



12/9/2018

# EKG Machine Learning

CMSC 495 Current Trends & Projects in CS

**Group 1 - Team <3 AI**

Deograstius Kalule, Jeffrey Liott, Timothy Patat, Jon Simmons, Andrew Young

# Table of Contents

<b>Introduction</b>	3
<b>Progress Report</b>	4
<b>Specifications</b>	7
EKG Machine Learning Architecture	10
<b>User Guide</b>	11
Installation	11
User Sign Up and Login	11
Read Your Live EKG Chart	11
View Information about Various Heart Conditions	12
View EKG History	12
Delete EKG History	13
<b>Scenarios / User Stories</b>	14
<b>Product Requirements</b>	16
<b>Test Plan</b>	21
<b>Resource Requirements</b>	24
Hardware Requirements	24
Software Requirements	24
<b>Risk Management</b>	26
<b>Milestones</b>	28
<b>Appendix A – EKG Device Schematic</b>	29
<b>Appendix B – HeartConMon Documentation</b>	30
<b>Appendix C – Heartbeat API</b>	34
Account	34
Fields:	34
Mutations:	34
Queries:	34
Archive	34
Fields	34
Queries:	34
Device	35
Fields:	35
Mutations:	35

Queries: .....	35
Record.....	35
Fields: .....	35
Methods: .....	35
Session .....	36
Fields: .....	36
Methods .....	36
Return Types.....	36
Archive.....	37
Record .....	38
Session.....	38
<b>Appendix D – GraphQL Schema .....</b>	<b>39</b>
<b>Appendix E - DynamoDB Database Diagram .....</b>	<b>41</b>
<b>References .....</b>	<b>42</b>

## Introduction

Healthcare is a crucial concern and a massive expenditure with over \$3.3 trillion spent on healthcare in the U.S. alone (CMS, 2016). Today, individuals are looking for greater control and data transparency in regards to their own health. While there are numerous health-related applications as well as heart rate devices such as those by Fitbit, none that we are aware of offer an electrocardiogram (EKG) diagnosis of their own heart.

Our team plans to create a Raspberry Pi-based EKG and Android application for self-monitoring of an individual's heart for particular heart conditions. Physionet EKG data will be

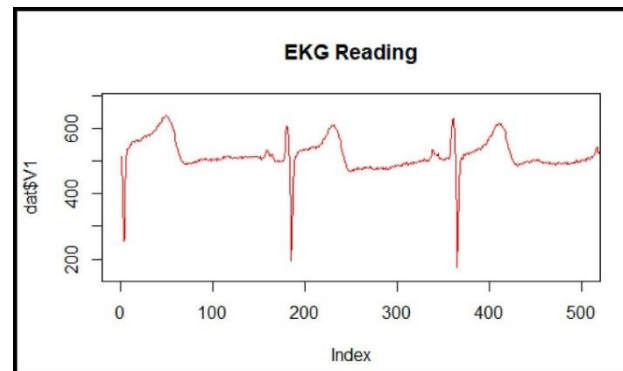


Figure 1. EKG data from our device.

utilized to train a machine learning model which will be able to recognize one or more heart conditions. The model will then be stored on an Amazon Web Services (AWS) Sagemaker service, with user data sessions stored in AWS DynamoDB.

The Android application will chart live user data (Figure 1) captured from the EKG device (Figure 2) after it is sent to AWS. The machine learning model will then be applied against user session data and will return results back to the user, marking each heart beat with any condition detected. The application will then display the number of beats matching any particular conditions found. Historic sessions will also be viewable in the application, including restreaming of the chart. Additionally, the application will display the individual's heart rate as well as informational pages about various heart conditions, including an animated EKG chart typical of each condition.

## Progress Report

At this point the team is slightly behind schedule, largely due to our AWS account getting abused and many members with busy work schedules. Someone retrieved our AWS account and ran up an overnight bitcoin farming routine. We are trying to get as much done as possible by end of day today since we all finally have time and AWS is up and running again. Here is where we're at based on the original milestones, as requested (Table 1). We've since expanded our milestones a bit more to include the hardware as well as additional details such as retrieving user chart history (Figure 7).

*Table 1. Progress meeting milestones.*

Milestone	Target Completion	Current Completion	Milestone Met?
Application design	100%	100%	Yes
Application development	100%	80%	No
Discuss improvements / security testing	100%	100%	Yes
Train and test model	100%	100%	Yes
Integrate ML with AWS backend	100%	50%	No
Users can log in and out through Cognito pool	100%	100%	Yes
User information is being sent to and from app	100%	100%	Yes
IoT core data preprocess. Streaming ECG data	100%	50%	No
Integration with other application components	100%	30%	No
OS install (hardware)	100%	100%	Yes
Hardware specification	100%	100%	Yes
Breadboard prototype	100%	100%	Yes
Device MQTT connection	100%	80%	No
Perfboard prototype / finalize integration	50%	100%	Yes

We've made some minor modifications as we've progressed, but have mostly stuck with the original plan. One such modification was getting rid of AWS SNS notifications for heart conditions detected. We realized that machine learning for ECG diagnosis is best performed on individual beats, and as such it is better to send that information back to the client along with each beat's data (mV) values through the application's interaction with DynamoDB, thus slightly simplifying the project. An additional benefit is that the heart rate can more easily be determined with the heartbeats found via machine learning, which tends to be more accurate than identifying heart beats through code logic.

On the application side, we decided to display information screens by simply displaying an initial normal ("sinus rhythm") heartbeat, then offering a menu in the upper right corner to traverse other heart condition information screens. This allows for simpler, more user-friendly navigation versus an additional heart condition selection screen, which would require the user to move back and forth between the heart conditions list and the detail screen. We also decided to display videos from Khan Academy for the condition information screens.

We are quite pleased with the decisions we have made so far, although working in a team online rather than in the same room with a whiteboard certainly has tremendous challenges. We've already had a number of miscommunications, which isn't surprising in a project this complexity, with so many hardware, software and data interfaces and details to work through.

Documentation changes include the addition of an AWS database diagram (Appendix E), GraphQL schema (Appendix D) and API documentation (Appendix C). Application screenshots have also been updated (Figure 3). Additionally, developer contact and responsibility details have been added to a table for easier access and viewing (Table 2).

Besides those items, other changes have mostly been to add in details as we determine them following initial research. This includes details about hardware and AWS service requirements, and then adding such details to our test plan, specifications, user guide and milestones. Images were added and changed to provide updated visual details on the overall plan. The application design was modified for an improved user experience as mentioned earlier. We also broke out “requirements” into “resource requirements” and “project requirements” as each is a very different type of requirement. Security details were added, and the milestones were updated with new details.

## Specifications

As mentioned in the introduction, this application will be receiving data from a custom EKG machine we create using Raspberry Pi, MCP3008 analog to digital integrated circuit, and an ADC8232 EKG board (Figure 2). See Appendix A for a detailed schematic. In a real-life product scenario, a “Device ID” will be included on the device and in the literature. This Device ID will be pre-populated in DynamoDB.

The EKG Analyzer application will allow the user to create a user account, requesting the user’s email address, password, and Device ID which will all be stored in a user table inside DynamoDB. User, device, and session data will be persisted in Amazon’s NoSQL database DynamoDB (Appendix E). This database will be accessed by both the machine learning instance, and multiple lambdas invoked by the client.

The bulk of the client interaction will be handled by a GraphQL API (Appendix C) contained within Amazon’s Appsync, and hosted in the cloud via AWS Cloudformation. The API will provide secure access to DynamoDB and Lambda service connections through AWS Cognito authentication. Each data mutation or query will be resolved using an AWS Lambda function. The API will also provide subscription capabilities to the client through an MQTT connection.

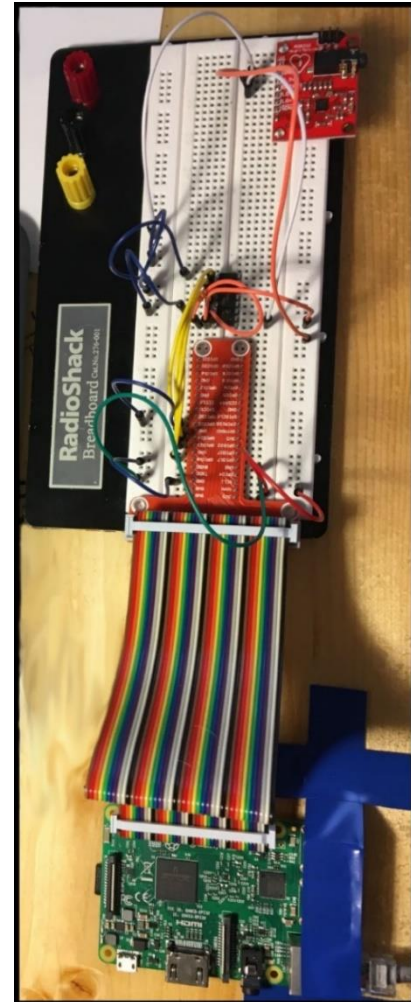


Figure 2. Raspberry Pi EKG device



Once logged in, the device will send the electrocardiogram data via secure MQTT WebSocket to AWS IoT. The device will publish and receive messages to specific channels in order to act appropriately. IoT will then store the data in DynamoDB as well as notify AWS SageMaker of new data to run through the machine learning model. Lambdas will be invoked to act as the transportation mechanism between the various services. SageMaker will then identify each heart beat and code each beat with a confidence level in one or more possible heart conditions identified by the model. The application will retrieve the data feed from DynamoDB through the GraphQL API and display the data to the user, including information about the types of conditions found in beats, quantified according to the machine learning's confidence in such findings. Sessions will also be stored in a history archive table on DynamoDB for later retrieval of the user's chart history from the application.

Details on the AWS database table schemas can be found in Appendix E. See Figure 4 for overall architecture design, which includes a significant number of AWS services and interfaces.

In the application, the user will be able to view a live chart, machine learning results, historic sessions, and information about various heart conditions (Figure 3).

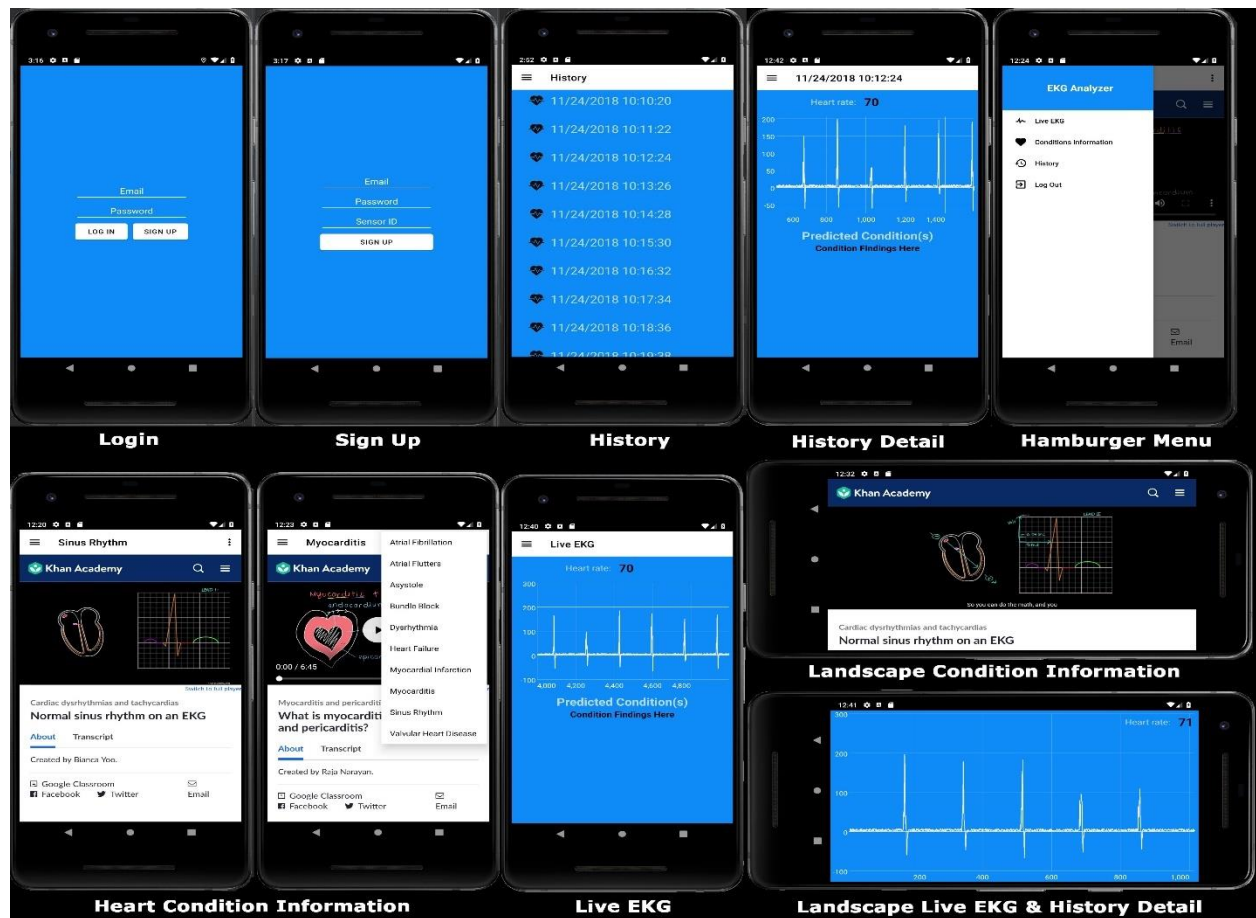


Figure 3. Mobile application screenshots.

## EKG Machine Learning Architecture

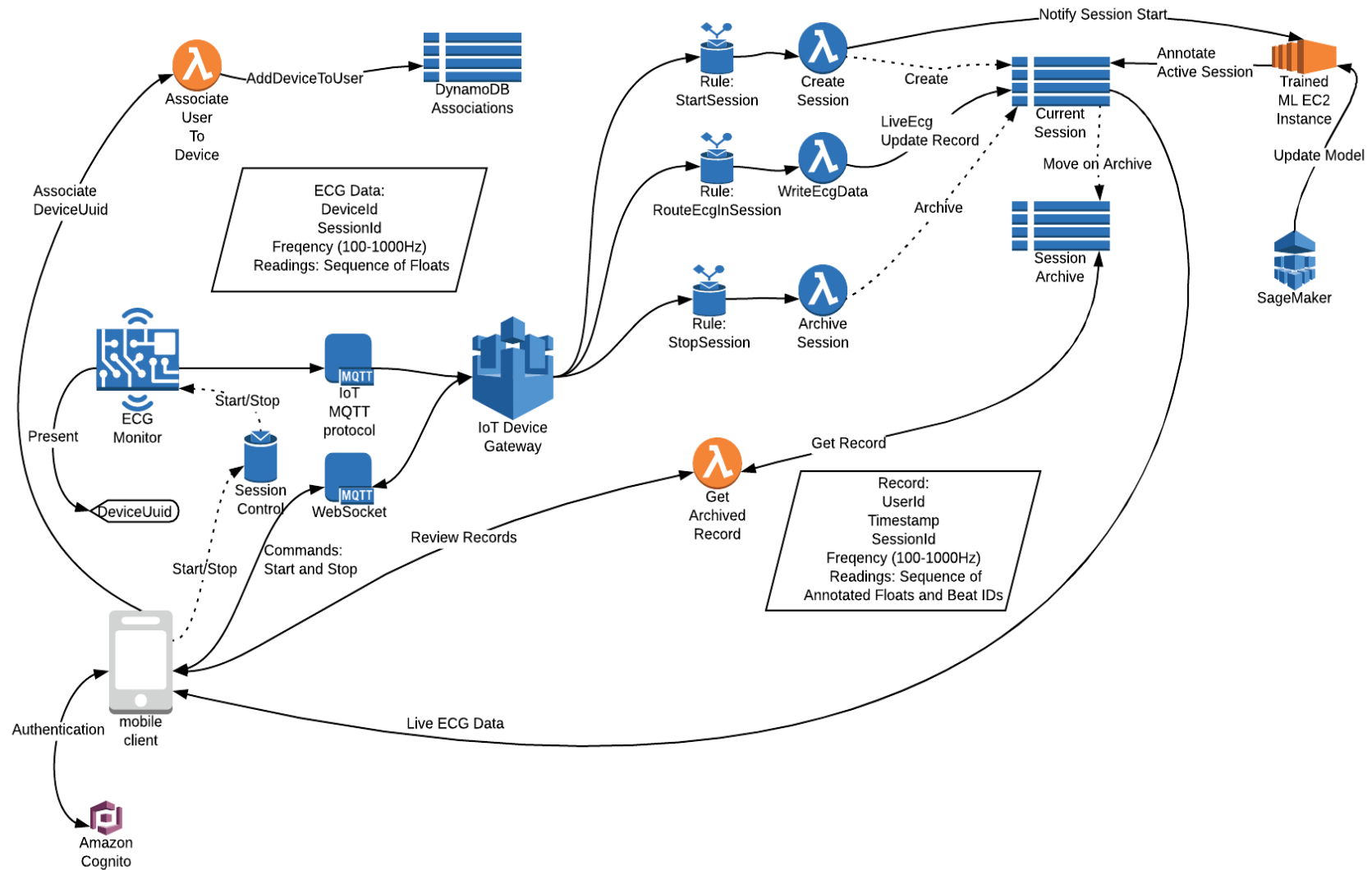


Figure 4. EKG machine learning architecture.

# User Guide

## Installation

Step 1: On your Android device, go to **Menu > Settings > Security >** and check **Unknown Sources** to allow your phone to install apps from sources other than the Google Play Store.

Step 2: Copy the included Android APK on to your device.

Step 3: Tap the APK on your android device to install the application.

Step 4: Follow instructions and accept required permissions.

## User Sign Up and Login

Step 1: Start the EKG Analyzer application on your Android device

Step 2: Click on the “Sign Up” button.

Step 3: When prompted, enter your email address, desired password, and the device’s unique sensor ID (your device label is printed on the device as well as on the front page of these instructions).

Step 4: Once you have a user account, enter your username (email) and password.

Step 5: Tap the “Log In” button.

## Read Your Live EKG Chart

Step 1: Plug the EKG cable into the EKG device (Figure 5) and ensure that the device is turned on.



Figure 5. EKG cable plug.

Step 2: Turn the Raspberry Pi EKG device on and ensure that it has a Wi-Fi Internet connection.

---

Step 3: One by one, remove the adhesive backing from each of the three ECG pads and place the pad marked “R” on your right arm, “L” on your left arm and “COM” on your right leg (Figure 6).

Step 4: Start the EKG Analyzer application on your smartphone.

Step 5: Click on the ≡ context menu icon in the upper left corner and select “Live EKG”.

Step 6: EKG Analyzer will now display real-time data of your heart rate and electrocardiogram (EKG) chart.

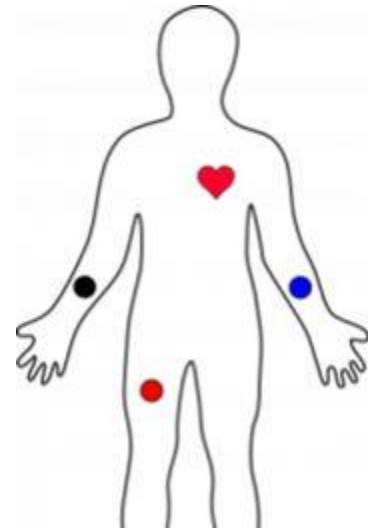


Figure 6. EKG pad placement  
(Sparkfun, N.d.)

## View Information about Various Heart Conditions

Step 1: Start the EKG Analyzer application on your device

Step 2: Tap the ≡ context menu icon and select “Conditions Information”

Step 3: Tap the ⋮ menu icon in the upper right corner

Step 4: Tap the heart condition you are interested in

Step 5: The EKG Analyzer will display animated EKG patterns and give the user information about the selected heart condition

## View EKG History

Step 1: Start EKG Analyzer application on your device and Login if necessary

OR


If the application is already running, tap the ≡ context menu icon and select “History”

Step 2: History menu will now be displayed

Step 3: Tap the historic EKG chart you wish to view

## Delete EKG History

Step 1: Start EKG Analyzer application on your device

Step 2: Tap the  context menu icon and select “Chart History”

Step 3: Tap the  edit button

Step 4: Select the user sessions you wish to delete

Step 5: Click on the “Delete” button

## Scenarios / User Stories

- As a user I want to be able to send my data online to be analyzed so that I can know if I have a heart condition
  - Acceptance Criteria
    - I am able to send EKG data online
    - I am able to tell that the data was analyzed online
    - I am able to get an accurate diagnosis
- As a user I want to be able to view my EKG information on my Android mobile device so that I can keep track of my EKG history and disease diagnosis
  - Acceptance Criteria
    - I am able to see a history of my EKG information
    - I am able to see able to see my heart rate
- As a user I want to be able to see live EKG results as they are read by an EKG machine on my android device so that I can see real time readings
  - Acceptance Criteria
    - I am able to see my heart rate change on the app
    - I am able to see the EKG wave change update
- As a user I want to be able to read about different heart conditions so that I can stay informed on those different EKGs
  - Acceptance Criteria
    - I am able to view information about different EKG conditions
- As a user I want to be able to see how the EKG wave looks like for different heart conditions

- Acceptance Criteria
  - I am able to see the different EKG waves for the selected heart condition
- As a user I want to be able to receive notifications so that I can be alerted if any heart conditions were found during the analysis
  - Acceptance Criteria
    - I am able to receive notifications about heart conditions
    - I am able to tap on a notification and view the information in more detail in the app



## Product Requirements

- 1) Create Android Application Login/Home Screen
  - a) Home Screen background color set to blue
  - b) Textfield created to enter user email address
  - c) TextPassword created to enter user password
  - d) Button created to “Log In”
    - i) Logs user in. Should trigger lambda to verify device is connected/ initialized.
  - e) Button created for “Sign Up”
- 2) Create Android Application New User Screen
  - a) Screen background color set to blue
  - b) Textfield created for “Sensor ID”
    - i) In a real-world application, device ID’s would be pre-printed on the device and packaging.
  - c) Textfield created for “Email Address”
  - d) TextPassword created for “Password”
  - e) Button created to “Create Account”
    - i) Should trigger lambda to create User account on AWS and tie the device ID to a UUID for the device via a pre-populated table of device ID’s.
    - ii) Logs user in and displays the Android App User EKG Data Screen
- 3) Create Android Application User EKG Data Screen
  - a) Data screen displayed after user logs in from the home screen
    - i) Client should establish connection to DynamoDB Stream

- ii) Upon first request, a lambda will store the device information into the user table.
  - iii) Client should store device data locally upon successful connection
- b) Turning phone sideways will display the chart in landscape view
- c) Hamburger button created for navigation
  - i) Hamburger provides context menu navigation to “Live EKG”, “History”, “<3 Condition Info” and “Log Out”
  - ii) Hamburger navigates to the appropriately titled screen
- d) Blue background screen created
- e) White navigation bar created
- f) View with Path created, generating an accurate, streaming ECG chart
  - i) Client should continually poll until leaving the screen
- g) Textview for heart rate is accurately displayed
- h) Diagnosis is displayed and matches the expected diagnosis
  - i) Diagnosis should come in the form of subscriber alerts from Amazon SNS
- 4) Create Android Application History Screen
  - a) List item created for each historical chart session, if any.
    - i) Sessions retrieved from DynamoDB through Cognito
    - ii) Clicking List item will display an Android History Detail screen
  - b) Button created for “Delete” at end of list
    - i) Should delete user sessions from the app screen and from DynamoDB
  - c) Hamburger button created for navigation

- i) Hamburger provides context menu navigation to “Live EKG”, “History”, “<3 Condition Info” and “Log Out”
  - ii) Hamburger navigates to the appropriately titled screen
- 5) Create Android History Detail Screen
  - a) Scroll View with Path created, displaying the historic data as a chart
  - b) Textfield with historic heartbeat created
  - c) Button created to “Delete”
    - i) Deletes the historic session from DynamoDB through Cognito
  - d) Textview created for historic heart condition
  - e) Hamburger button created for navigation
    - i) Hamburger provides context menu navigation to “Live EKG”, “History”, “<3 Conditions Info” and “Log Out”
    - ii) Hamburger navigates to the appropriately titled screen
  - f) Turning phone sideways will display the chart in landscape view
- 6) Create Specific Android Heart Condition Information Screen
  - a) A relevant heart condition information screen is displayed when any heart condition is selected from the <3 Conditions Info Screen
  - b) Hamburger button created for navigation
    - i) Hamburger provides context menu navigation to “Live EKG”, “History”, “<3 Conditions Info” and “Log Out”
    - ii) Hamburger navigates to the appropriately titled screen
  - c) Displays a video with information about the condition
  - d) Displays text describing the heart condition

- e) Blue background screen created
- f) White navigation bar created
- g) Menu in upper right corner created
  - i) Lists heart conditions
  - ii) Clicking item in list takes user to specific heart condition information screen
- 7) Clicking item in list takes user to specific heart condition information screen Heart Condition Machine Learning Model
  - a) Accepts raw electrocardiogram data text input of various frequencies (i.e. 200Hz) as its input
  - b) Processes and stores data into DynamoDB for client rendering
  - c) Triggers alerts via IoT rule upon potential condition detection
    - i) Detected conditions are logged to Dynamo to provide client history
  - d) Diagnoses heart conditions with 80%+ accuracy across various frequencies and for various conditions
- 8) Raspberry Pi EKG device reads a user's electrocardiogram (EKG)
  - a) Raspberry Pi, MCP3008, ADC8232 hardware and EKG pads function properly
  - b) Outputs accurate data reflecting the user's EKG reading in the proper format (line separated EKG values (double) sent at a 200hz frequency (200 values per second)).
  - c) Connects to the user's Wifi
  - d) Should establish connection to AWS IoT through MQTT channel, uploading the data accordingly

e) AWS IoT should act accordingly once conditions for a rule have been met. Rules subject but not limited to:

- i) Device is disconnected
- ii) Data cannot be interpreted

## Test Plan

Case	Requirement	Expected Output	Actual Output	Pass or Fail
1	<b>Requirement 1.</b> Create Android App Home Screen	<ul style="list-style-type: none"> <li>Home Screen background is blue</li> <li>"User Email" Textfield</li> <li>"Password" TextPassword</li> <li>"Login" button</li> <li>"New User" button</li> </ul>		
2	<b>Requirement 1(f) i.</b> User Login	<ul style="list-style-type: none"> <li>Logs user into application</li> <li>Verifies EKG device is connected.</li> </ul>		
3	<b>Requirement 2.</b> Create Android App New User Screen	<ul style="list-style-type: none"> <li>Screen background is blue</li> <li>"Email Address" Textfield</li> <li>"Password" TextPassword</li> <li>"Sensor ID" Textfield</li> <li>"Create Account" button</li> </ul>		
4	<b>Requirements 2(e) i, ii.</b> Create User	<ul style="list-style-type: none"> <li>Creates/stores new user account in DynamoDB with correct values</li> <li>Logs user into their new account</li> <li>Displays User EKG Data Screen</li> </ul>		
5	<b>Requirement 3.</b> Create Android App User EKG Data Screen Utilizing EKG Device for Data Input	<ul style="list-style-type: none"> <li>Screen background is blue</li> <li>White navigation bar</li> <li>Hamburger button</li> <li>Retrieves data from AWS DynamoDB</li> <li>Live, streaming, accurate EKG chart</li> <li>Heart rate</li> <li>Correct diagnosis (typically "Normal")</li> </ul>		
6	<b>Requirements 3(b) i, ii; 4(c) i, ii; 5(e) i, ii; 6(b) i, ii</b> Hamburger / Context Menu	<ul style="list-style-type: none"> <li>Includes "Live EKG", "History", and "&lt;3 Condition Info" menu items</li> <li>Every context menu item loads the correct Android screen</li> </ul>		

Case	Requirement	Expected Output	Actual Output	Pass or Fail
7	<b>Requirement 4.</b> Create Android App History Screen	<ul style="list-style-type: none"> <li>History Screen background is blue</li> <li>White navigation bar</li> <li>Historic user sessions properly retrieved and displayed in list</li> <li>Clicking a session will load a historic chart screen</li> <li>Edit-&gt; Delete button functionality created</li> </ul>		
8	<b>Requirement 4(b) i.</b> Edit-> Delete sessions from History	<ul style="list-style-type: none"> <li>Clicking the Edit button allows user to select and delete historic user sessions from the application as well as from DynamoDB</li> </ul>		
9	<b>Requirement 5.</b> Create Android App Historic Detail Screen	<ul style="list-style-type: none"> <li>Screen background is blue</li> <li>White navigation bar</li> <li>Hamburger button created</li> <li>Scrolling view with historic chart displayed</li> <li>Historic heart rate displayed</li> <li>Historic condition Displayed</li> <li>"Delete" button displayed</li> </ul>		
10	<b>Requirement 5(c) i.</b> Delete from Historic Session Screen	<ul style="list-style-type: none"> <li>Clicking "Delete" button deletes historic user session from the application as well as from DynamoDB</li> </ul>		
11	<b>Requirement 6.</b> Create Android App Heart Condition Information Screen	<ul style="list-style-type: none"> <li>Information Screen background is blue</li> <li>White navigation bar</li> <li>Hamburger button created for navigation</li> <li>Video describing the condition is displayed</li> <li>Text describing the heart condition displayed</li> <li>Menu created listing additional conditions</li> </ul>		

Case	Requirement	Expected Output	Actual Output	Pass or Fail
12	<b>Requirement 6(g) i, ii.</b> Click Conditions menu	<ul style="list-style-type: none"> <li>Menu displays additional heart conditions</li> <li>Clicking Item brings up a refreshed condition information screen for the selected condition</li> </ul>		
13	<b>Requirements 3(b) &amp; 5(f)</b> Turning screen sideways on charts	<ul style="list-style-type: none"> <li>White screen background</li> <li>Chart displayed in landscape view</li> <li>Heart rate Textfield displayed</li> </ul>		
14	<b>Requirement 7.</b> EKG Machine Learning	<ul style="list-style-type: none"> <li>Accepts data at various frequencies</li> <li>Stores data, including the chart data and any conditions detected into DynamoDB</li> <li>ML Algorithm provides 80%+ accuracy in predicting the correct diagnosis on all of the uploaded datasets at 200Hz frequency.</li> </ul>		
15	<b>Requirement 8.</b> Hardware powered on	<ul style="list-style-type: none"> <li>Hardware functions properly</li> <li>Establishes connection to AWS IoT through MQTT</li> <li>Connects to WiFi and the AWS IoT service</li> <li>Uploads data in the proper format</li> <li>AWS IoT reacts accordingly to improper data and condition detection</li> <li>Pertinent alerts are sent to the correct user over SNS</li> </ul>		
16	<b>Requirement 8(e).</b> Hardware powered off	<ul style="list-style-type: none"> <li>AWS IoT reacts accordingly to device disconnect, alerting user via SNS</li> </ul>		



## Resource Requirements

### Hardware Requirements

Our solution requires a wide variety of hardware components in order to create the EKG device, train the initial machine learning model, and run the application. For the EKG device it requires a Raspberry Pi with an AD8232 board, ECG electrode pads to obtain user ECG data, and an MCP3008 ADC integrated circuit for converting analog signals to digital signals.

In order to develop our machine learning models without running into high AWS expenses, we need a high-end computer with 16+ GB RAM, NVidia GeForce 1060 GTX or better video card running CUDA, and an i7-6700 or better CPU. Machine learning requires highly intensive computing power, including the use of high end video cards in order to test models within a reasonable timeframe.

Any modern Android phone or tablet with 8GB or more of RAM is sufficient to run the Android application.

We anticipate free usage of AWS services, and minimal costs for Arduino components, with the required components costing less than \$80. No additional expenses are anticipated.

### Software Requirements

Software requirements include Android Studio for application development, a Python and Java IDE for machine learning and Raspberry Pi development, and Amazon AWS services including EC2, Sagemaker, DynamoDB, and IoT to retrieve, store, and process user data to determine its relevance based on our machine learning

model. The application requires Android API version 28 or greater in order to run the EKG Analyzer application.

### Personnel Requirements

Five developers have been assigned to this project (Table 2). Andrew will focus on Raspberry Pi development and sending user data to AWS IoT. Jeff and Timothy will focus on the AWS backend and machine learning. Jon and Deo will focus on creating the Android application. All of us hope to learn a lot from this project so we will be watching and helping each other in the process. If one area becomes too easy or too much of a burden, we will move developers around as necessary and work together to resolve any issues. We will use the Kanban process through Trello in order to stay on top of work requirements and assignments.

*Table 2. Personnel.*

Developer	Responsibilities	Contact
Jon Simmons	Android Development, Documentation	zenwarrior01@gmail.com
Deograstius Kalule	Android Development	deograstius@gmail.com
Timothy Patat	AWS Backend	tpatat1@gmail.com
Jeff Liott	Machine Learning & AWS	jeffliott@gmail.com
Andrew Young	EKG Device & Code	kb3guy@gmail.com

## Risk Management

There are a few risks we have identified as we move forward with this product. First, there is the possibility that we are unable to create a reliable machine learning model. This risk is exacerbated by the fact that our training data from the Physionet PTB Diagnostic ECG Database utilizes medical grade equipment with 16 input channels (Physionet, N.d.), while the Arduino user device only outputs a single ECG signal resulting from 3 electrodes. Meshing the two datasets, as well as dealing with potential signal noise, will be a significant challenge. Our hope is that proper electrode placement, choice of data leads, and artificial noise generation will suffice in creating a usable signal. Initial testing indicates significant success when utilizing lead “I” from the 12 lead datasets and we are more confident that this will match up well with the 3-lead signal as we have obtained additional 3 lead data to test. We have already seen 90% accuracy in our training with 12 leads, and are approaching this again using the 3-lead data. Moreover, this has been in determining **all** of the heart conditions, rather than just one. However, we still need an actual working EKG device to compare this data with. Of course, the product will also still provide a nice ECG chart and other relevant information, so we will still have a viable product even without the machine learning.

Another risk is that some of the team does not have enough time to work on such an intensive project. We plan to mitigate this risk by working around the team members schedules and allowing them to do work when they can. We are also using an Agile approach and will be able to assign different stories to different team members so no two members are working on the same task at the same time.

An additional risk is that our AWS account is broken into or misused. In order to mitigate this risk, we need to set tight security and permissions on AWS, as well as set spending limits and proper notifications.

Lastly, potential hardware failure is a significant risk, whether it be the EKG device failing or computer failure while coding. If the EKG device is somehow broken or starts failing after much has already been completed, then we will likely be unable to finish the project. Initially we hoped to have a couple people making at least 2 different devices to mitigate this risk, but it requires a good amount of expertise to make the device correctly. Computer failure can be mitigated by regularly backing up the code to GitHub.

# Milestones

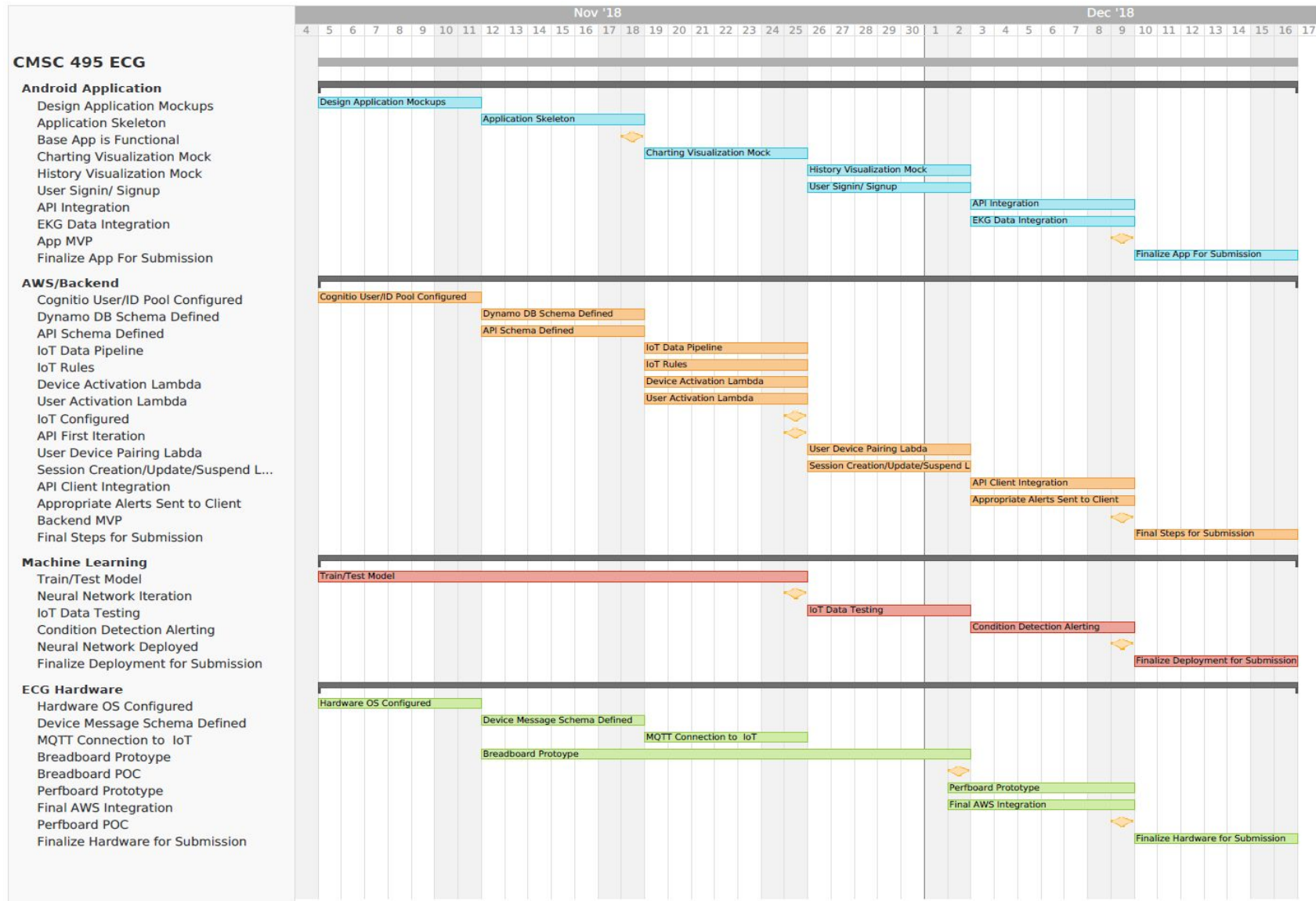
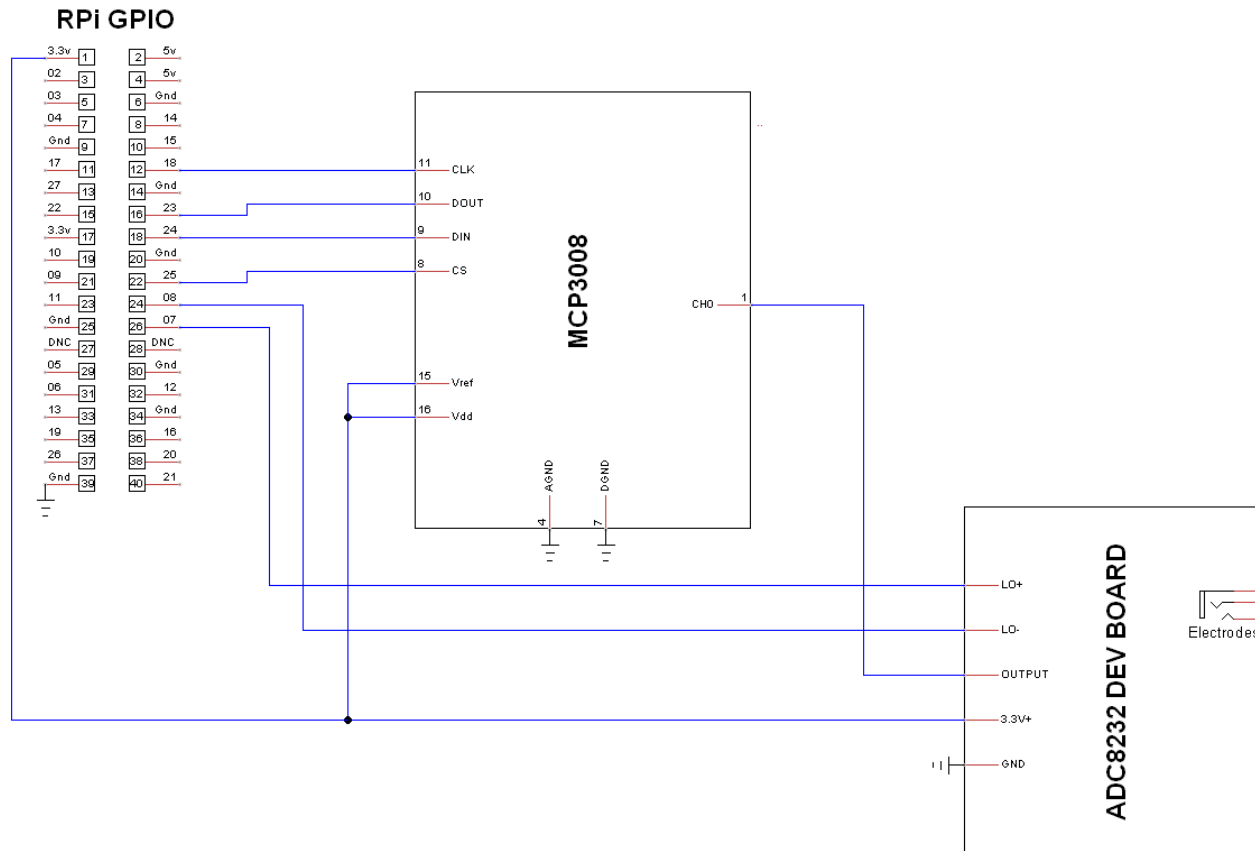


Figure 7. Milestones.

## Appendix A – EKG Device Schematic



Title <b>RPi Heart Rate Monitor Prototype</b>		
Author <b>Andrew Young</b>		
File C:\Users\Andrew Young\Dropbox\hwSchematic.dsn	Document	
Revision 1.0	Date <b>11/11/2018</b>	Sheets 1 of 1

## Appendix B – HeartConMon Documentation

### HeartConMon:

#### Module Description

The HeartConMon wraps its sub-modules (optionally) and provides utilities that allow external products to interface with the program overall.

#### Full Python Module Path:

heartconmon

#### Module Dependencies:

numpy  
keras  
cudnn  
tensorflow  
tensorflow-gpu  
cudatoolkit  
flask

#### Globals:

models = heartconmon.models.\_models

#### Interface:

predict

(ecg\_data=numpy.array[N], freq=200, model\_name='default')

Converts the given ECG data (ecg\_data) into segments that can be ingested by the model, collects the predictions for all of the segments, averages the results, and returns the indices that are the center of the strongest diagnostic criteria the model identified for the top three conditions the data suggests.

Returns a tuple of the following data:

indices - A list of up to 10 indices in the given data that are the center of the the diagnostic criteria.

diagnostics - A list of tuples of the most probable conditions and the strength of each estimation.

### Models:

#### Module Description

The library of model constructions, weights, and utilities to interact with them.

#### Full Python Module Path:

heartconmon.models

**Globals:**

models =

```
{'default' : 'medium.h5', 'tiny' : 'tiny.h5', 'medium' : 'medium.h5', 'large' :  
'large.h5'}
```

Dictionary of file names associated with differently sized models and their trained weights.

models\_dir = None

Absolute or relative path to the directory where the models are located.

\_model

Model used for prediction and training through this interface. Due to realistic memory limitations, only one module is allowed to be active at a time. This means if the interface functions below are called with different values for the “model\_name” parameter, there could be substantial loading time between calls while Keras processes the saved weights for that module.

**Interface:**

model\_predict

```
(ecg_data=numpy.array[N], freq=200, model_name='default', )
```

Converts the given ECG data (ecg\_data) into segments that can be ingested by the model, collects the predictions for all of the segments, averages the results, and returns the indices that are the center of the strongest diagnostic criteria the model identified for the top three conditions the data suggests.

Returns a tuple of the following data:

indices - A list of up to 10 indices in the given data that are the center of the the diagnostic criteria.

diagnostics - A list of tuples of the most probable conditions and the strength of each estimation.

load\_model(model\_name='default')

Loads the specified model.

save\_model(model\_name='default')

Saves the specified model to the file name associated with the model.

```
train_model(model=keras.Model,  
ecg_data=numpy.array[N][200][1],  
ecg_labs=numpy.array[N][1][classes],  
lr=0.001,  
decay=0.001,  
epochs=1,  
validation_rate=0.10)
```

Runs the Keras Model fit function using the given ECG data (ecg\_data), with the specified learning rate (lr), decay rate (decay), and number of (epochs). A percentage (validation\_rate) of the data elements are reserved for validation of the model.

This function returns several statistics from training as a tuple in the following order:



final\_accuracy - The accuracy when the data set is validated across the entire given data set.

final\_loss - The total loss across the entire given data set.

last\_val\_acc - The accuracy of the last validation run during training.

last\_val\_loss - The total loss of the last validation run during training.

## **DataIngest**

### **Module Description:**

A set of data conversion and interpretation algorithms to facilitate the using alternative data sets, user provided data, and other systems (such as a real-time monitor).

### **Full Python Module Path:**

heartconmon.dataingest

### **Globals:**

data\_path = None

datasources={'ptptb' : './physionet', 'mitph' : './mitph'}

### **Interface:**

segment\_data(ecg\_data=numpy.array[N\_1][N\_2], freq=200, samp\_per\_seg=200, sub\_samp=50)

Converts the provided data to data structured in a way the model can ingest and use for prediction. If two dimensional data is given, the first dimension will be used to create multiple records of segments.

get\_data\_source(name='ptptb', sub\_samp=25)

Returns the requested data set segmented with the requested sub-sampling interval.

## **WebService**

### **Module Description:**

A set of data conversion and interpretation algorithms to facilitate the using alternative data sets, user provided data, and other systems (such as a real-time monitor).

### **Full Python Module Path:**

heartconmon.webservice

### **Globals:**

None

### **Interface:**

start(model\_name='default', addr='0.0.0.0', port=80)

Starts the webservice with the specified configuration.

Only responds to any POST messages that provide comma-delimited floating point sequences and a frequency specified for the data (so various sources can be interpolated).

The comma-delimited sequence form element must be named: ecgdata  
The frequency (specified in Hertz) form element must be named: freq  
stop())  
Causes the webservice to terminate.

## Appendix C – Heartbeat API

### Account

#### Fields:

- **archive:** (Archive) archive created along with the user
- **device:** (Device) device paired to user at creation time.
- **session:** (Session) session subscribed to by the device
  - Used for quick reference to said session from client
- **email:** (string) email provided at creation time
- **password:** (string) salted password value
- **userId:** (string) generated uuid

#### **Mutations:**

- **createAccount**(email: string, password: string, deviceId: string): Account
  - Desc: Creates user in DB, Assigns provided device to user, creates archive for session storage, pairs device storage.
  - Requirements: None

#### Queries:

- **getAccount**(userId: string): Account
  - Desc: queries for Account fields based on uuid provided from creation
  - Example:
- **login**(email: string, password: string): Account | Null
  - Desc: Verifies the user credentials. Returns the user information if successful, error if not.

### Archive

#### Fields

- **archiveId:** (String) generated UUID of Archive
- **userId:** (String) id reference of the containing user
- **records :** ([Records]) List of record objects

#### Queries:

- **getArchive**(userId): Archive
  - Desc: returns the archive associated to the use
  - Requirements: User has been created

## Device

### Fields:

- deviceId: (String) serial number used to pair device
- deviceStatus: (String) "active | inactive | unknown"
- user: (Account) user object reference
- session: (Session) session object reference

### Mutations:

- **startDevice**(deviceId: String, userId: String): Device
  - Desc: Send a signal to initiate the device. Can be used to start data flow.
  - Requirements: Device is powered on/ has made previous connection with IoT core
- **stopDevice**(deviceId: String, userId: String): Device
  - Desc: Send a signal to stop the device. Can be used to stop data flow.
  - Requirements: Device is powered on/ has made previous connection with IoT core

### Queries:

- **getDevice**(deviceId: string): Device
  - Desc: returns fields requested of associated device
  - Requirements: Device has made a previous connection with IoT Core

## Record

### Fields:

- frequency: (int) frequency of readings
- recordId: (UUID) generated id
- beats[]:
  - value: ([float]) list of readings
  - conditions[]
    - name: (String) name of condition
    - confidence: (float) percent confidence in condition
- timestamp: (timestamp) generated timestamp (for potential queries)

### Methods:

- **recordUpdate**(deviceId: String, frequency: Int, beats[]): Record
  - Desc: Triggers an update to the record table. Should be triggered from a lambda & not the client.

- **Note:** This endpoint is used internally. It should not be accessed by the client. This acts as the trigger to update record subscriptions
- Requirements: Device has made initial connection to IoT core
- **subscribeToSession**(deviceId: String): Record
  - Desc: Used for live streaming data from the client. Updates are triggered when 'recordUpdate' is called.
  - **Note:** This will create an MQTT websocket connection with the API. Said connection is anticipating 'recordUpdate' to be called from other sources.
  - Requirements: Device has made initial connection to IoT core

## Session

### Fields:

- record: (Record) record reference
- sessionId: (UUID) generated uuid for reference
- deviceId: (String) reference to parent device
- userId: (String) reference to parent user

### Methods

- **getSession**(deviceId): Session
  - Desc: retrieve device session. Not intended for use by the client
  - Requirements: Device has made initial connection to IoT core

## Return Types

### Account

```
{
  "Account": {
    "email": { "S": "emailTest" },
    "userId": { "S": "1234" },
    "password": { "S": "somePassword" },
    "device": {...Device object },
    "session": {...Session object },
    "archive": {...Archive object }
  }
}
```

## Archive

```
{
  "Archive": {
    "archiveId": { "S": "1234" },
    "userId": { "S": "4567" },
    "records": [
      ... Record object,
      ... Record object,
      ...
    ]
  }
}
```

## Device

```
{
  "Device": {
    "deviceId": { "S": "12345" },
    "deviceStatus": { "S": "active | inactive | unknown" },
    "userId": { "S": "456" },
    "session": { ...Session object }
  }
}
```

## Record

```
{
  "Record": {
    "beats": [{
      "value": [0.12, 0.11, 0.13],
      "conditions": [
        {
          "name": "someCondition",
          "confidence": 0.1
        },
        ...
      ]
    }]
  }
}
```

## Session

```
{
  "Session": {
    "userId": { "S": "12345" },
    "deviceId": { "S": "12345" },
    "sessionId": { "S": "12345" },
    "record": {... Record Object}
  }
}
```

## Appendix D – GraphQL Schema

```
schema {  
  query: Query  
  mutation: Mutation  
  subscription: Subscription  
}  
  
type Account {  
  archive: Archive  
  device: Device  
  email: String  
  session: Session  
  userId: ID!  
}  
  
type Archive {  
  archiveId: ID!  
  beats: [Beat]  
  userId: String  
}  
  
type Beat {  
  conditions: [Condition]  
  start: Int  
  stop: Int  
  value: [Float]  
}  
  
type Condition {  
  conf: Float  
  name: String  
}  
  
type Device {  
  deviceId: ID!  
  deviceStatus: String  
  session: Session  
  userId: String  
}  
  
type Mutation {  
  addBeat(conditions: [ConditionInput], start: Int, stop: Int, value: [Float]): Beat
```



```

    createAccount(deviceId: String, email: String, password: String): Account
    getAccount(userId: String): Account
    getArchive(userId: String): Archive
    getDevice(deviceId: String): Device
    getSession(deviceId: String): Session
    login(email: String, password: String): Account
    sessionCreate(recordID: ID, sessionId: ID!, userId: ID): Session
    sessionDataUpdate(data: [Float], sessionId: String): Session
    startDevice(deviceId: String): Device
    stopDevice(deviceId: String): Device
}

# Query Container
type Query {
    getAccount(userId: ID): Account
    getDevice(deviceId: ID): Device
    getSession(userId: String): Session
}

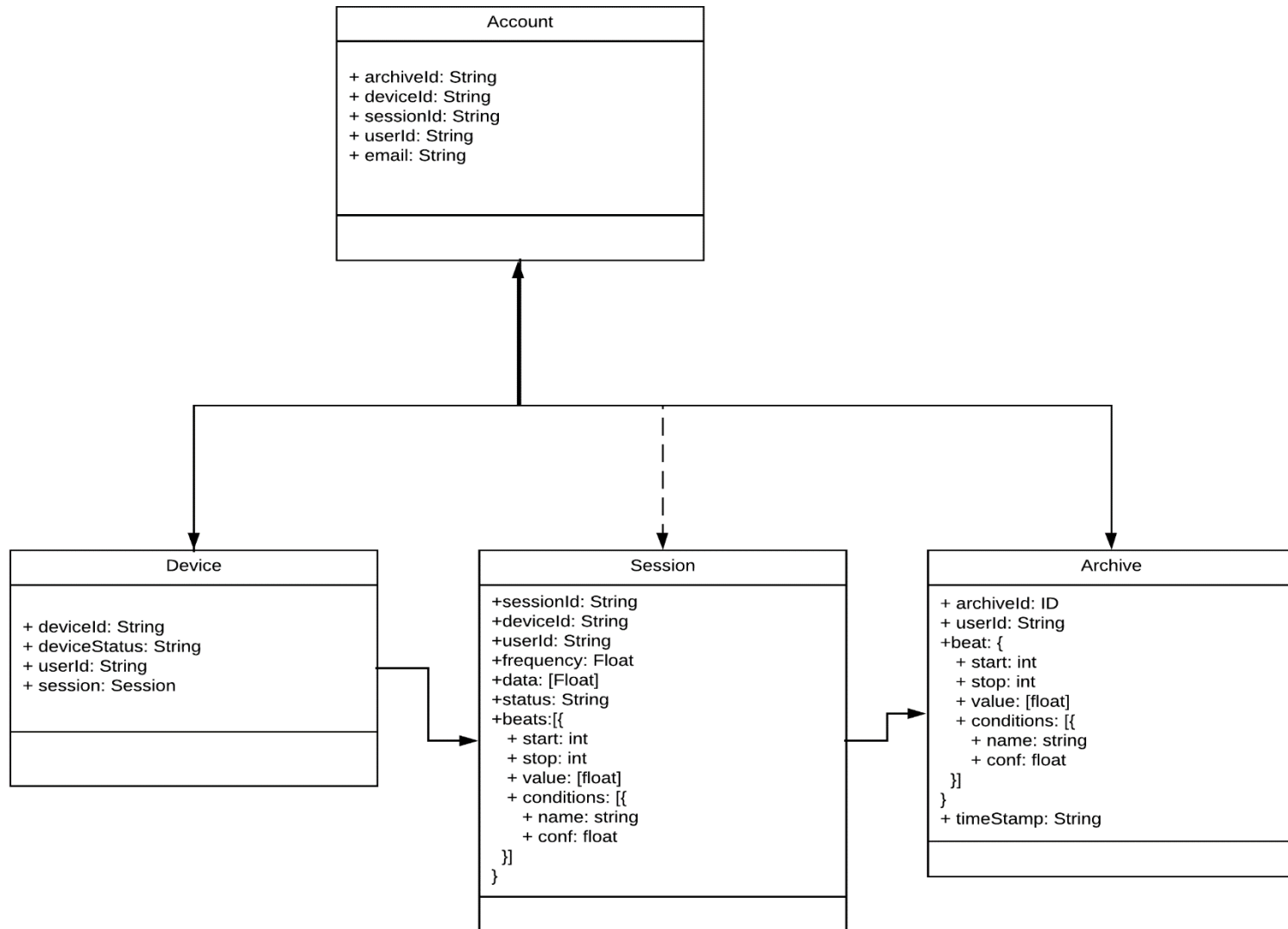
type Session {
    beats: [Beat]
    data: [Float]
    deviceId: String
    frequency: Float
    sessionId: String
    status: String
    userId: String
}

type Subscription {
    # Subscribe to a session via MQTT service provided by AWS-SDK
    ##### Events are created when a sessions record is updated
    subscribeToBeats(sessionId: ID!, userId: ID!): Beat @aws_subscribe(mutations :
    ["addBeat"])
    subscribeToSession(sessionId: ID): Session @aws_subscribe(mutations :
    ["sessionDataUpdate"])
}

input ConditionInput {
    conf: Float
    name: String
}

```

## Appendix E - DynamoDB Database Diagram



## References

- Centers for Medicare & Medicaid Services. (2016). *National Health Expenditures 2016 Highlights*. Retrieved from: <https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/NationalHealthExpendData/NationalHealthAccountsHistorical.html>
- Physionet. (N.d.). *The PTB diagnostic ECG database*. Retrieved from <https://physionet.org/physiobank/database/ptbdb/>
- Sparkfun. (N.d.). *AD8232 Heart rate monitor hookup guide*. Retrieved from <https://learn.sparkfun.com/tutorials/ad8232-heart-rate-monitor-hookup-guide/all>