller and the sensors. Using an Arduino Due for development purposes, along with the BLE112 development kit, allowed us to map out and test all the connections between the sensors, microcontroller, and BLE module without having to solder any expensive components. The prototype PCB served as a good tutorial for using the EAGLE software. It also allowed us a safe way to ensure the integrity of our connections, and the functionality of the hardware before mounting everything onto the demonstration PCB.

2.2.5.2 Demonstration PCB

The demonstration PCB is an example of how a developer may use our framework to create a hardware application. The demonstration PCB contains the microcontroller, sensors, BLE module, battery, and any other auxiliary components. It is a PCB that is meant to operate self-sufficiently with the capability to send sensor data to the Phone Application. After the connections between the components had been tested and our hardware application was confirmed to work with the Phone Application, we mapped out all the connections in EAGLE and developed the demonstration PCB. The connections between all of the major components on the demonstration PCB, such as the microcontroller, sensors, and BLE module, are had been individually mapped out by us. There were, however, minor connections for components such as the micro USB port, joint test action group (JTAG), and crystal oscillator which were referenced from the Arduino Due schematic provided by the Arduino community [8].
2.3 Implementation and Integration

In this section, we will discuss the implementation and integration of the microcontroller, sensors, Bluetooth, and power management components for the hardware application.

2.3.1 Microcontroller

After we decided on the microcontroller, we began writing an application which would read the data from the sensors populated on the PCB, and send it to the BLE112. After some time developing the software, it came to our attention that the Wire library, a library which supports Two-Wire Interface (TWI) communication, has a bug in the software. To communicate with our accelerometer, we were required to use a repeated start bit for multi-byte transmissions. The bug in the code, however, did not have the repeated-start bit functionality completed for the ATSAM3X8E/4E. This meant that the I²C communication between the ATSAM3X4E and the MMA8452Q was broken. After various unsuccessful attempts to find a solution from the Arduino community, and unable to fix the issue ourselves, we decided to switch to the ATMega1280. Although the ATMega1280 is a 8-bit microcontroller with an AVR architecture, and has much less performance and memory compared to the ATSAM3X4E [9], the Wire library for this chip is completely functional. This decrease in performance may be an issue in some applications, but for our purposes it had no adverse effects. At this point and time, an issue revolving around programming the ATMega1280 came to light. We had problems with directly programming a bootloader onto the microprocessor without the use of the co-processor used on the Arduino Mega [10]. This issue is more severe than the problems with the Arduino Wire library for the ATSAM3X family, and for this reason, we had to switch back to using the ATSAM3X4E. We were able to resolve the issues with the Wire library and are using the ATSAM3X4E microcontroller in our hardware application.

To integrate all of the peripheral components with the microcontroller, we developed three separate libraries. These libraries handled the communication between the ATSAM3X4E and the LM74, the ATSAM3X4E and the MMA8452Q, and the ATSAM3X4E and the BLE112. Each
library was written in C++ and put into the appropriate directories in the Arduino folder to allow compilation of the main program.

The program loaded onto the ATSAM3X4E is called a sketch, which consists of the two functions `setup` and `loop`. `setup` is called first, where all initializations must be made. The program then jumps to `loop`, which is continuously repeated until the microcontroller is reset. This portion of the program is where the main functionality goes.

For our program, `setup` consists of the steps found in Figure 2.1.

![Diagram of program execution steps](image)

**Fig. 2.1:** Steps executed in `setup` when the program begins to run.

`setup` begins by first initializing the LEDs populated on the demonstration PCB. The LEDs that are initialized include:

- A yellow LED that lights up when UART Transmit (TX) is being handled.
- A yellow LED that lights up when UART Receive (RX) is being handled.
• A yellow LED that lights up when SPI TX is being handled.

• A yellow LED that lights up when SPI RX is being handled.

• A yellow LED that lights up when I²C TX is being handled.

• A yellow LED that lights up when I²C RX is being handled.

• A red LED for indicating errors.

• Blue LED to notify when a super is connected to the BLE112.

This task must be completed first as many sequential initialization functions rely on lighting the red error LED if there are issues initializing the devices. Next, the UART is initialized, and set to a speed of 115200 bits per second (bps). After the UART has been initialized, the program calls the BLE112 initialization routine in the BLE112 library, which is shown in Figure 2.2.
The microcontroller first sends a “System Hello” command to verify that the BLE112 is functional and responding. After the first message has been sent, the microcontroller sets up the BLE advertisement and bonding status, allowing the Phone Application to see the hardware application when searching for BLE devices. If any of the responses to the commands are sent back to the microcontroller, and they are invalid, this will notify the user that a warning has occurred by turning on the red error LED.
After the BLE112 has been initialized, the program sets up the SPI functionality, and initializes the temperature sensor. The sensor is initialized with the function `update` inside the LM74 library, shown below in Figure 2.3. If the manufacturer ID returned to the microcontroller is incorrect, the red error LED will be turned on.

![Flowchart](Fig. 2.3: Steps executed when `update` in the LM74 library is called.)

Next, the I²C connection is set up, and following that, the MMA8452Q accelerometer is initialized as shown in Figure 2.4. To verify if the device is working correctly, the microcontroller sends a “Who Am I?” message to the MMA8452Q. If the correct value is returned, the microcontroller configures the accelerometer to have the correct X/Y/Z Axis configuration specified by the developer, otherwise, the error LED is turned on. Lastly, two software timers are set up for both the LM74 and the MMA8452Q, each based on the time delay required before a new reading can be read from the sensors.
After `setup` has been completed, `loop` will be called next. `loop` as it is structured for our application, can be found in Figure 2.5. The first part of `loop` checks whether or not the sensor information needs to be updated by checking the current time against the software timers defined in `setup`.
If the timer for the temperature sensor expires, the program will call the same update function shown in Figure 2.3. If the timer for the accelerometer expires, the program will call the `getXOutput`, `getYOutput`, and `getZOutput` residing within the MMA8452Q library, as shown in Figure 2.6. The program will send simple commands over I²C, and the current register readings on the MMA8452Q are returned.
After the readings have been updated, the program will check and see if the BLE112 is currently connected to a mobile device over BLE. If the device is disconnected, the program will turn the blue LED off, to indicate that there is no connection present. If the program detects that there is a connection, the program will turn on the blue LED, and send the updated data over UART to the BLE112, which is sent to the connected device.

The main program also includes a function named `serialEvent`, which is executed at the end of `loop`. The purpose of this function is to determine whether or not data has been received from the BLE112 over UART. If data is present in the UART RX register, the `serialEvent` function will call the BLE112 command handler, shown in Figure 2.7.
For our application needs, we are not receiving data from the BLE112, and only transmitting to the BLE module, so the only commands we need to check are whether or not someone has connected to the BLE112, or disconnected from the BLE112.

### 2.3.2 Sensors

The application schematic for the accelerometer, as shown in Appendix B, was built upon by the application provided in the MMA8452Q datasheet [11].

This gave us a simple way to implement the MMA8452Q, without the worry of whether our connections were correct. The implementation for the LM74 temperature sensor, as shown in
Appendix B was built upon by the application provided in the LM74 datasheet \[12\].

This again gave us security in our connections. We did, however, come across a problem that was remedied. The LM74 communicated over SPI as we had wanted, but it did so over three wire SPI. This meant that the Master Out Slave In (MOSI) and Master In Slave Out (MISO) were on the same pin. As a result, we needed to employ a resistor between the LM74 MOSI and MISO pin, and the microcontroller, to have the MOSI and MISO function independently. The only issue with implementing the sensors was ensuring they were connected to the proper pins on the microcontroller. This was taken care of by simply creating a prototype PCB, which is discussed in Section 2.2.5.1.

### 2.3.3 Bluetooth

Since we decided on selecting the BLE112 as our external BLE module, we had two options when it came to interfacing the module with the microcontroller. The first method was to have the microcontroller communicate with the BLE112 through the Bluegiga Application Program Interface (BGAPI). The BGAPI protocol is a transport layer added on top of standard UART protocols which allows the microcontroller to control specific functionality on the BLE112 \[13\]. The other method of interfacing the BLE112 and the microcontroller together was through the use of a Bluegiga Script (BGScript) program. BGScript is an abstracted programming language created by Bluegiga which allows a user to easily program and run applications on the BLE112, and allows the program to internally call BGAPI commands without the need of an external microcontroller \[14\]. This would involve creating an event handler on the BLE112 which would parse any data received over UART.

Since the direct BGAPI method would require the microcontroller to not only transmit packets, but receive and parse packets, whereas a BGScript program would only have to receive transmitted packets, offloading processing power on the microcontroller, we initially decided that we would go with the BGScript approach. However, after many attempts to try and integrate the BLE112 and microcontroller together with a BGScript program, we ran into an unknown locking issue. We
determined that the issue was related to the UART handler in the program, but we were unable to resolve the problem [15]. As a result, we decided to then switch to the BGAPI method. Since there were no real timing requirements, and processing time on the ATSAM3X4E was not a concern, it did not affect our hardware application in any way.

2.3.4 Power Management

The two power sources on our demonstration PCB are separated by a simple ON-OFF-ON switch. This switch allows us to switch between using the micro USB as a power source and the coin cell battery as a power source. It also gives us the option of cutting off all power to the demonstration PCB by putting it to the OFF position. The output of the switch is sent to any components, mainly the LEDs, that require 5-6 volts for operation, and the remaining components are fed with the output of 3.3 volts of the voltage regulator.
3. Phone Application

Within our framework, the Phone Application serves as a relay point between the microcontroller and the server. To accomplish this, the application can be divided into three tasks: reading data over BLE, displaying the incoming data to the user, and transmitting the data to the server. The following sections will discuss our design decisions and implementation of these tasks.

3.1 Design Decisions

In this section, our initial design decisions are discussed. These decisions include both the mobile platform and development environment used for developing the Phone Application.

3.1.1 Development Platform

When work began on the Phone Application, we were informed by Dr. Ferens that we would be given freedom to choose the platform used to develop the Phone Application. We had three options available. We could develop the Phone Application for either Android, iOS, or both. Initial research into Android showed promising results for our purposes. Android is programmed entirely in Java, a programming language that we are all familiar with [16]. This familiarity meant that we would not need to learn a new programming language before we could begin development. Another benefit of Android, is that the development tools exist for both Windows and Unix computers, allowing us to perform all the development on our personal computers [16]. The deciding point for choosing Android, is that we have access to multiple devices to use for testing, removing the need to test using emulators. This gives us the ability to address errors that can be present when using a physical device opposed to a virtual one.

After beginning our research into iOS, we discovered that it would not be a viable development
platform for our purposes, for two reasons. The first problem with iOS is that it is programmed entirely in Objective-C [17]. Given that Objective-C is a C based programming language, it would not have been difficult to learn, but this learning curve would slow down development. iOS also presented the issue of requiring development software that can only be used on OSX [17]. This posed another issue, as no member of our group has access to a computer with OSX. This meant it would be necessary to either purchase a computer with OSX, or arrange for the required development software to be installed in an engineering laboratory that has a computer with OSX.

Given the above issues with iOS development, we came to the decision to perform our development exclusively on Android. This decision allowed us to begin development immediately without requiring us to purchase any development hardware.

### 3.1.2 Development Environment

Once it was decided that we would be developing our Phone Application for Android, we had to decide on our development software. Given the open source nature of Android, we were presented with two possibilities. We could use the Android Development Tools (ADT) from the command line, or the dedicated version of the Eclipse IDE that integrated the ADT [16]. When assessing the possibility of using the ADT from the command line, we found it to be unsuitable. The use of the ADT would allow each of us to develop using our text editor of choice. The use of our preferred editors would have resulted in a short learning curve since we would only need to learn the Android specific tools. Use of the command line tools would, however, have introduced the issue of consistency. By not developing with a single editor, there would be consistency issues introduced in the formatting of the code. While these inconsistencies would not be a problem in a small program, our initial research showed that our code for the Phone Application would span multiple files, making the framework harder to understand and use.

Our assessment of Eclipse found it to be ideal for our purposes. The dedicated version of Eclipse designed for Android development has all of the ADT functions integrated into it, drastically reduc-
ing the learning curve. Eclipse also allows for easy formatting of code based on a template, allowing us to easily format our code to match any agreed upon standards. Based on these advantages, Eclipse was the obvious choice for our purposes.

3.2 Implementation

Implementation of the Phone Application was divided into three sections. Within this section, implementation of the Bluetooth reader, application interface, and server communication is explained.

3.2.1 Bluetooth

The development of the Bluetooth functionality for the Phone Application began with an assessment of possible open source programs that could be used as a base. Although there are many open source programs demonstrating traditional Bluetooth communication, examples of the newer BLE protocol were nearly non-existent. A single open source example was found from the official Android developer resources, which was used as our starting point.

Once we had the sample program, we needed to understand how it functioned before we could make proper use of the program’s code. The program worked by scanning for BLE devices and providing a list of Bluetooth devices available for connection. Once the user selected one of the discovered devices, a new Android Activity would open. This new activity would connect to the selected device, then display a list of Universally Unique Identifiers (UUID) that the connected device was transmitting on. The user could select one of these characteristic UUIDs to read the data from it.

Our first step in modifying the code was to change the behavior of the program upon selecting a Bluetooth device. We needed the ability to connect to a known Bluetooth device automatically. Our approach to this problem was to find a way to save the selected Bluetooth device in persistent storage where it could be retrieved when needed. As there was no functionality for this within
Android, it had to be developed.

The issue of saving a Bluetooth device was resolved by storing the data necessary to create a Bluetooth device at launch, locally on the phone. We found that this is possible through the use of the device’s unique Media Access Control (MAC) address. It was then a matter of creating a class that could be passed a Bluetooth device. This class takes the Bluetooth device, retrieves its MAC address, and saves the address in persistent storage. On launch, this class retrieves the Bluetooth device. In this case the device address is retrieved from storage, and then a Bluetooth device is created from this address and returned. Once this class was created, implementing it in the Phone Application, was a matter of changing the behavior of selecting a Bluetooth device, so the device would be saved.

Once the selection and saving of a Bluetooth device was finished, it was necessary to adapt the reading of data to meet our needs. In this case, we needed the Phone Application to automatically read data from a list of known UUIDs at a regular interval. To accomplish this, we broke the problem into two tasks: reading from known UUIDs, and continuously polling for new data without user intervention.

We approached the problem of reading from a known UUID by establishing a list of all UUIDs that had available data. The first step was to add a check to the process of listing all of the UUIDs to see if the known UUID was in the list. If the known UUID was found, the Phone Application automatically reads from the UUID. While this method did allow us to read from a known UUID without having to manually select it, there were major performance issues. The entire list of UUIDs would have to be created each time we wanted to read a value.

To improve the performance while reading Bluetooth data, we needed to save the Generic Attribute Profile (GATT) associated with the UUID, so the GATT could be reused without the need to generate a list of UUIDs. This was accomplished by creating a globally accessible object to save the GATT. We then set the Phone Application to create the list of GATTs as soon as the first read request was issued. When a known UUID was found, the corresponding GATT would be saved
globally. We then called the `readCharacteristic` method that initiated a read using the globally saved GATT.

The next step was to get the Phone Application to read from the Bluetooth device continuously at regular intervals. Our approach to this problem was to insert a loop into `readCharacteristic`. Within the loop, a read is performed, followed by a `sleep` for the desired length of time. Unfortunately, this approach presented an issue. Since the read is performed by a service running in a separate thread, a read request can be issued before the previous read request's data is received, causing the program to crash.

The next attempt at continuously reading from the Bluetooth device was to add a `sleep` after receiving a value, and then performing a new read. This solution prevented the issue of having multiple read requests issued at once. In testing, we found that this solution caused the User Interface (UI) to lock up. The reason for the UI locking was due to the `sleep` being performed on the same thread as the UI. In most cases, since the `sleep` will be longer than the time required to issue a read request and receive the data, the UI thread will spend most of its time sleeping, unable to accept user input.

The use of Android’s `AsyncTask` class solved this problem. This class allowed us to create an object containing a single method that, when called, executes on a new thread. By creating an `AsyncTask` that performed a `sleep` followed by a read, we were able to execute a single occurrence of this task after receiving data. This solution allowed us to remove the processing from the UI thread, while still preventing the Phone Application from issuing multiple reads before receiving data.

The final function added to the Bluetooth reading code was to allow for reading from multiple UUIDs. This task was performed in multiple steps. The first step was to save the GATTs for multiple UUIDs. This was done by creating an array of GATTs the same size as the array of UUIDs. While generating the list of valid UUIDs, each UUID in the array is checked and, if present, the corresponding GATT is added to its array using the same index.
Once we had a list of GATTs, we needed a way of changing which GATT is being read. To accomplish this we introduced a global indicator which corresponds to the index of the GATT being read. After receiving a value, the index is incremented to the next GATT and the `AsyncTask` that performs the read is passed the corresponding GATT.

3.2.2 Interface

The purpose of the UI is to provide users a means of managing their connections, and viewing the sensor readings. It is through the UI, that the user can start and stop the reading of data from the microcontroller, and transmission of this data to the server, as seen in Figure 3.1.

![Fig. 3.1: Relationship of the Phone Application with the framework.](image)

The relationship of all the Activities can be seen in Figure 3.2.
After the Phone Application is installed on the phone and run for the first time, the Phone Application will redirect to the configure settings menu. This allows the user to set the Phone Application’s connection credentials as shown in Figure 3.3.
The information in the settings page is required to log into the server and transmit data. The settings page will continue to appear when the Phone Application is started, as long as all the required fields are not filled. The required fields are:

- Username
- Password
- Server Internet Protocol (IP) address or Uniform Resource Locator (URL)
- Port number

To protect the information of the user, the password in the settings page is not shown and is only known by the user. Once the user has filled in the required fields and presses the “Save Settings"
button, the fields will be saved, and the user will not have to fill in the credentials again, unless the cache of the Phone Application is cleared. The Phone Application will then stop redirecting to the configure settings menu and will show the home screen on startup. After pressing the “Save Settings” button, the Phone Application will prompt the user to turn on the Bluetooth adapter. If the Bluetooth adapter is not already turned on, the Bluetooth device selection page is displayed, as shown in Figure 3.4. If a user wishes to access and change the settings of the Phone Application, the user can do so by pressing phone’s menu button->Settings->Configure Credentials.

![Device selection screen](image)

**Fig. 3.4**: Phone Application Bluetooth device selection screen.

In the “Settings” page, which can be accessed by pressing the phone’s menu button->settings, the user is able to view the currently saved username, server, and Bluetooth device. There is also an option for users to connect to the server or transmit data only through Wireless Fidelity (Wi-Fi) as shown in Figure 3.5.
This option prevents the Phone Application from using a cellular data plan. This option is for users who have limited or no data plan from their service provider, and require the option of only...
connecting or transmitting to the server through a Wi-Fi connection.

Once all the settings have been configured, the Phone Application will go to the home page and is ready to be used as shown in Figure 3.6.

![Figure 3.6: Phone Application home screen.](image)

The Phone Application can then connect to the Bluetooth device by pressing the “Connect to Bluetooth Device” button which outputs an alert, indicating whether the Bluetooth device was successfully connected. If successful, the button will then change to “Disconnect from Bluetooth Device”. The user can then read from the sensors on the demonstration PCB by pressing the “Read from Bluetooth” button. If successful, the button will change to “Stop reading from Bluetooth” and the readings will be continuously displayed on the lower part of the home screen. The user is able to scroll through the readings to view previous readings, if any previous readings are available. The readings are also pushed onto a queue which will be used when transmitting data to the server.

Initially, incoming Bluetooth data was simulated by reading the phones internal sensors. This
led to a significant performance decrease in the Phone Application since the read method was running on the main thread. The Phone Application slowed down further when the number of sensors being read increased. In some cases performance was so severely impacted that the Phone Application would crash. To remedy this problem, the method to read from the sensors was put into a separate thread. The code used to simulate the readings was then replaced with the code that performed the actual reading of the data from a connected Bluetooth device.

Connecting and transmitting to the server requires that the user have a data plan or a Wi-Fi connection established. The user can connect to the server using the “Connect to Server” button, which will try to connect to the address and port number that was entered in the configure settings page. An alert will show the status of the connection and whether the connection was successful. The button will then change to “Disconnect from Server”. If the user did not choose to fill in the settings page and tries to connect to the server, the Phone Application will give an alert indicating that the settings page needs to be completed. Upon a successful connection, the server is ready to receive sensor data from the hardware application.

The user can transmit data to the server by pressing the “Transmit to Server” button, which will pull the data from the queue and send it to the server until the queue is empty. The button will then change to “Stop Transmitting to Server” to notify the user that the Phone Application is transmitting to the server. The Phone Application can continuously read from the Bluetooth device and transmit to the server concurrently, as long as the Phone Application is connected to both the Bluetooth device and server. If a Wi-Fi connection is unavailable, the Phone Application will queue the data until a connection is established. The user does not have to connect to the Bluetooth device and then to the server in this specific order. The connections are mutually exclusive and can be done in any order.

The reading of data from the Bluetooth device and the transmitting of data to the server are tasks which run in the background. In the event the user presses the home button on the phone and the Phone Application loses focus, operation will continue to run in the background. This
background operation allows the user to be able to do other things on the phone while also reading data from the Bluetooth device and transmitting the data to the server.

By utilizing the lifecycle of an activity \cite{18}, we are able to automatically save data and close all connections to prevent any loss of data or connection issues that may occur the next time the Phone Application is run.

Before the Phone Application exits, a method is called that saves the data that has not been transmitted to the server. This unsent data is saved in a local file on the phone. The data is then loaded back onto the queue upon the startup of the Phone Application, and the file that is locally saved is deleted to prevent the duplication of data.

Initially, the Phone Application was developed using a tabbed Activity and the use of Fragments to allow users to more quickly switch between screens. This caused several issues with accessing methods and resources outside the Fragment which led to roundabout access methods that complicated the structure of the Phone Application. We later rebuilt the Phone Application using a blank Activity and switching to other pages such as the settings or selection of the Bluetooth device through the use of intents.

During some of the testing periods of the Phone Application, a problem was encountered, causing the Phone Application to crash when the orientation of the phone changed. The buttons and fields could not be correctly placed due to screen size constraints, which caused the Phone Application to crash. To fix this, the Phone Application is fixed to run in a portrait orientation which is set in the Android manifest.

A minor problem in the Phone Application was that some methods in the code would present warnings. These warnings are due to an older API using deprecated methods. The range of API when developing the Phone Application varies depending on implementation of the methods and classes used. This was fixed by changing the minimum API to 18.

To summarize, the interface allows the user to connect to the Bluetooth device and/or server and be able to receive and/or transmit data. Furthermore, all implementation of the code for the
Phone Application is abstracted and divided into methods and classes, allowing developers to tailor the framework to suit their specific hardware application. In order to use certain functions of the Phone Application, it is required to edit the android manifest through the IDE and add permissions to use certain functions, namely the Bluetooth and internet functions.

### 3.2.3 Server

Initially, the connection to the server was developed using Sockets from the provided java library. These Sockets were tested and worked with a local server that was developed and run on the same internal network as the phone. This required both the server and the client to develop their own protocols for communication. Testing with the local server was successful and the local server was able to receive data sent from the Phone Application through Wi-Fi.

A problem encountered when testing with a local server was, that the server had to be running on the same Wi-Fi network as the client to be able to establish a connection. We could not verify if the connection would work with an actual web server since the connection had to be established locally. Once the web server was set up, we realized that the methods in the WebServeletSockets class are deprecated for the server side implementation which caused connection issues to the web server.

The WebSocket class, contained in the AutoBahn library, was found as an alternative to the Java Sockets library. On top of WebSockets, which allows bidirectional real-time messaging on the web, the Web Application Messaging Protocol (WAMP) adds asynchronous Remote Procedure Calls (RPC) [19]. To connect to the server, we call the connect method and pass the method the address and the port number of the server, separated by a colon. The communication protocol with the server was developed by using override methods which are called automatically upon a:

- Successful connection.
- Message is received.
• Connection is closed.

In this implementation, when a connection is opened the user is informed that the Phone Application has successfully connected to the server. Upon receiving a message from the server, the Phone Application processes the incoming data and displays the data on the homepage for debugging or viewing purposes. When the connection is closed, the method will close the sockets along with the input and output streams which were created to process the data that was incoming or outgoing from the Phone Application to the server. This is also used to alert the user that the connection is closed.

To send a message to the server, a message is passed to a send method, but the AutoBahn library did not support sending a stream of data. Thus, we made the decision for the communication protocol to have to send the username, password, and data on each send method call. Although this communication protocol does increase the amount of data sent, the increase was not significant enough to affect the transmission speeds of the Phone Application.

### 3.3 Integration

Once the components of the Phone Application were implemented, we began integrating them with the rest of the framework. Integration was performed in two parts: integration with the microcontroller, and integration with the server. This section discusses the process and results of the integration.

#### 3.3.1 Hardware Application

After developing the Bluetooth code we began integrating it with the microcontroller. This integration involved testing possible situations that may arise in real world use to find any bugs that went unnoticed during development. Two problems were discovered during this process that were fixed: reconnecting to a Bluetooth device and changing the GATTs being read.

The first problem occurred when disconnecting from the Bluetooth device and attempting to
reconnect without closing the Phone Application. In this case, the Phone Application not only failed to connect a second time, it crashed. After examining the code responsible for disconnecting the Bluetooth device, we discovered that while the device was disconnected, the associated Bluetooth service was still running. Since the service was still running, trying to reconnect resulted in a second service being attached to the Bluetooth device. A Bluetooth device can only handle a single service, leading to a failure when a second is assigned. Our solution was to include code to terminate the service when disconnecting.

Our second problem came when we attempted to change the UUIDs of the GATTs being read. In this case the Phone Application was unable to find the associated GATTs, resulting in no data being read. After further testing, we discovered that there is an internal cache for the phone's Bluetooth. When creating the list of UUIDs for a given device, the list will be taken from the cache if it exists, without checking to see if the values in the cache have changed. To clear this cache, it was necessary to reset the Bluetooth module in the phone. We eventually found a solution in the form of an unreleased method contained in Android, designed to refresh the cache. Knowing that this method existed, it was possible for us to use it, resetting the Bluetooth cache after disconnecting from the Bluetooth device.

The resulting only requires that the list of UUIDs and associated sensor types are supplied to it in order to function. When the user wants to begin reading Bluetooth data, a list of valid GATTs to receive data from is generated. While generating this list, each UUID is checked against the list of desired UUIDs and if a match is found, the corresponding GATT is saved. The Phone Application then issues a read request on the first GATT. This read request is sent to the Bluetooth service that is running in a separate thread. Once the service receives the data, it gets passed back to the UI thread. Once the UI thread receives the data it sets the next GATT to read from and executes an AsyncTask. This AsyncTask enters a new thread where it performs a wait for the preprogrammed period of time then issues another read request to the Bluetooth service. This cycle continues until another method is called to signal that during the next loop, the AsyncTask should not be executed.
3.3.2 Application Server

Integration of the Phone Application with the server is done through use of the WebSockets class from the AutoBahn library. The only concern was deciding on the communication protocol between the Phone Application and the server. The communication protocol is required to be set up so that the server will be able to process the data properly. We agreed that the sent messages will include, and be in the order of:

- Username

- Password (encrypted using Message-Digest 5 (MD5))

- Sensor type

- Timestamp

- Sensor reading

We went through a stress test phase during integration to test the boundaries of the data handling of the server and Phone Application. The test was carried out using multiple phones connected and transmitting to the server at the same time with transmissions speeds set to 1/100th of a second. Due to only having one Bluetooth device to read from at one time, the testing was done using dummy data created on the Phone Application itself. The server was successful in receiving and handling a stream of data entries from multiple devices which proves that a significant amount of data can be sent at one time, and that the server can handle the input from multiple connections.
4. Server

The server is a computer in a remote location running the software we have developed. The server has a MySQL database for data storage and retrieval. The server is also running both the Web Application and the Phone Listener concurrently, allowing users to connect to a system which is always active. An overview of the server and its subsystems can be seen in Figure 4.1.

![Diagram of server subsystems](image)

**Fig. 4.1:** An overview of the server with details on communication between its multiple subsystems.
4.1 Requirements

Through meetings with Dr. Ferens, we came up with a list of features the server would need to support. The primary concern when designing the server was to realize a framework. We decided that the server should be developed with convenience in mind because we expect other developers to use our framework in the future. In addition, a framework which is easy to use is more functional and will be used more often.

We determined that the Web Application would be required to use HyperText Transfer Protocol Secure (HTTPS) to ensure the security of any information transferred between the client and server. Wi-Fi was used for the transfer of sensor readings from the Phone Application to the server, because Wi-Fi is widely used and supported [20]. We also determined that the server would be required to store not only the sensor data, but also the account information of users. For ease of access and quick retrieval, the sensor data and account information needed to be stored in a clear and logical manner. For the above reasons we decided that the server would require a database.

Realizing the need to store sensor data and user account information, we determined that the server would also require an interface to display that data and information to its users in a well-structured, highly comprehensible, stable, and efficient manner. With that in mind, we decided that the server would use a web interface. Finally, since the original goal was the development of a framework, developers require a way of understanding the server and its functions. For that reason we decided that an API documentation detailing the different aspects of the server would be required.

4.2 Design Decisions

In the following three subsections, we discuss the design decisions we had to make and how we made them for the database, Web Application, and Phone Listener.
4.2.1 Database

There are four criteria that needed to be met by the database we chose. The first is that the database has to be inexpensive. Showing the framework works with affordable software will make it an option for more developers with a limited budget. The second criterion is that the database has to be designed for multiple users. Since our Web Application handles multiple connected users, the database has to be able to handle multiple users to keep up with the server. The third criterion is that the database must support an integrated management system. We found that a Database that supports an integrated management system will better facilitate server-database development, therefore, this is seen as an advantageous feature. The fourth criteria is that the Database must support Java DataBase Connectivity (JDBC). Since the server is being developed in Java, and because Java is a programming language the team is very familiar with, a database that supports Java oriented connectivity is seen as a major advantage.

Using the four criteria above we assessed desktop databases and relational databases to determine which better suited our project. The desktop databases we researched, turned out to be inexpensive, but were not designed for multiple users and did not have an integrated management system. On the other hand, the relational databases we researched, turned out to be designed for multiple users and did have an integrated management system, but are expensive. JDBC support is something that varied from database to database. Through more research we were able to find a relational database that is free and has JDBC support. This database, MySQL, is the database that was chosen for this project.

MySQL is a free, open-source, relational, server database and database management system developed by Oracle. It is a database that is used by organizations such as Google, YouTube, and Facebook and has an operational history of nineteen years [21]. Combined with online support and example code, it was obvious that MySQL was the right choice for this project.
4.2.2 Web Application

We looked at two different free to use servers, for our Web Application and Phone Listener to run on; Apache Tomcat and NginX. We ultimately decided to use the Apache Tomcat 7 for the Web Application, and an Apache Tomcat 8 for the Phone Listener. While both Tomcat and NginX servers can run JavaServer Faces (JSF), JavaServer Pages (JSP), and PHP: Hypertext Processor (PHP) [22], [23], [24], [25], [26], we chose Tomcat due to its proven reliability, open source community, and better suitability for our particular application, delivering unique content to multiple users simultaneously. The convenience of NginX servers is only realized when delivering static pages to multiple users simultaneously, using minimal resource consumption [27]. When pages become dynamic, Tomcat becomes the better choice. Tomcat servers run a new thread every time a new connection is made, guaranteeing a user’s session on our system is unique. This allows us to deliver each user’s specific information quickly and efficiently.

For coding the server, we considered three different languages: JSP, JSF, and PHP. As no member in our group has ever used PHP before, and our project did not give us the time to learn PHP, we decided that PHP was no longer an option. We were immediately drawn to JSP and JSF as both provide an extended version of HyperText Markup Language (HTML) allowing us to forgo low level website design. Both options also offer a powerful backing called Beans, allowing us to handle complicated coding tasks in a language we are very familiar with. JSF was chosen as our coding language as JSF includes a custom JSP tag library allowing developers to use JSP components within a JSF webpage [28]. We chose to use Tomcat 7 for the Web Application as it was the easiest Tomcat server to install JSF on.

4.2.3 Phone Listener

In developing the Phone Listener, we considered many approaches to connect the Phone Application to the server. The first was to use servlets with an HTTP Session to transfer data. Since the HTTP Session only saves connection information with no data transfer ability, we looked into
WebSocketServlets. In previous versions of Tomcat this was the standard for creating remote connections. As of Tomcat 7, WebSocketServlets were deprecated and replaced with WebSockets in Tomcat 8 [29]. For that reason we chose to use a Tomcat 8 server for the Phone Listener. After some more research we found the WebSocket API and the AutoBahn open sourced API. WebSockets gave us the appropriate implementation of communications between the Phone Application and the server. AutoBahn facilitated client based software connections to a WebSocket.

4.3 Implementation

In the following three subsections, we discuss the problems encountered, and solutions found during the implementation of the Database, Web Application, and Phone Listener.

4.3.1 Database

Through multiple discussions and revisions, we created four types of tables in the Database to facilitate the functions of both the Web Application and Phone Listener. The four tables are: the LoggedIn table, the Users table, the SensorThreshold, and the SensorData table. Each user added to the system gets a SensorData table of their own. Hence, at any given time there are 3 plus “m” tables in the system, where “m” is the total number of users registered in the system, minus 1, due to the Groot user having no data. These tables can be seen in Figure 4.2 which illustrates the four types of tables and the variables they are used to store.
The LoggedIn table consists of a single column to store usernames. The main functions of the LoggedIn table are to store the usernames of users currently logged into the system; and provide the list of currently logged in users it contains to the Web Application when requested. The LoggedIn table is used to solve the problems arising with the same user logging in multiple times and simultaneously, by assisting the Web Application in permitting a user to login only once at any given time. Figure 4.3 illustrates how this is done.
When users login, the LoggedIn table is checked and their login is permitted only if their username is not found in the LoggedIn table. Assuming the user has logged in successfully, their username is added to the LoggedIn table and the user shows up as logged in. From there, a username is only removed from the LoggedIn table if that user logs out, except in the special case of the server being reset. In that special case, all usernames are removed from the table.

The Users table consists of multiple columns for storing different elements of a user’s personal information. A user’s personal information consists of elements such as:

- **Username**: A string used as identification to access the system in combination with a password.
- **Password**: A string used as a secret word or phrase to access the system in combination with a username (This is stored as a hashed value for security purposes).
- First Name: A string indicating the user’s first name.

- Last Name: A string indicating the user’s last name.

- User Tier: A string indicating what elements of the Web Application the user does and does not have access to. The three tiers are Groot, Admin, and User.

- Registered By: A string indicating the username of the Admin or Groot tier user that registered the user.

- Registered On: A string indicating the date and time a user was added to the system.

The main functions of this table are: To store a user’s personal information when they are added to the system, remove that personal information when that user is removed from the system, update if a change to the personal information was made (like a password change), and provide the user personal information it contains to the Web Application when requested.

The SensorThreshold table consists of 4 columns for storing the different elements of sensor threshold information. Sensor threshold information consists of the following elements:

- Sensor Type: An integer used as identification to determine which sensor the threshold is for.

- Sensor Name: A string indicating the name of the sensor the threshold is for.

- Sensor Threshold: A value indicating the threshold.

- Sensor Data Type: A string used to determine the data type the sensor the threshold is for returns.

The main functions of this table are: to store sensor threshold information when a threshold is added to the system, remove that information when that threshold is removed from the system, and provide the sensor threshold information it contains to the Web Application when requested.

When creating Sensor Thresholds, we decided that each threshold would apply to all users. We did not want to make the SensorData tables any larger, as such we made the SensorThresholds table in
order to save sensor type and safe threshold values.

The SensorData table consists of 4 columns for storing different elements of sensor readings. Sensor readings consist of elements such as:

- **Timestamp**: A string used to indicate the date and time of the data reading.
- **Sensor Type**: An integer used as identification to determine which sensor the reading is from.
- **Data**: A value indicating the data reading itself.
- **Num Readings**: An integer indicating the place the reading was received in.

The main functions of this table are to store sensor readings in chronological order, and provide the sensor readings it contains to the Web Application when requested. Data tables cannot change the data types they are expecting to store after they have been created. This means storage of readings and retrieval of readings both had to be designed in a way to overcome this static nature of the tables. In the case of storing the readings, we decided to convert all readings to strings for storage in the SensorData table. In the case of retrieving the readings, Figure 4.4 illustrates the procedure.
Fig. 4.4: A Flowchart illustrating the logic of the Web Application when retrieving sensor readings from the SensorData table.

We use the sensor type a reading was stored with, to find the matching sensor type in the SensorThreshold table. Once a match is made we use the corresponding data type, of the matching sensor threshold in the SensorThreshold table, to de-convert the reading from a string to the data type it was before.

4.3.2 Web Application

Work on the Web Application began with the design and style of a website template. A website template facilitates development and functionality of the Web Application, as well as assists in establishing a more uniform appearance across the different pages of the Web Application. We
agreed on a template design consisting of three sections: header, footer, and body content. The header and footer are placed on a page to show sitewide information such as the title of the website and the copyright information. The header specifically can also be used to include any JavaScript or Cascading Style Sheet (CSS) stylings which are to be used throughout the entire website. The body content is the substance of each web page.

For our framework we decided to divide the grouping of web pages based on access, as illustrated in Figure 4.5.

![Figure 4.5](image)

**Fig. 4.5:** An overview of the Web Application detailing the contents of: the Web Server and its Packages, the Web Site and its pages, and communications to and from internal and external components.

Pages we deemed as “public” are pages that can be seen by anyone. In the Web Application,
there is only one page that is “public” and that is the Login page. Pages we deemed as “secured” are pages that require the user requesting the page to be logged in. This means that the user must be registered with the system, in order to have access to these “secured” pages. There are six pages that are “secured”, and they are:

- Welcome(Groot): A welcome page for Groot tier users.
- Welcome(Admin): A welcome page for Admin tier users.
- Welcome(User): A welcome page for User tier users.
- Database Credentials: A page for acquiring database login credentials.
- Change Password: A page for changing passwords.
- Sensor Threshold: A page for adding and removing sensor reading thresholds.

The Welcome (Groot) page allows a Groot tier user to: view the data of all registered users, remove any registered user, or add new Admin tier or User tier users to the Database. The Welcome (Admin) page allows an Admin tier user to: add new User tier users to the Database, view the data of the users that they registered, and removed the users they have registered. The Welcome (User) page allows a User tier user to log in and view their specific sensor readings. Only Groot and Admin tier users can access the Sensor Threshold page. All tiers of users are given access to the Change Password page. The Database Credentials page is a page that can only be accessed by Groot tier users under the condition that the Web Application was not able to make a connection with the Database because of incorrect or missing Database login credentials.

We decided on three tiers of user permissions, in order for the Web Application to quickly determine which features a user should or should not be able to interact with: User, Administrator (or Admin), and Groot. The tier of a user is determined when they are added to the system. The different features made available at each tier can be seen in Table 4.I.
The User tier is comprised of users with the ability to only view their own sensor readings and user information. The Admin tier is comprised of users with the abilities to:

- View their own sensor readings.
- View the sensor readings of other users they have added to the system.
- View their own user information.
- View the user information of other users they have added to the system.
- Add or remove other users at the tier of User they have added to the system.
- Add or remove Sensor Thresholds.

The Groot tier is comprised of users with all the abilities, in exception to viewing their own sensor information.
readings. Groot tier users cannot view their own sensor readings because they do not have sensor readings to view. This tier is intended for the system administrator or developer, for system maintenance.

JDBC is the base for our entire server-database communication scheme. The JDBC API is an industry standard for database independent communication between Java applications and a wide variety of databases such as MySQL [30].

Seeing as all our data is stored in, and retrieved from, the database, we have decided to create a data access hierarchy consisting of a top level parent and various children. The parent class is abstract so as to force all children to implement at least the bare essentials for proper database manipulation. These bare essentials include:

- Creating and deleting data tables.
- Creating and deleting table specific data entries.
- Retrieving a subset of data stored in a datatable for display on web page.
- Updating a specific value in a specific column.
- Retrieving the total number of rows in a data table.

This parent abstract class that enforces all the previously mentioned functions is called the Data Access Object (DAO). The DAO also has the responsibility: to open and close database connections, direct database manipulation, and handle any Groot tier user specific changes.

As we have many data tables, we saw fit to create a new DAO for each data table. This is to ensure no DAO can improperly or accidentally manipulate the wrong table. Furthermore, each data table consists of different data and each data table is structured differently. All of the DAOs that make up the Data Access package are illustrated in Figure 4.5.

Each DAO implements the same primary functions but in different ways. The UserDAO is used to edit only a single users data table. The UserDAO is used by all of our Beans. The Beans take
inputs from the web pages and submit them to DAOs which retrieve or store data. All of the Beans that make up the Beans package are illustrated in Figure 4.5.

The UserBean is a session scoped Bean meaning it retains all of its values from the moment a user logs in, to the moment they log out. The UserBean is responsible for user login and logout. How the UserBean handles user login and logout can be seen in Figure 4.6.

![Flowchart illustrating the logic of the UserBean for user login and logout.](image)

**Fig. 4.6:** A Flowchart illustrating the logic of the UserBean for user login and logout.

Once a user attempts to login, the UserBean encrypts the entered password with an MD5 hash. The UserBean then uses the UserDAO to create an SQL query which is passed to the parent DAO to be executed. The query will attempt to retrieve any information corresponding to the entered
username. If the user is found, the UserBean proceeds to use the UserDAO to fetch the user’s encrypted password for comparison against the password entered by the user. Based on the result of the password comparison the UserBean will either direct the user to: the appropriate version of the welcome page if the passwords match; Or the login page with an error message if the user is not found or the passwords do not match. The UserBean will also monitor for inactive users. If a user is inactive for a predetermined amount of time, as set by the system administrator, they are automatically logged out. This is done through the UserBean which has a method which can catch a timeout event. When this timeout event is detected, the UserBean uses the LoggedDAO to remove the user from the LoggedIn table.

The SensorDAO handles all interactions between the SensorBean and the data base. Each user, when added to the Database is given a SensorData table which stores their unique sensor readings coming from the Phone Application. When a user has successfully logged in, the Web Application shows them their sensor readings, indicating which readings have exceeded the predefined safe threshold with red highlighting. How the SensorBean does this can be seen in Figure 4.7.
The SensorBean uses the SensorDAO to create a query which is executed by the parent DAO. This query grabs “n” number of sensor readings, where “n” is a predefined number, from the users SensorData table. In order to maintain a comprehensible and functional user interface, we decided...
that only “n” readings should be loaded on the web page at a time, rather than all the readings at once. As the “n” sensor readings are gathered from the database, they are checked against thresholds stored in the SensorThreshold table. Readings that are past the threshold are marked, so that they appear highlighted when they reach the web page. In order to match up where the sensor the reading came from with the appropriate threshold for that sensor, the SensorBean uses the SensorThresholdDAO as well as a sensor identification number.

The GrootBean and AdminBean do a lot of the same tasks, as seen in Figure 4.8.
Due to the nature of JSF, it is very difficult to produce a proper hierarchy which minimizes the duplication of code. The GrootBean and AdminBean are both responsible for adding and removing users from the system at the request of Groot tier and Admin tier users. Those Beans are also responsible for the viewing of sensor readings of other users at the request of Groot tier and Admin tier users. The GrootBean and AdminBean are special as they are the only Beans which allow for...
the creation of new users.

The LoggedDAO is a very small class which is responsible for keeping track of who is currently logged into the system. How the LoggedDAO does this can be seen in Figure 4.3. When a user logs into the Web Application their username is saved in the LoggedIn table, and when that user logs out their username is removed from the table. The LoggedDAO is solely responsible for preventing multiple logins by the same user, simultaneously. This means that two or more people cannot log in with the same username at the same time. We did this to prevent the situation of multiple logins, simultaneously, by the same user. This is especially important for the Groot tier or Admin tier users as this situation leads to circumstances where a user may be edited while their data is being viewed. Furthermore, we have a special start-up method which is run when the server starts up. This method uses the LoggedDAO to remove all usernames from the LoggedIn table. If the server were to shut down unexpectedly, any user whose username is in the LoggedIn table on server start would not be able to log in again.

Since this project is also a framework, we had to consider how the different elements of the server would work together when handed out to other developers. Initially, we had developed the Web Application while making assumptions about the state of many of the elements. This includes elements like successful connection to the database, knowledge of the login credentials of the database, the schema already existing in the database, and the default profile already existing in the database.

After multiple discussions and revisions we were able to determine that those were all elements the Web Application would have to be able to detect and solve on its own in order for startup to be handled smoothly. The way we solved this startup problem was through the use of a simple text file that stores the login credentials of the database, as can be seen in Figure 4.9.
When the Web Application is started, it first looks for the credentials text file. If the Web Application does find the credentials text file, it tests the values held in the file to determine if they are the correct database login credentials. If the credentials are correct, the Web Application
checks the database for the schema. If the Web Application finds the schema, it goes on working as normal. If the schema is not found, the credentials are wrong, or the credentials file was not found, the Web Application enters its startup phase.

In the startup phase, the Web Application first waits for a user to login using the default Groot credentials. After that the Web Application asks the user for the login credentials of the database by directing the user to the Database Credentials page. If the correct login credentials (username and password) are not provided, then the Web Application displays an error and asks the user to retry with the correct credentials. When the correct login credentials are provided, the Web Application saves the credentials permanently in the credentials text file, so it can use them from then on without repeatedly asking the user. The Web Application also creates the schema, creates the LoggedIn, Users, and SensorThreshold tables, and creates a default Groot tier user account so that the user will have an account to access the system. From this point, the start-up phase ends, and the server as a whole is now ready.

4.3.3 Phone Listener

We decided to make the Phone Listener its own Java project. We initially attempted to integrate the Phone Listener into the same Java project and on the same server as the Web Application. This would have allowed us to have one server that listens on two ports. Port 80 for all normal internet traffic, and port 6789 for all phone communications. In order to have both the Phone Listener and the Web Application running on the same server, we attempted to use an engine listener which was supposed to throw an event any time a connection was made on port 6789. However, the event we were hoping to catch was never thrown. The engine listener would only ever throw events for "started", "stopped", and "keep alive". None of which allowed us to detect the connection from a phone to the server.

Our second attempt was to use an HttpSessionListener which uses HTTP Servlets to detect a connection. Having made a class to implement an HttpSessionListener we found that the session
listener would only detect a session which was created by the Web Application, which is a session that is formed by the javax faces classes. The session which we were hoping to catch, while being an HTTP Session, is different than the HTTP Session which is created by the Web Application.

The third attempt was to make a WebSocketServlet which was previously the standard for creating a remote connection. This would have involved a special webpage which would be activated by the phone. This webpage access would instantiate a Web Servlet which would then open a websocket which would accept a connection from the Phone Application. It was later found out that the WebSocketServlet API had become deprecated in Tomcat 7 and newer Tomcat Servers. The Websocket API became the replacement for WebSocketServlets and is implemented in Tomcat 8 servers.

Through more research, we found out that the most common way of running two projects, in our case the Web Application and Phone Listener, on a Tomcat server was to just run two instances of a Tomcat server and run a single project on each instance. With that knowledge we decided to move the Phone Listener into its own Tomcat 8 server, which in turn gave us the Websocket API included with the server. This can be seen in Figure 4.10.

![Diagram](image)

**Fig. 4.10:** An Overview of the Phone Listener detailing the contents of the Phone Listener Server and its Package, and communications to and from external components.

Running the Websocket API in conjunction with the AutoBahn API for creating client side software on mobile devices, we were now able to create a connection from the Phone Application to
the server. Running the server was easy but getting a proper connection still proved difficult. After searching, we found that for the Phone Listener to be able to connect, we had to add the Websocket API and the Web Application project into the build path of not only the Phone Listener project, but into the build path of the Phone Listener Tomcat 8 server as well. To test the Phone Listener, we created a basic webpage which used javascript to create a WebSocket connection to the Phone Listener. Using imitation data, we were able to save received information in the database.

With the Phone Listener, we were able to connect three phones and transfer one hundred data points each second for each user. This caused us to have an error in which too many consecutive connections were made to the database causing some data to be lost. This forced us to look at how and when we create our DAOs, as well as close our connections safely.

### 4.3.4 Remote Server

We decided that in order to have a proper demonstration of our working framework, we would need to be able to host and access both the Web Application and the Phone Listener, from a remote location. As such we set up a server which runs the Web Application and the Phone Listener. To make changes to the server we used TeamViewer, a program for remote control between computers, to remotely access the server and change running code. This server is pointed to from a Uniform Resource Locator (URL) which was registered by our group. While setting up the remote server we first tried to use a machine running Linux. While on Linux, Eclipse could not locate all the necessary libraries for JSF and WebSockets, as such we decided to install the server on a Windows machine. Due to the design of the server, we discovered that the installation has to be put in a specific location. Once the server was installed, a new instance of MySQL had to be created as our server was not able to create the necessary data tables. The Phone Listener proved most difficult to get running as it would attempt to run in a non-existent runtime environment. This was remedied by completely reinstalling the Phone Listener and creating a new runtime environment for it.
4.4 Integration

Integrating the Phone Application with the Phone Listener began with agreement on a communication standard. This standard determines how a sensor reading will be formatted by the Phone Application before it is transmitted to the Phone Listener. We decided that the data would be sent as a Comma Separated String consisting of five tokens. Each token is a value which is to be stored in a user’s SensorData table, excluding Username and Password which are used to locate the appropriate SensorData table. These tokens can be seen in Figure 4.11.

![](Fig. 4.11: An illustration of the format of a Sensor Reading when transmitted by the Phone Application to the Phone Listener.

The five tokens are:

- **Username**: A string indicating the username of the account the reading is associated with.

- **Password**: A string indicating the password of the account the reading is associated with. This is encrypted using an MD5 hash for security purposes.

- **Sensor Type**: An integer used as identification to determine which sensor the reading is from.

- **Timestamp**: A string indicating the date and time the measurement was taken to develop this reading.

- **Sensor Reading**: A value indicating the measurement of the reading.

![](Fig. 4.12: Phone Application/Server Communication Standard.
The Phone Listener implements four main functions: **OnOpen**, **OnMessage**, **OnClose**, and **OnError**. These are the four functions the WebSockets API requires developers to use when creating communication schemes. These four functions are places in an abstract parent class which all Phone Listener classes are a child of. This forces all child classes to implement at least these essential functions.

**OnOpen** will catch any incoming connections from the Phone Listener and send an acknowledgement that a connection was made.

**OnMessage** will catch any incoming messages from the Phone Application. When a message is received the message is checked for data, if the message contains no data we have received an empty message, the connection is closed and an error is sent to the Phone Application. If the message contains data, the message is broken down into its five tokens. A UserDAO is created and used to fetch the information corresponding to the received username. If the username does not exist, the connection is closed and an error is sent to the Phone Application. If a user is found by the username, the entered password is compared against the saved password. If the passwords match, a SensorDAO is created and the remaining data is parsed and used to create a new Sensor Data object. This new Sensor Data object is then stored in the SensorData table of the database by the SensorDAO. If the passwords do not match, the connection is closed and an error is sent to the Phone Application. If data is saved, a success message is sent to the Phone Application and the connection is closed.

**OnClose** will catch any connection closed events triggered by the Phone Listener or the Phone Application. **OnClose** is used to perform any actions on a closed connection.

**OnError** will catch any events thrown by an error in transmission. This method simply closes all connections if an error is encountered.
5. Conclusion

The purpose of this project was to design and implement a framework for a BSN. To complete this task we separated the project into three sections. The hardware application reads data from sensors placed on an individual’s body, and transfers it to the Phone Application through BLE. The Phone Application displays the sensor data and transfers it to the server. Finally the server permanently stores the sensor readings and user information for display when requested.

The hardware application is comprised of a PCB that has been populated with three core components: a microcontroller, sensors, and a BLE device. The components are interchangeable, as long as they have the capability to communicate with the other selected components through the defined protocols. The microcontroller relays data from the sensors to the BLE device, which transfers the data to the Phone Application. The sensors communicate with the microcontroller through the I\textsuperscript{2}C and SPI protocols, while the BLE device communicates with the microcontroller through the UART protocol.

The Phone Application implements three tasks: reading data from the microcontroller, displaying incoming data, and transmitting data to the server. Reading data from the microcontroller is done through the BLE protocol. This data is then displayed on screen for the user and transmitted to the server using the WebSocket class. In the event the Phone Application closes before all of the data is sent to the server, it saves the data to a file. Any unsent data is loaded when the Phone Application restarts and is sent to the server as soon as a connection is available.

The server is composed of three elements: A Phone Listener, a Web Application, and a database. The server is installed and runs on a remote computer. The Phone Listener accepts connections from the Phone Application and pushes the data received to the database for permanent storage. The Web Application allows users to log in and view their data. Furthermore, the Web Application
facilitates the following services based on the condition of a user’s tier:

- Adding and removing users from system.
- Adding and removing user defined thresholds.
- Viewing data of other users.

To conclude, our project provides an example hardware application, and abstracts the Phone Application and server. This gives developers a starting point when creating BSN applications. The developers will have access to various sections of our code, giving them the ability to make any desired changes to suit their needs, while retaining the full functionality of our framework.
References


REFERENCES


## A. Budget

Below are the costs associated with our project. All prices are in CAD.

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Part Number</th>
<th>Supplier</th>
<th>Cost ($)</th>
<th>Quantity</th>
<th>Total ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>ATSAM3X4E-AU-ND</td>
<td>Digikey</td>
<td>13.02</td>
<td>1</td>
<td>13.02</td>
</tr>
<tr>
<td>ICE Debugger</td>
<td>AT91SAM-ICE-ND</td>
<td>Digikey</td>
<td>133.17</td>
<td>1</td>
<td>133.17</td>
</tr>
<tr>
<td>Bluetooth Module</td>
<td>1446-1017-1-ND</td>
<td>Digikey</td>
<td>22.70</td>
<td>1</td>
<td>22.70</td>
</tr>
<tr>
<td>Bluetooth Dev Kit</td>
<td>1446-1040-ND</td>
<td>Digikey</td>
<td>248.69</td>
<td>1</td>
<td>248.69</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>LM74C1MX-5/NOPBCT-ND</td>
<td>Digikey</td>
<td>2.12</td>
<td>5</td>
<td>10.60</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>MMA8452QR1CT-ND</td>
<td>Digikey</td>
<td>1.72</td>
<td>5</td>
<td>8.60</td>
</tr>
<tr>
<td>1x12 Pinhead</td>
<td>N/A</td>
<td>ECE</td>
<td>N/A</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>1x4 Pinhead</td>
<td>N/A</td>
<td>ECE</td>
<td>N/A</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>1x3 Pinhead</td>
<td>N/A</td>
<td>ECE</td>
<td>N/A</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>Services</td>
<td>PCB Manufacturing/Soldering</td>
<td>ECE</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Capacitor 0.1 uF</td>
<td>N/A</td>
<td>ECE</td>
<td>N/A</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>Capacitor 0.1 uF</td>
<td>1276-1007-1-ND</td>
<td>Digikey</td>
<td>0.13</td>
<td>11</td>
<td>1.43</td>
</tr>
<tr>
<td>Capacitor 4.7 uF</td>
<td>1276-1065-1-ND</td>
<td>Digikey</td>
<td>0.26</td>
<td>2</td>
<td>0.52</td>
</tr>
<tr>
<td>Capacitor 22 pF</td>
<td>339-1113-1-ND</td>
<td>Digikey</td>
<td>0.13</td>
<td>2</td>
<td>0.26</td>
</tr>
<tr>
<td>Capacitor 1uF</td>
<td>339-1284-1-ND</td>
<td>Digikey</td>
<td>0.16</td>
<td>1</td>
<td>0.16</td>
</tr>
<tr>
<td>Capacitor 0.011 uF</td>
<td>490-8297-1-ND</td>
<td>Digikey</td>
<td>0.54</td>
<td>3</td>
<td>1.62</td>
</tr>
<tr>
<td>Resistor 0 ohm</td>
<td>CR0805-J/-000ELFCT-ND</td>
<td>Digikey</td>
<td>0.14</td>
<td>1</td>
<td>0.14</td>
</tr>
<tr>
<td>Resistor 1k ohm</td>
<td>CR0805-FX-1001ELFCT-ND</td>
<td>Digikey</td>
<td>0.14</td>
<td>9</td>
<td>1.26</td>
</tr>
<tr>
<td>Resistor 10k ohm</td>
<td>P10KACT-ND</td>
<td>Digikey</td>
<td>0.14</td>
<td>1</td>
<td>0.14</td>
</tr>
<tr>
<td>Resistor 39 ohm</td>
<td>RHM39AHCT-ND</td>
<td>Digikey</td>
<td>0.23</td>
<td>2</td>
<td>0.46</td>
</tr>
<tr>
<td>Resistor 4.7k ohm</td>
<td>1276-5544-1-ND</td>
<td>Digikey</td>
<td>0.028</td>
<td>10</td>
<td>0.28</td>
</tr>
<tr>
<td>USB Micro B</td>
<td>609-4613-1-ND</td>
<td>Digikey</td>
<td>0.60</td>
<td>1</td>
<td>0.60</td>
</tr>
<tr>
<td>Voltage Regulator</td>
<td>576-2790-1-ND</td>
<td>Digikey</td>
<td>1.64</td>
<td>1</td>
<td>1.64</td>
</tr>
<tr>
<td>20-10 JTAG Adaptor</td>
<td>1188-1016-ND</td>
<td>Digikey</td>
<td>9.17</td>
<td>1</td>
<td>9.17</td>
</tr>
<tr>
<td>Red LED</td>
<td>160-1427-1-ND</td>
<td>Digikey</td>
<td>0.41</td>
<td>1</td>
<td>0.41</td>
</tr>
<tr>
<td>Blue LED</td>
<td>160-1645-1-ND</td>
<td>Digikey</td>
<td>0.57</td>
<td>1</td>
<td>0.57</td>
</tr>
<tr>
<td>Green LED</td>
<td>160-1423-1-ND</td>
<td>Digikey</td>
<td>0.39</td>
<td>1</td>
<td>0.39</td>
</tr>
<tr>
<td>Yellow LED</td>
<td>160-1175-1-ND</td>
<td>Digikey</td>
<td>0.47</td>
<td>6</td>
<td>2.82</td>
</tr>
<tr>
<td>Varistor</td>
<td>CG0603MLC-05ECT-ND</td>
<td>Digikey</td>
<td>0.26</td>
<td>3</td>
<td>0.78</td>
</tr>
<tr>
<td>Crystal Oscillator</td>
<td>XC1282CT-ND</td>
<td>Digikey</td>
<td>0.77</td>
<td>1</td>
<td>0.77</td>
</tr>
<tr>
<td>Resistor Banks</td>
<td>Y9104CT-ND</td>
<td>Digikey</td>
<td>0.17</td>
<td>2</td>
<td>0.34</td>
</tr>
<tr>
<td>Push Button Switch</td>
<td>CKN9112CT-ND</td>
<td>Digikey</td>
<td>0.29</td>
<td>1</td>
<td>0.29</td>
</tr>
<tr>
<td>Switch</td>
<td>Y9104CT-ND</td>
<td>Digikey</td>
<td>8.20</td>
<td>1</td>
<td>8.20</td>
</tr>
<tr>
<td>Battery Holder</td>
<td>1061KCT-ND</td>
<td>Digikey</td>
<td>3.82</td>
<td>1</td>
<td>3.82</td>
</tr>
<tr>
<td>Item</td>
<td>Code</td>
<td>Supplier</td>
<td>Quantity</td>
<td>Unit Cost</td>
<td>Total Cost</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>Battery Coin Cell</td>
<td>N189-ND</td>
<td>Digikey</td>
<td>10</td>
<td>0.39</td>
<td>3.09</td>
</tr>
<tr>
<td>PCB Copper</td>
<td>473-1007-ND</td>
<td>Digikey</td>
<td>1</td>
<td>30.33</td>
<td>30.33</td>
</tr>
<tr>
<td><strong>Subtotal ($)</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>507.08</strong></td>
<td><strong>507.08</strong></td>
</tr>
<tr>
<td><strong>GST ($)</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>25.35</strong></td>
<td><strong>25.35</strong></td>
</tr>
<tr>
<td><strong>PST ($)</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>40.57</strong></td>
<td><strong>40.57</strong></td>
</tr>
<tr>
<td><strong>Total ($)</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>573.00</strong></td>
<td><strong>573.00</strong></td>
</tr>
</tbody>
</table>
B. Prototype PCB Layout

Below is the PCB layout for our prototype board.
C. Prototype PCB Schematic

Below is the schematic for our prototype board.
D. Demonstration PCB Layout

Below is the PCB layout for our demonstration board.
E. Demonstration PCB Schematic

Below is the schematic for our demonstration board.