*Florida International University*

*School of Computing and Information Sciences*

Computer Science

Senior Project

Final Deliverable

Biosensing 2.0

**Team Members:** Galo Romero, Jordan Laing

**Product Owner(s)**: Shekhar Bhansali, Yogeswaran Umasankar

**Mentor(s)**: Vishal Chopade, Apurva Sonawane

**Instructor**: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) *2016 Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

***Abstract***

*In this document, we present the Android app “Biosensing” which was designed to aid in the monitoring of health for those with certain medical concerns, such as the elderly. The app has the ability to connect to Bluetooth Low-Energy (BLE) devices and collect and display data from its biosensors. It can gather data on health metrics, like heart rate and temperature, and store them in a database. It can also access the data stored in this database and display them to the user as graphs. Through the course of this document, we more thoroughly introduce the system as well as describe the user stories implemented and those that were not, the planning that went behind this project, its system design, and our system validation techniques.*

**Table of Contents**

[**Introduction**](#_30j0zll) **6**

[Current System](#_cjgjzlim31ap) 6

[Purpose of New System](#_ya4g7dblq7sb) 7

[**User Stories**](#_2et92p0) **8**

[Implemented User Stories](#_tyjcwt) 8

[Pending User Stories](#_3dy6vkm) 8

[**Project Plan**](#_1t3h5sf) **9**

[Hardware and Software Resources](#_4d34og8) 9

[Sprints Plan](#_2s8eyo1) 10

[Sprint 1](#_17dp8vu) 10

[Sprint 2](#_eoz42kc9h4g) 10

[Sprint 3](#_o03ckk9o6rj9) 11

[Sprint 4](#_3xydx5y0fv2j) 11

[Sprint 5](#_xpaznh237z6q) 12

[Sprint 6](#_s1deitrm3z5q) 13

[**System Design**](#_3rdcrjn) **15**

[Architectural Patterns](#_xies5dprfrp2) 15

[System and Subsystem Decomposition](#_lnxbz9) 16

[Deployment Diagram](#_35nkun2) 17

[Design Patterns](#_1ksv4uv) 18

[**System Validation**](#_44sinio) **19**

[**Glossary**](#_2jxsxqh) **25**

[**Appendix**](#_z337ya) **26**

[Appendix A - UML Diagrams](#_3j2qqm3) 26

[Appendix B - User Interface Design](#_xpxsvu6xjtoi) 44

[Appendix C - Sprint Review Reports](#_wyi39pikxmq) 47

[Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents](#_6w97znsp1i6t) 50

[**References**](#_48m2ux3f02a) **53**

# 

# 

# Introduction

This system was produced with the thought of helping people with heart issues and other health concerns, from mild to severe; people that visit the doctor very frequently and/or require constant monitoring. Though it has the potential ability to connect to any device that uses BLE technology, it was primarily designed for use with a very small device that can be worn like a wristwatch and is currently in development. During its design, it was also important that the product be easy to use.

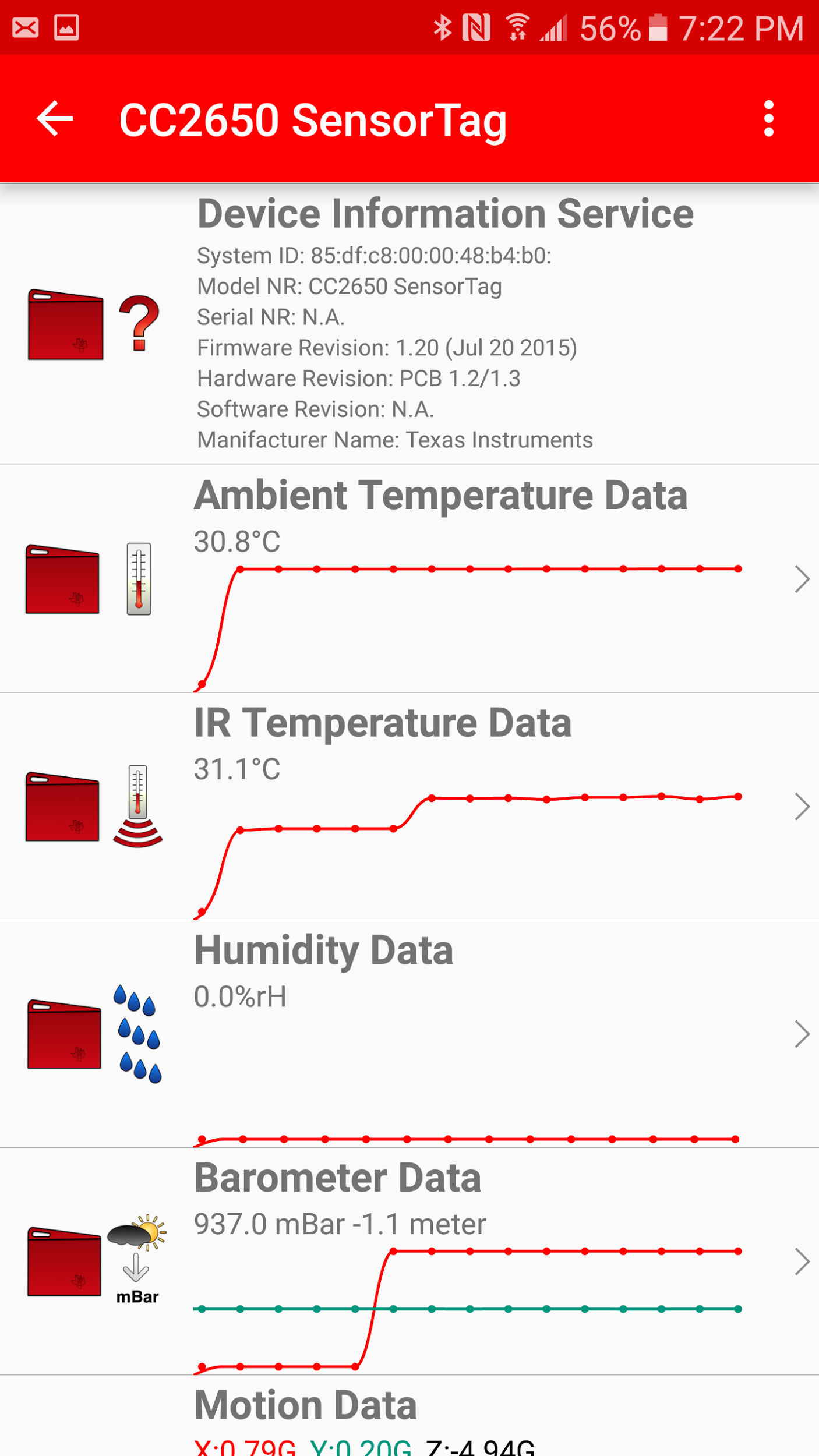
The system we designed allows the user to use an Android application to connect to a BLE biosensing device’s sensors and read their data. Additionally, while the app is connected to a sensor, it will automatically store the current reading of that sensor in a database every 30 secs. The user can also view real-time graphs of biosensor data and graphs with data in user-specified ranges in the app.

## 

## Current System

A similar system already exists that was developed by Texas Instruments. They have an Android app called T.I. SimpleLink that is designed to be used with the T.I. SimpleLink SensorTag, a small device that collects weather measurements like temperature and humidity. Although we did sometimes use the SensorTag to test our own Android app, it did not gather the types of data we needed and that would be useful to individuals and medical professionals. It also is not very user-friendly nor does it store its data in a database, and the graphs it displays are not very helpful.





## Purpose of New System

The system we developed improves upon the current one by storing the data it collects in a database so it can be accessed later. The graphs also more helpful as you can use your fingers on the screen to scroll or zoom. The user can also input their own range of dates and times to plot the data within that range on a graph and display it. It would be more useful to both individuals and medical professionals that need to monitor health data as well as view past data.

# User Stories

The following section provides the detailed user stories that were implemented in this iteration of the Biosensing 2.0 project. These user stories served as the basis for the implementation of the project’s features. This section also shows the user stories that are to be considered for future development.

## Implemented User Stories

1. User Story 185 - Connect Android app to BLE device.
2. User Story 195 - Write simple android app that connects to BLE device.
3. User Story 199 - Extract data from Android app.
4. User Story 203 - Display line graph on app.
5. User Story 207 - Integrate MS SQL Server with R services.
6. User Story 208 - Format gathered data from device.
7. User Story 216 - Pull data from server and display in graph.
8. User Story 220 - Add current equation to the app.
9. User Story 225 - Add capability to gather and display data from Sunny device.
10. User Story 229 - Merge modules into finished app.

## Pending User Stories

Due to time constraints the following user stories were moved to the product backlog:

1. User Story 233 - Make sensors send data to server on BLE connection.
2. User Story 234 - Fix disconnection and crashing errors.

# Project Plan

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

## Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

**Hardware Resources:**

* Android mobile device versions 5.0 and 6.0
* ACPI x64-based PC

**Software Resources:**

* Windows 10
* Android Studio
* MS SQL Server 2016
* Java SDK 8

## 

## 

## Sprints Plan

### Sprint 1

The first Sprint was used to meet the product owner and go over the system requirements. This Sprint was also used to install the required tools and prepare the system for coding.

### Sprint 2

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar, Vishal Chopade

After discussion, the velocity of the team were estimated to be 24.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 185 - Connect Android app with BLE device.

**Acceptance Criteria:**

1. Android app scans for BLE devices.
2. User can select and connect to appropriate BLE device.

* User Story 195 - Write simple android app that connects to BLE device.

**Acceptance Criteria:**

1. Should connect to correct BLE device
2. Should display success (or failure)
3. (Optional) The app successfully displays collected data (temperature, humidity, etc.)

The team members indicated their willingness to work on the following user stories.

* Galo Romero
* User Story 185 - Connect Android app with BLE device.
* Jordan Laing
* User Story 195 - Write simple android app that connects to BLE device.

### Sprint 3

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar, Vishal Chopade

After discussion, the velocity of the team were estimated to be 24.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 199 - Extract data from Android app.

**Acceptance Criteria:**

1. Android app is able to collect data from BLE device.
2. Data is collected as raw data.

* User Story 203 - Display line graph on app.

**Acceptance Criteria:**

1. The app generates test temperature data.
2. The app displays a line graph with the test data plotted on it.

The team members indicated their willingness to work on the following user stories.

* Galo Romero
* User Story 199 - Extract data from Android app.
* Jordan Laing
  + User Story 186 - Display data on Android app.

### Sprint 4

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar, Renny Fernandez

After discussion, the velocity of the team were estimated to be 32.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 207 - Integrate MS SQL Server with R services.

**Acceptance Criteria:**

1. The app connects to a database running on a locally hosted SQL Server.
2. The app successfully sends data to the database.

* User Story 208 - Format gathered data from device.

**Acceptance Criteria:**

1. The Android app shows data formatted correctly.
2. Make services and characteristics UUIDs easy to swap in code.
3. Data should be collected and displayed dynamically.

The team members indicated their willingness to work on the following user stories.

* Galo Romero
* User Story 208 - Format gathered data from device.
* Jordan Laing
  + User Story 207 - Integrate MS SQL Server with R services.

### Sprint 5

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar

After discussion, the velocity of the team were estimated to be 16.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 216 - Pull data from server and display in graph.

**Acceptance Criteria:**

1. The app connects to the database on the SQL Server.
2. The app pulls data from the database.
3. The app displays the data in a graph.

* User Story 220 - Add current equation to the app.

**Acceptance Criteria:**

1. Vref\_div and Rtia default values should be 0.1645 and 350000, respectively.
2. You should be able to change the values for Vref\_div and Rtia manually.
3. Display message that shows the updated value for Vref\_div and Rtia.

* User Story 229 - Merge modules for a complete app.

**Acceptance Criteria:**

1. App can connect to BLE device and collect data.
2. App can store data in SQL Server.
3. App can display data in graphs.

The team members indicated their willingness to work on the following user stories.

* Galo Romero
* User Story 220 - Add current equation to the app.
* Jordan Laing
  + User Story 216 - Pull data from server and display in graph.
  + User Story 229 - Merge modules into finished app

### Sprint 6

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar

After discussion, the velocity of the team were estimated to be 48.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

* User Story 229 - Merge modules into finished app.

**Acceptance Criteria:**

1. App can connect to BLE device and collect data.
2. App can store data in SQL Server.
3. App can display data in graphs.

* User Story 225 - Add capability to gather and display data from Sunny device.

**Acceptance Criteria:**

1. The Android app can gather and display data from Sunny device.
2. The Android app works on Android version 6.0.

The team members indicated their willingness to work on the following user stories.

* Galo Romero
* User Story 225 - Add capability to gather and display data from Sunny device.
* Jordan Laing
* User Story 229 - Merge modules for a complete app.

# 

# 

# System Design

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The composition of the system is shown in a diagram and the subsystems are explained. Another diagram shows the physical deployment of the system and then, finally, the design patterns used in the project are discussed.

## Architectural Patterns

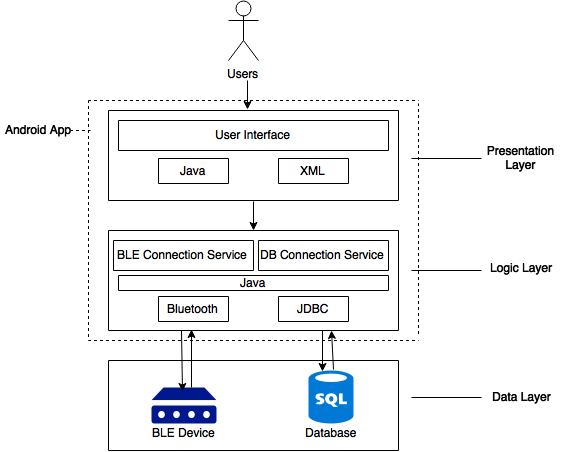
This system uses the multilayered architecture. There are three layers which from front-end to back-end are the Presentation layer, the Logic layer, and the Data layer. The Presentation and Logic layers together comprise the Android application, while the Data layer is composed of the BLE biosensing device and the database server. The user only ever interacts with the Presentation Layer’s UI, leaving the Logic layer to retrieve and store data from the biosensing device as well as retrieve data from the database to be displayed to the user.

This system also implements the client/server architecture as the Android app is the client that gives the user an interface to the biosensing device’s GATT server and the database running on the MS SQL Server.

## 

## 

## System and Subsystem Decomposition



Each of the three layers of this system can be seen as a subsystem of the whole.

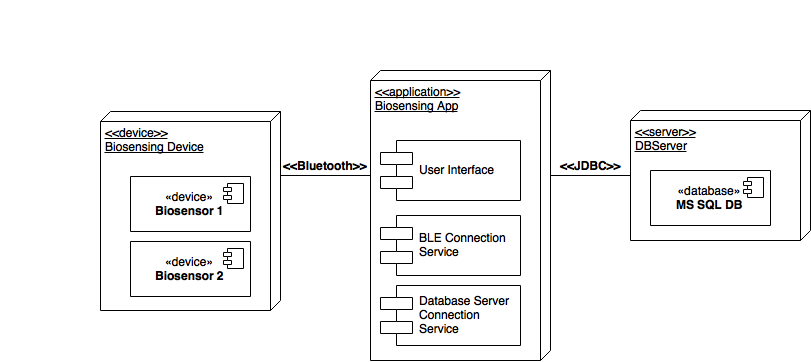
**User Interface:** Allows the user to control the system with touchscreen presses. Gives the user the options to scan for and connect to a device, initiating the connection services in the next layer. The UI displays to the user the state and measurements of the device’s biosensors and also allows the user to select what kind of graphs they would like to view. The UI is built in Java and XML.

**Logic**: Moves data between the UI and the Data subsystems as well as between the biosensing device and the database. This subsystem is made up of the BLE Connection service and the DB Connection service. They allow the app to connect to the biosensing device and the database, respectively. The BLE Connection service also manages the sensors on the device and formats the data from them to make it readable. Just like the first, this subsystem is coded in Java.

**Data**: Consists of the BLE biosensing device, where the data originates from, and the database, where the data is stored. In between this, the data moves through the Logic layer for processing. The database runs on a MS SQL Server.

## 

## Deployment Diagram



## 

## 

## Design Patterns

One design pattern we used in this system is facade. The intent of this design pattern is to wrap a complicated subsystem with a simpler interface, which is what we did with the Logic subsystem. All the code that makes up this subsystem is in one single class and all the complex procedures that it carries out are easily initiated by the user through the UI.

Another pattern we used is adapter. Because we used the open-source Graph View plotting library to implement the graphs in our app, we had to conform to the data types that library expected. We solved this by wrapping certain classes in other interfaces.

The last pattern we used is the command pattern. Following this pattern, user actions are encapsulated as objects and sent to methods that handle them.

# System Validation

**User Story 185 - Connect Android app to BLE device.**

* **Test Case ID: 185-001 (Sunny day)**
* Description/Summary of Test:
* Test if Android app can connect to BLE device.
* Pre-condition:
* Android app is installed on mobile device.
* Mobile device has bluetooth capability.
* Expected Results:
* The Android app connects to BLE device.
* Actual Result:
* The user connects to BLE device.
* Status (Fail/Pass):
* Pass.
* **Test Case ID: 185-002 (Rainy day)**
* Description/Summary of Test:
* Notify the user that the mobile device does not support Bluetooth.
* Pre-condition:
* Android app is installed on mobile device.
* Expected Results:
* Display message that Bluetooth is not supported
* Actual Result:
* Bluetooth not supported message displayed.
* Status (Fail/Pass):
* Pass.

**User Story 195 - Write simple android app that connects to BLE device.**

* **Test Case ID: 195-001**
* Purpose: To test if the mobile app can connect to the SensorTag.
* Preconditions:
  + Evothings Studio installed on computer
  + Evothings Viewer installed on Android mobile device
  + Mobile device has bluetooth capacity and it is enabled
  + TI SensorTag is on
* Expected Result: The app connects to the SensorTag and displays the collected data.
* Actual Result: The app did not connect to the SensorTag.

**User Story 199 - Extract data from Android app.**

* **Test Case ID: 199-001 (Sunny Day)**
* Description/Summary of Test:
* Test if Android app can retrieve data from BLE device.
* Pre-condition:
* The Android app is already connect to a BLE device.
* Expected Results:
* Android app displays services and characteristics.
* Actual Result:
* Services and characteristics are displayed.
* Status (Fail/Pass):
* Pass.
* **Test Case ID: 199-002 (Rainy Day)**
* Description/Summary of Test:
* Display data from unknown UUIDs as HEX.
* Pre-condition:
* The Android app is already connect to a BLE device.
* Expected Results:
* Android app displays data in HEX format.
* Actual Result:
* Android app displays data in HEX format.
* Status (Fail/Pass):
* Pass.

**User Story 203 - Display line graph on app.**

* **Test Case ID: 203-001**
* Purpose: To test if the mobile app can display a line graph of randomly generated data.
* Preconditions: Android application package installed on mobile device
* Expected Result: The app generates 100 random numbers between 70 and 80 and displays a line graph with the values plotted on it.
* Actual Result: The app successfully generated the test data and displayed the graph.

**User Story 207 - Integrate MS SQL Server with R services.**

* **Test Case ID: 207-001**
* Purpose: To test if the mobile app can send temperature data to the server.
* Preconditions:
  + Android application package installed on mobile device
  + SQL Server running on same local network as mobile device
* Expected Result: Whenever the app detects a temperature change, it sends the current temperature to the server, which adds a timestamp to it
* Actual Result: The app fails to start successfully (android version problem)

**User Story 208 - Format gathered data from device.**

* **Test Case ID: 208-001 (Sunny Day)**
* Description/Summary of Test:
* Test if Android app formats BLE device data and displays it correctly.
* Pre-condition:
* Android app is connect to BLE device and the app can gather data from it.
* Expected Results:
* Android app displays formatted BLE data.
* Actual Result:
* The data collected from the BLE device was formatted correctly.
* Status (Fail/Pass):
* Pass.
* **Test Case ID: 208-002 (Rainy Day)**
* Description/Summary of Test:
* Test what happens if Android app can’t collect data from BLE device.
* Pre-condition:
* Android app is connect to BLE device and the app can gather data from it.
* Expected Results:
* Android app data field shows no data string.
* Actual Result:
* The Android app displays the correct message.
* Status (Fail/Pass):
* Pass.

**User Story 216 - Pull data from server and display in graph.**

* **Test Case ID: 216-001**
* Purpose: To test if the mobile app can pull temperature data from the server and display it in a line graph.
* Preconditions:
  + Android application package installed on mobile device
  + SQL Server running on same local network as mobile device
* Expected Result: The app pulls the temperature data stored in the server and displays it in a line graph.
* Actual Result: The app successfully pulls the temperature data and displays the line graph.

**User Story 220 - Add current equation to the app.**

* **Test Case ID: 220-001 (Sunny Day)**
* Description/Summary of Test:
* Test if the EquationDialog changes the values for Vref\_div and Rtia correctly.
* Pre-condition:
* Android app is connect to BLE device and can gather data from it.
* Expected Results:
* A message is displayed with the updated values.
* Actual Result:
* Values were shown correctly.
* Status (Fail/Pass):
* Pass.
* **Test Case ID: 220-002 (Rainy Day)**
* Description/Summary of Test:
* Test if no values are entered for Vref\_div and Rtia, then the values should change to zero.
* Pre-condition:
* Android app is connect to BLE device and can gather data from it.
* Expected Results:
* A message is displayed with the updated values.
* Actual Result:
* Values were shown correctly.
* Status (Fail/Pass):
* Pass.

**User Story 225 - Add capability to gather and display data from Sunny device.**

* **Test Case ID: 225-001**
* Description/Summary of Test:
* Test if Android app runs on Android version 6.0.
* Pre-condition:
* Android app is installed on Android device.
* Expected Results:
* The Android app ask for location access and if user selects ok, it can detect BLE devices.
* Actual Result:
* The Android app scans and finds BLE devices.
* Status (Fail/Pass):
* Pass
* **Test Case ID: 225-002**
* Description/Summary of Test:
* Test if Android app can gather and display data from Sunny device.
* Pre-condition:
* Android app is connect to Sunny device.
* Expected Results:
* The Android app can gather and display data from Sunny device.
* Actual Result:
* The Android app receives and displays data from Sunny device.
* Status (Fail/Pass):
* Pass

**User Story 229 - Merge modules into finished app.**

* **Test Case ID: 229-001**
* Purpose: To test if the mobile app can connect to a BLE biosensing device’s sensors, store the data it collects in a server, and display graphs with data pulled from the server
* Preconditions:
  + Android application package installed on mobile device
  + SQL Server running on same local network as mobile device
  + A BLE biosensing device (such as the T.I. SensorTag) is nearby and turned on
* Expected Result:
  + The app connects to the device’s temperature sensor and pushes the data to the server every 30 seconds
  + The app displays a user-given range of temperature data in a line graph
* Actual Result:
  + The app successfully connects to the T.I. SensorTag’s temperature sensor and pushes temperature data to the server every 30 seconds.
  + The app successfully displays a line graph with that sensor’s data within the range of date and times the user specifies

# Glossary

**Biosensor** - Biosensors are analytical devices that convert a biological response into an electrical signal. These signals are processed by a tiny microcontroller in the sensor to yield measurements. An example is a blood glucose biosensor that breaks down blood glucose and measures the resulting electric current to yield a reading of the concentration of glucose.

**Bluetooth Low Energy (BLE)** - is designed to provide significantly lower power consumption. This allows Android apps to communicate with BLE devices that have stricter power requirements, such as proximity sensors, heart rate monitors, and fitness devices. [2]

**Generic Attribute Profile (GATT)** - The GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a BLE link. All current Low Energy application profiles are based on GATT. [2]  
  
**Attribute Protocol (ATT)** - GATT is built on top of the Attribute Protocol (ATT). This is also referred to as GATT/ATT. ATT is optimized to run on BLE devices. To this end, it uses as few bytes as possible. Each attribute is uniquely identified by a Universally Unique Identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information. The attributes transported by ATT are formatted as characteristics and services. [2]

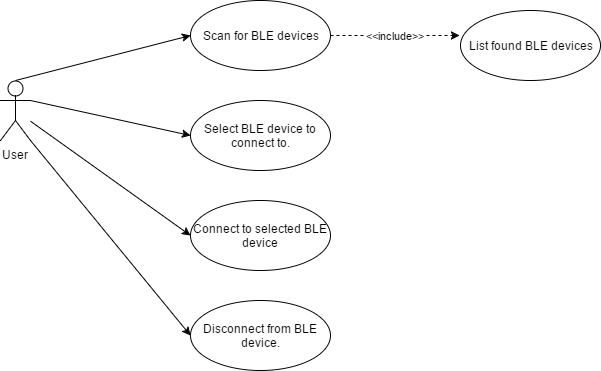
**Characteristic** - A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class. [2]

**Descriptor** - Descriptors are defined attributes that describe a characteristic value. For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure that is specific to a characteristic's value. [2]

**Service** - A service is a collection of characteristics. For example, you could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement." [2]

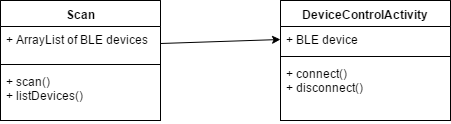
# Appendix

## Appendix A - UML Diagrams

**Figure 1 - Use Case Diagram for User Story 185.**

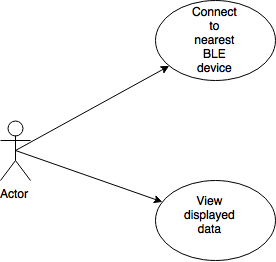
SequenceDiagram.png

**Figure 2 - Sequence Diagram for User Story 185.**

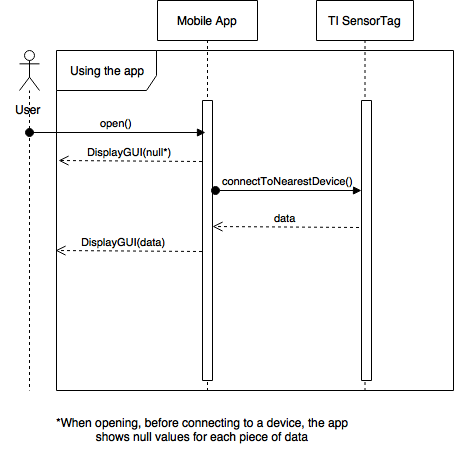


**Figure 3 - Class Diagram for User Story 185.**

## 



**Figure 4 - Use Case Diagram for User Story 195.**



**Figure 5 - Sequence Diagram for User Story 195.**

## 

## 

## UseCaseDiagram.png

**Figure 6 - Use Case Diagram for User Story 199.**

SequenceDiagram.png

**Figure 7 - Sequence Diagram for User Story 199.**

ClassDiagram.png

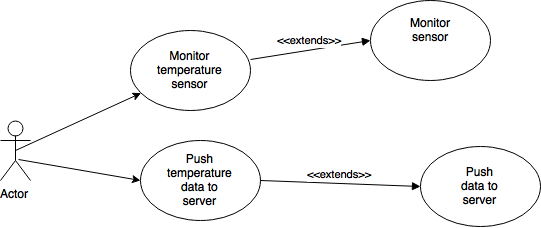
**Figure 8 - Class Diagram for User Story 199.**

sprint 3 use case diagram.png

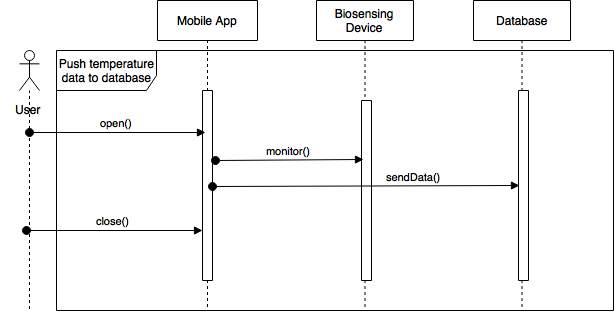
**Figure 9 - Use Case Diagram for User Story 203.**

sprint 3 sequence.png

**Figure 10 - Sequence Diagram for User Story 203.**



**Figure 11 - Use Case Diagram for User Story 207.**



**Figure 12 - Sequence Diagram for User Story 207.**

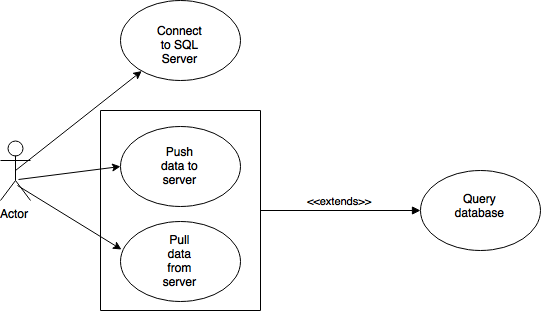
UseCaseDiagram.png

**Figure 13 - Use Case Diagram for User Story 208.**

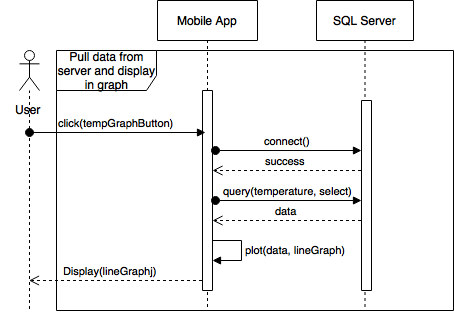
## 

## 

**Figure 14 - Sequence Diagram for User Story 208.**SequenceDiagram.png



**Figure 15 - Use Case Diagram for User Story 216.**



**Figure 16 - Sequence Diagram for User Story 216.**

**Figure 17 - Use Case Diagram for User Story 220.**UseCaseDiagram.png

## SequenceDiagram.png

**Figure 18 - Sequence Diagram for User Story 220.**

## UseCaseDiagram.png

**Figure 19 - Use Case Diagram for User Story 225.**

**Figure 20 - Sequence Diagram for User Story 225.**SequenceDiagram.png

## 

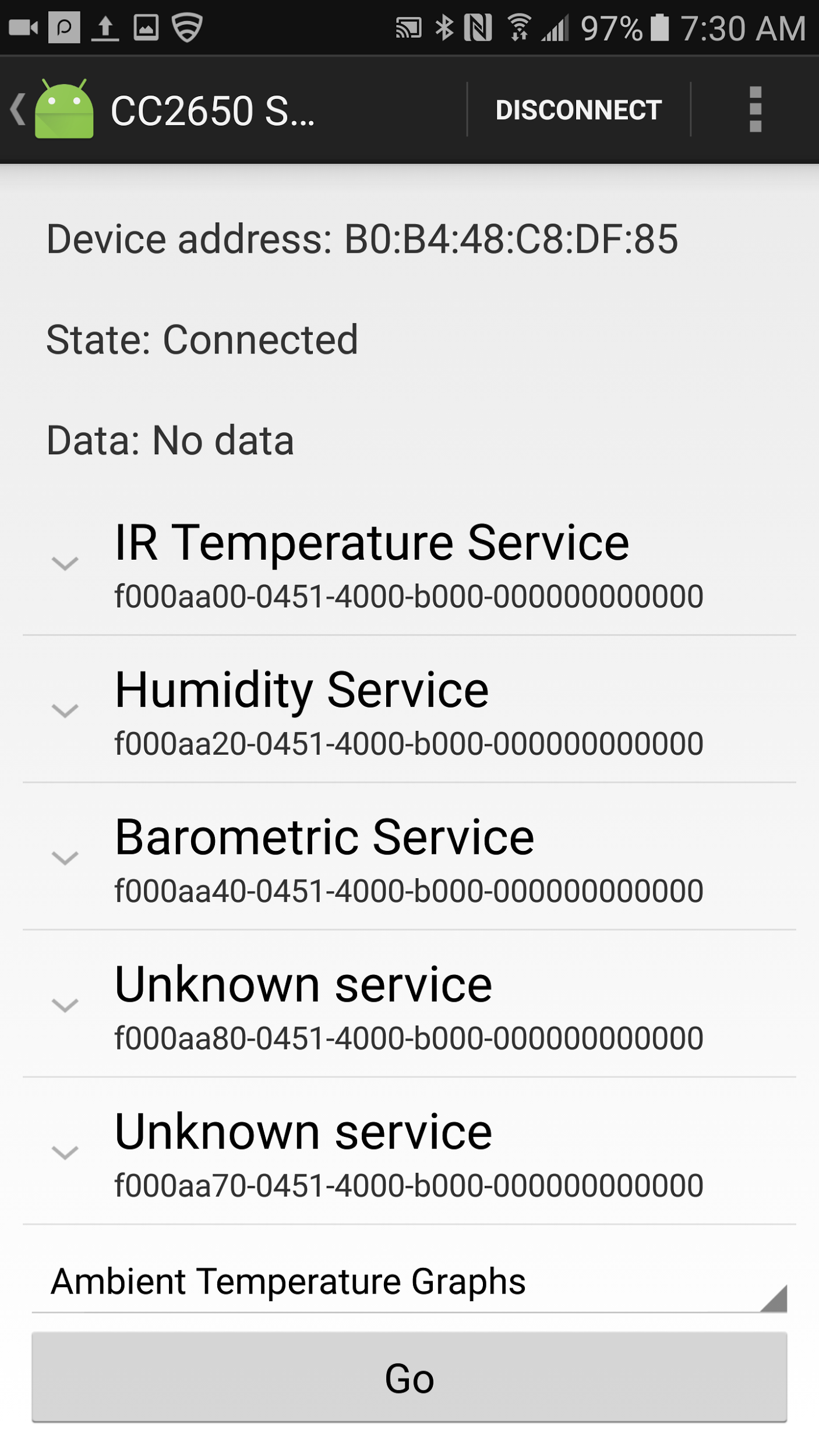
## 

**Figure 21 - Use Case Diagram for User Story 229.**

## 

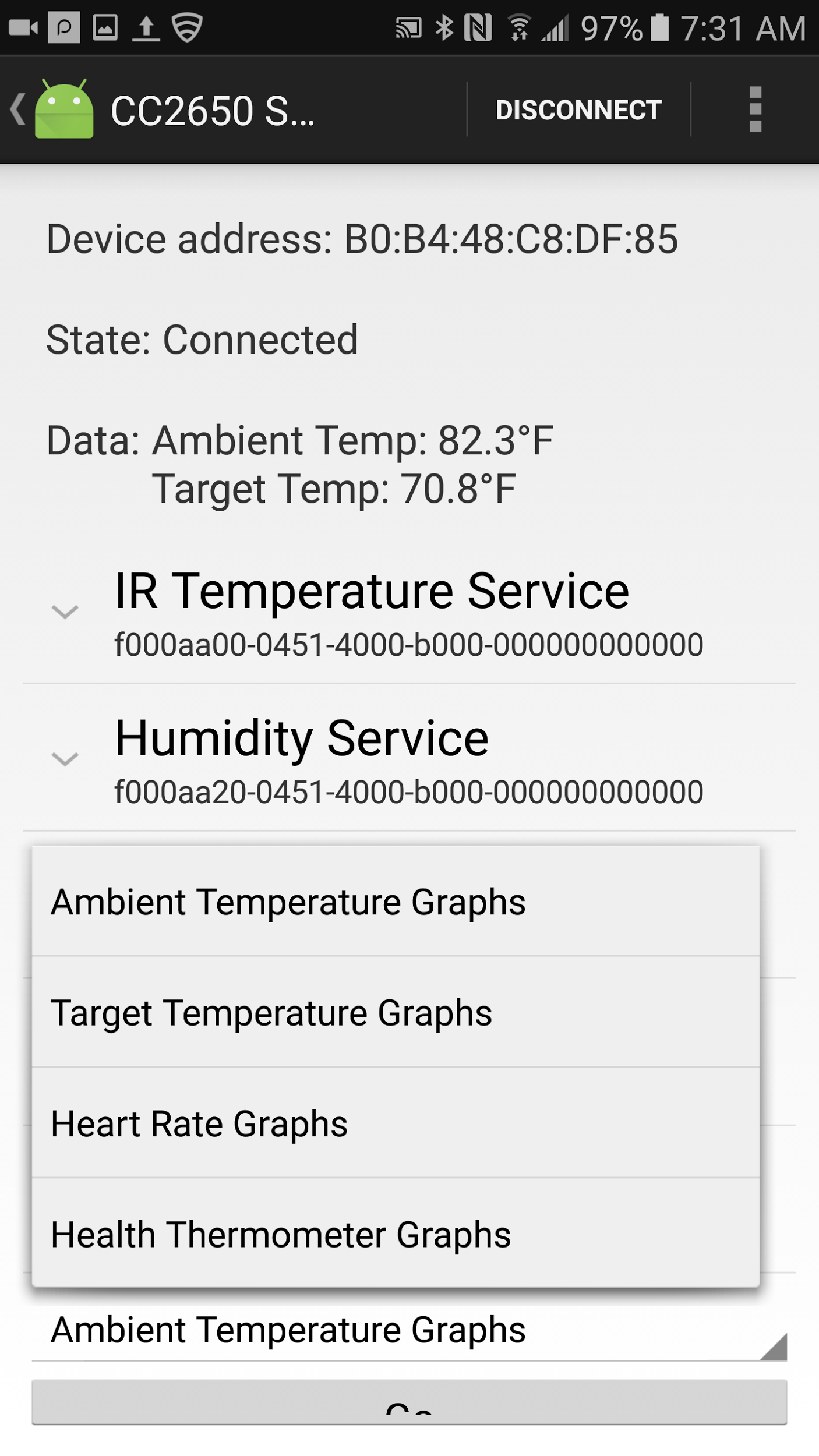
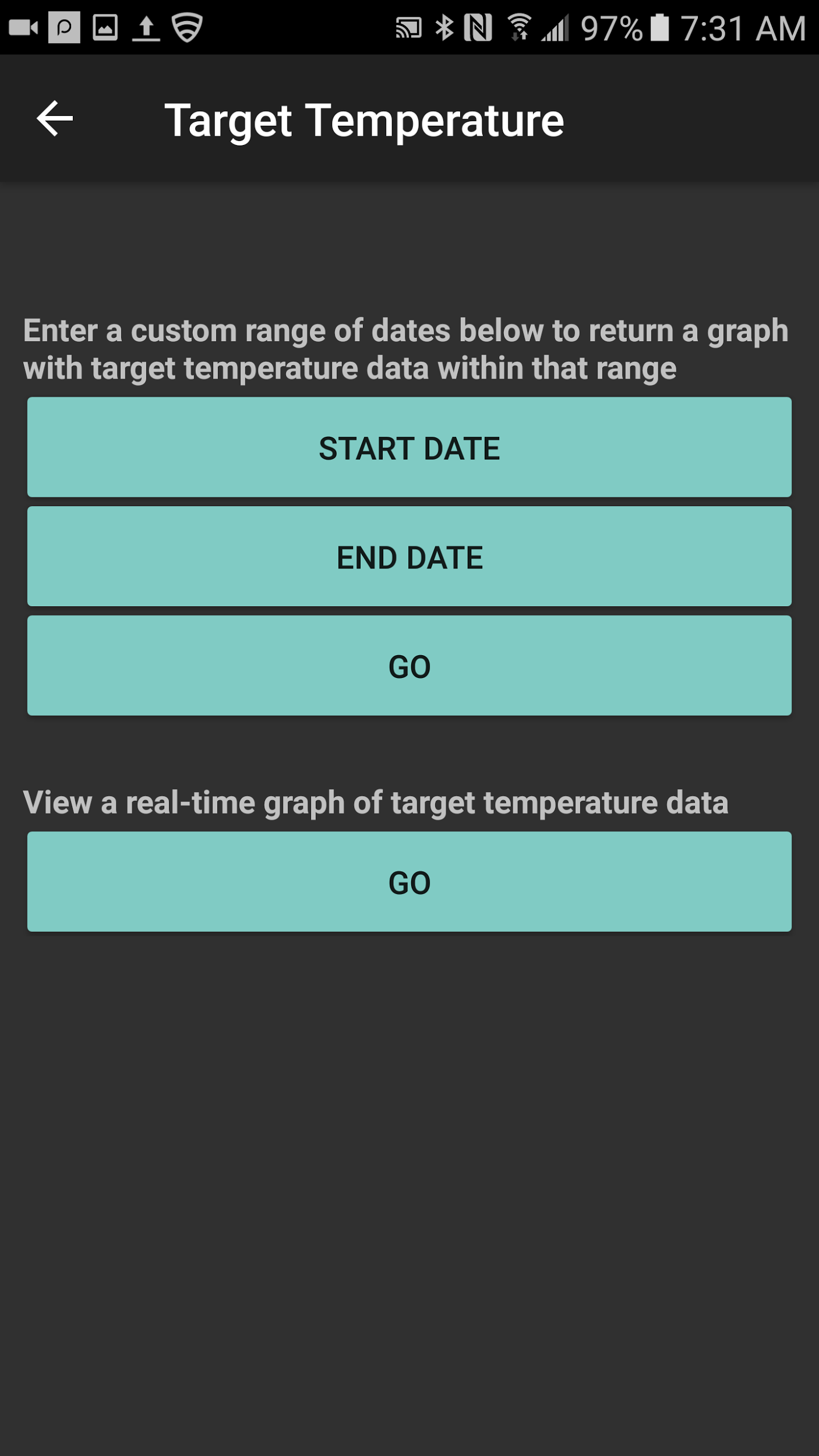
**Figure 22 - Sequence Diagram for User Story 229.**

## Appendix B - User Interface Design

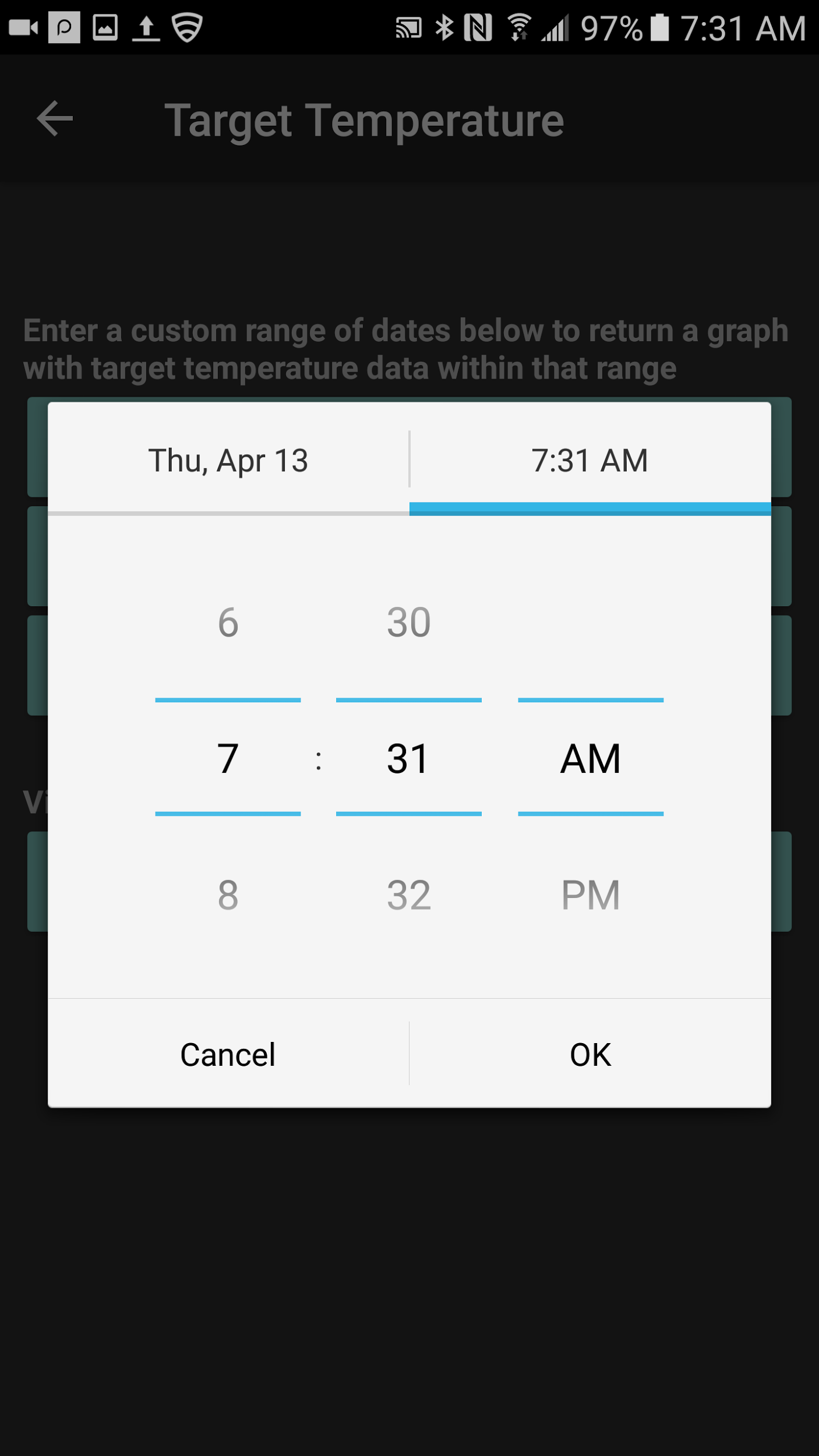


## 

## 



## 



## 

## Appendix C - Sprint Review Reports

**Sprint 2**

Date: 02/10/2017

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar, Vishal Chopade

Start time: 3:20 PM

End time: 4:17 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

* User Story 185 - Connect Android app to BLE device.

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

* User Story 203 - Display line graph on app.
* How this should be reflected on the user story definition in Mingle:
  + The app must show the raw data collected from the device for example: temperature, humidity, voltage, etc.

**Sprint 3**

Date: 02/24/2017

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar, Renny Edwin Fernandez

Start time: 3:00 PM

End time: 4:15 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story 199 - Extract data from Android app.
* User Story 203 - Display line graph on app.

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

* None.

**Sprint 4**

Date: 03/10/2017

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar

Start time: 3:00 PM

End time: 4:00 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story 207 - Integrate MS SQL Server with R services.
* User Story 208 - Format gathered data from device.

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

* None.

**Sprint 5**

Date: 03/24/2017

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar

Start time: 4:30 PM

End time: 5:30 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story 216 - Pull data from server and display in graph.
* User Story 220 - Add current equation to the app.

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

* User Story 229 - Merge modules into finished app.
* How this should be reflected on the user story definition in Mingle:
  + The User Story was moved back to the product backlog because the Biosensing app would not run on Android version 6.0.

**Sprint 6**

Date: 04/07/2017

Attendees: Galo Romero, Jordan Laing, Yogeswaran Umasankar

Start time: 3:30 PM

End time: 4:30 PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

* User Story 229 - Merge modules into finished app.
* User Story 225 - Add capability to gather and display data from Sunny device.

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

* User Story 233 - Make sensors send data to server on BLE connection.
* User Story 234 - Fix disconnections and crashing errors.
* How this should be reflected on the user story definition in Mingle:

Both of these user stories were moved to the Product Backlog.

## Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

**User Manual**

Using the Biosensing app is fairly simple. First, make sure the BLE device you want to connect to is nearby and available and that the database server is running on the same local network as the mobile device that the app is running on. After you have done that and opened the app, tap the button in the upper right corner to scan for nearby BLE devices. If the BLE device you are trying to connect to is nearby and available, it will be listed on the screen. Tap on its name to connect to it and be redirected to the next screen.

This screen lists the “services”, or sensors that the device has and that the app recognizes, such as Temperature Service or Heart Rate Service. Wait a few seconds for the app to discover the sensors on that device and then tap on the name of the service you want to connect to do so.

Currently this app supports the T.I. SensorTag’s temperature service that reads both ambient and target temperature and the Sunny device’s (the BLE device this app was designed to be used with) heart rate and health thermometer services.

Once you are connected to a sensor, the app will automatically begin pushing data from that sensor to the database every 30 seconds. This is why if your database server is not running, the app will crash 30 seconds after connecting to a sensor. Tapping the name of the service will open a drop-down menu with the “characteristics” of the service that it recognizes. Tap on the characteristic that ends with “Data” to cause the app to start displaying live data at the top of the screen.

This screen also has a menu at the bottom that pops up when you tap on it and allows you to select which sensor’s graphs you would like to view. You can view graphs from any of the sensors that the app currently supports by tapping on its name and then tapping the Go button. This will redirect you to another screen that gives you the option of viewing two line graphs.

The first is a line graph with data pulled from the database in a range of dates and times that you specify. Just tap the Start Date and End Date buttons to cause the date and time pickers to pop up. After you have selected a start and ending date and time, tap the Go button. The app will get the data in the range you specified and plot it on a line graph that is displayed to you.

The second line graph you can view is real-time graph of the selected sensor’s readings. First, the app pulls the most recent 20 values from the database and plots them on a line graph. Then, every 30 seconds it adds the current reading from the sensor to the graph.

**Installation Guide**

To run this system, you must have Android Studio, the Java SDK 8, Microsoft SQL Server 2016, and Microsoft SQL Server Management Studio (SSMS) installed on your computer. You must have access to the entire project folder. Note its location. You also must have the Android device you are installing the app on and a USB connector for it.

First, open Android Studio. If it is your first time using the IDE, a small window will pop up in the center of your screen that gives you the option to Open an existing project. Click it. If not, click File -> Open. Either way, you can now navigate to the location the project folder is located on your computer, Double-click on it and then double-click on the file called “Biosensing”, which the IDE will recognize as an Android Studio project. Android Studio will now automatically begin downloading files and code libraries that the project depends on. This may take several minutes, or even hours, depending on your Internet connection speed.

Before you can run the app, you must also set up the database. Open SSMS and connect to the server using Windows authentication. In the Object Explorer on the left side, right-click on the Databases folder and then click Restore Database. In the pop-up window, in the section to fill out the Source, click the “...” and navigate to the location of the project folder and double click on the file “biosensor.bak”, press OK, and then press OK again. This will create a database named “biosensor” on your database server.

Now, to allow the app to access the database, create a new login on the server by expanding the Security folder and right-clicking the Logins folder under it. Select New Login and as the Login name enter “client”. Check the SQL Server authentication bubble and uncheck the Enforce password policy bubble. As the Password enter “bio” and press OK. Now expand Databases and right-click on biosensor. Select New Query and in the window that pops up, type the following query exactly as follows:

USE biosensor ALTER USER client WITH LOGIN = client

At the toolbar at the top, press Execute. This has associated the new login you just created with the user of the same name that is preprogrammed into the database.

You are now ready to install the app onto your Android device and run it. Make sure the database server is running and the BLE device you are connecting to is on and available. Plug one end of the USB into the Android device and the other into your computer. Follow the device’s user manual to enable developer options on the device. In Android Studio, at the top of the page, click Run -> Edit Configurations -> + -> Android app. On the right side of the window, select “app” as the Module and select USB device. Then click OK to close that window and click the green Play button at the top of the page. The app is now being installed on your Android device. Wait a few minutes for the process to complete.

**Shortcomings/Wishlist**

One obvious shortcoming of the Biosensing app is that it can only be connected to one device’s sensor at a time. If we had more time, we would like to solve this problem, probably using multithreading. The app also crashes or disconnects from the BLE device sometimes. This is also a problem that we would have liked the time to solve.

Lastly, looking back, the app would have been more accessible to users if a SQLite database was implemented instead of a SQL database residing on a server. This is because a SQLite database could have been stored on the device itself and so would not have required the user’s Android device to have Internet access.

# 

# References

[1] Gehring, Jonas. *GraphView - open source graph plotting for Android.* (http://www.android-graphview.org/)

[2] Android Developers. *Bluetooth Low Energy.* (https://developer.android.com/guide/topics/connectivity/bluetooth-le.html#terms)

[3] *CC2650 SensorTag User's Guide.* (http://processors.wiki.ti.com/index.php/CC2650\_SensorTag\_User's\_Guide#Calibration)

[4] Bluetooth. *GATT XML.* (https://www.bluetooth.com/specifications/gatt)