Fit Factory Project Report

by Ethan Chen

The Device

The Device is made up of the Sparkfun Thing Dev, an accelerometer, and a particle sensor. The code below uses data from the accelerometer to figure out how many reps the user has completed of an exercise. It also analyzes the data from the particle sensor using rising and falling edges of the data to find out the users heart rate. The application used a modified Demo 4 in order to send data to the Azure Hub. It sends the average heart rate over the ten seconds as well as the reps completed.

The Following is the Arduino IDE code (ends on page 6, look for red message):

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <WiFiUdp.h>
#include <AzureIoTHub.h>
#include <AzureIoTProtocol_MQTT.h>
#include <AzureIoTUtility.h>
#include "config.h"

static bool messagePending = false;
static bool messageSending = true;
static char *connectionString;
static char *ssid;
static char *pass;
static int interval = INTERVAL;
#include "MAX30105.h"
#include "SparkFunLSM6DS3.h"
#include "SPI.h"
#include <Wire.h>
#include "heartRate.h"

MAX30105 particleSensor;
const byte RATE_SIZE = 4; //Increase this for more averaging. 4 is good.
byte rates[RATE_SIZE]; //Array of heart rates
byte rateSpot = 0;
long lastBeat = 0; //Time at which the last beat occurred

float beatsPerMinute;
int beatAvg;
```

```
LSM6DS3 myIMU; //Default constructor is I2C, addr 0x6B

float startY = 0;
float curY = 0;
float curPos = 0;
bool wentUp = false;
int step = 0;
int lastRead = 0;

void blinkLED()
{
  digitalWrite(LED_PIN, HIGH);
  delay(500);
  digitalWrite(LED_PIN, LOW);
}

void initWifi()
{
  // Attempt to connect to Wifi network:
  Serial.printf("Attempting to connect to SSID: %s.\r\n", ssid);

  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
    // Get Mac Address and show it.
    // WiFi.macAddress(mac) save the mac address into a six length array, but the endian may
be different. The huzzah board should
    // start from mac[0] to mac[5], but some other kinds of board run in the oppsite direction.
    uint8_t mac[6];
    WiFi.macAddress(mac);
    Serial.printf("You device with MAC address %02x:%02x:%02x:%02x:%02x:%02x connects
to %s failed! Waiting 10 seconds to retry.\r\n",
        mac[0], mac[1], mac[2], mac[3], mac[4], mac[5], ssid);
    WiFi.begin(ssid, pass);
    delay(10000);
  }
  Serial.printf("Connected to wifi %s.\r\n", ssid);
}

void initTime()
{
  time_t epochTime;
```

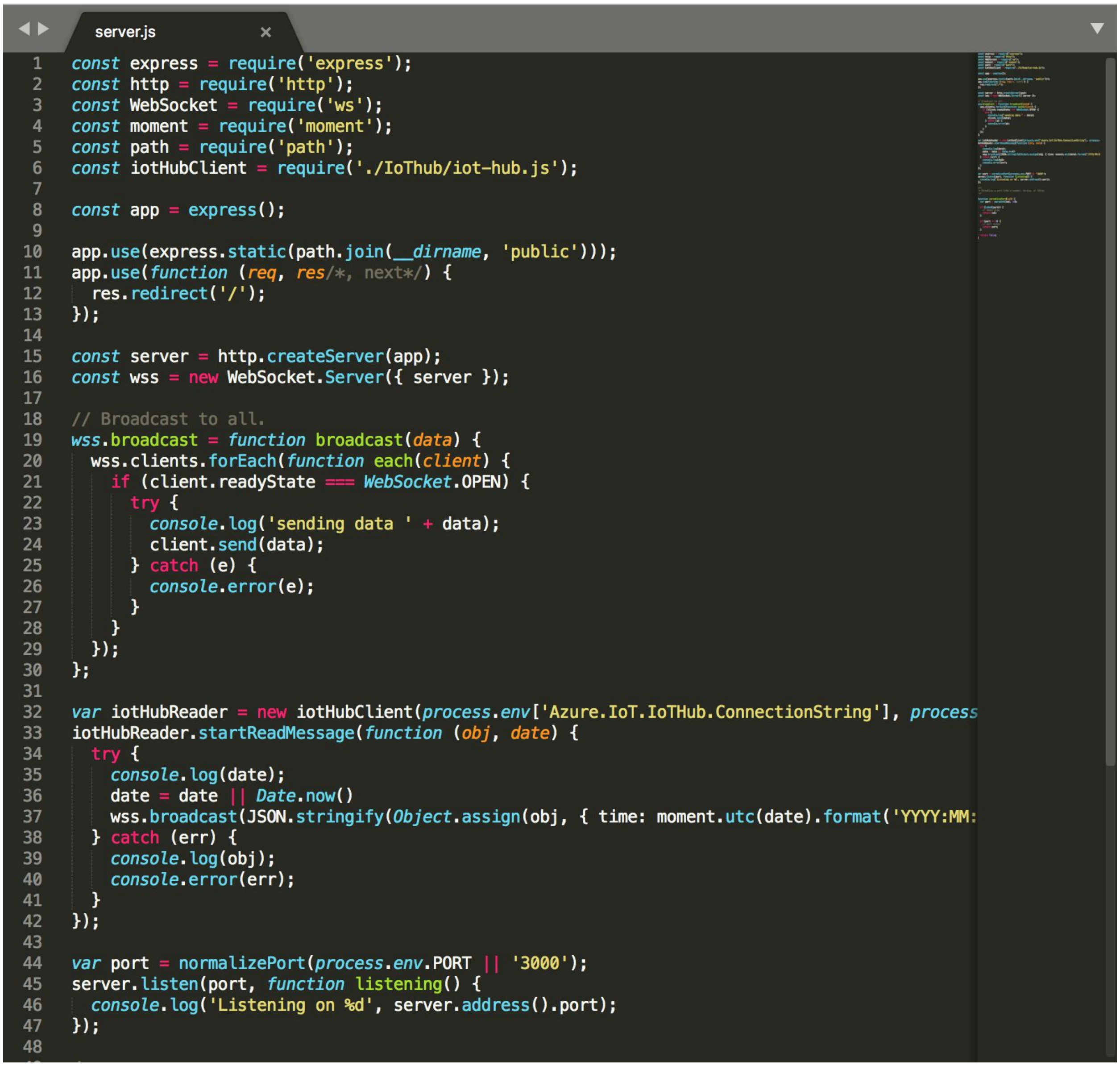
```
    configTime(0, 0, "pool.ntp.org", "time.nist.gov");

    while (true)
    {
        epochTime = time(NULL);

        if (epochTime == 0)
        {
            Serial.println("Fetching NTP epoch time failed! Waiting 2 seconds to retry.");
            delay(2000);
        }
        else
        {
            Serial.printf("Fetched NTP epoch time is: %lu.\r\n", epochTime);
            break;
        }
    }
}

//The callibration function records heartrate values until it finds 2 that are unique and above 60


void callibration(){
  beatsPerMinute = 0.0;
  float first = 0.0;
  float second = 0.0;
  bool done = false;
  while(!done){
  long irValue = particleSensor.getIR();

  if (checkForBeat(irValue) == true)
  {
    //We sensed a beat!
    long delta = millis() - lastBeat;
    lastBeat = millis();

    beatsPerMinute = 60 / (delta / 1000.0);

    if (beatsPerMinute < 255 && beatsPerMinute > 20)
    {
      rates[rateSpot++] = (byte)beatsPerMinute; //Store this reading in the array
      rateSpot %= RATE_SIZE; //Wrap variable
    }
    if(first < 60){
      first = beatsPerMinute;
    }else{
      second = beatsPerMinute;
    }
    if(first > 60 && second > 60 && first!=second){
      done = true;
    }
  /*
  if(beatsPerMinute>60)  {
    done = true;
  }
  */
  }

  Serial.print("IR=");
  Serial.print(irValue);
  Serial.print(", BPM=");
  Serial.print(beatsPerMinute);

  if (irValue < 50000)
    Serial.print(" No finger?");

  Serial.println();
}
}

static IOTHUB_CLIENT_LL_HANDLE iotHubClientHandle;
void setup()
{
    pinMode(LED_PIN, OUTPUT);

    initSerial();
    delay(2000);
    readCredentials();

    initWifi();
    initTime();
    initSensor();
    iotHubClientHandle = IoTHubClient_LL_CreateFromConnectionString(connectionString,
MQTT_Protocol);
    if (iotHubClientHandle == NULL)
    {
        Serial.println("Failed on IoTHubClient_CreateFromConnectionString.");
        while (1);
    }

    IoTHubClient_LL_SetMessageCallback(iotHubClientHandle, receiveMessageCallback,
NULL);
    IoTHubClient_LL_SetDeviceMethodCallback(iotHubClientHandle, deviceMethodCallback,
NULL);
    IoTHubClient_LL_SetDeviceTwinCallback(iotHubClientHandle, twinCallback, NULL);

    myIMU.begin();
    startY = myIMU.readFloatAccelY()*50+5;


  Serial.println("Callibrating:");
  callibration();
  Serial.println("Done Callibrating");


}

static int messageCount = 1;
long timer= millis();
  //while(millis()<time + 10000){

void loop()
{
//modified a sparkfun heart rate example to calculate the heart rate using the MAX30105 with
the following algorithm
  long irValue = particleSensor.getIR();

  if (checkForBeat(irValue) == true)
    {
      long delta = millis() - lastBeat;
      lastBeat = millis();
      beatsPerMinute = 60 / (delta / 1000.0);

      if (beatsPerMinute < 255 && beatsPerMinute > 20)
      {
```

```
    rates[rateSpot++] = (byte)beatsPerMinute; //Store this reading in the array
    rateSpot %= RATE_SIZE; //Wrap variable

   }
  }

//In order to track user movement, I used the following formula to record movements.
  curY = myIMU.readFloatAccelY()*50+5;

 if(millis()-lastRead > 500){
  if(abs(curY) > 10 + startY){
   if(!wentUp){
    wentUp=true;
    //step++;
   }
  }
  if(abs(curY) < 3 + startY){
   if(wentUp){
    wentUp = false;
    step++;
    lastRead = millis();
   }
  }
 }
//  I modified the message payload to only update every 10 seconds.
  if (!messagePending && messageSending&& millis()>timer+10000)
  {
    char messagePayload[MESSAGE_MAX_LEN];

// I modified the temperatureAlert function to also include the beats per minute and steps
calculated in the 10 second period
    bool temperatureAlert = readMessage(messageCount,
messagePayload,beatsPerMinute,step);
    sendMessage(iotHubClientHandle, messagePayload, temperatureAlert);
    messageCount++;
    step = 0;
    //delay(interval);
    timer = millis();
  }
  IoTHubClient_LL_DoWork(iotHubClientHandle);
  delay(10);
}
```

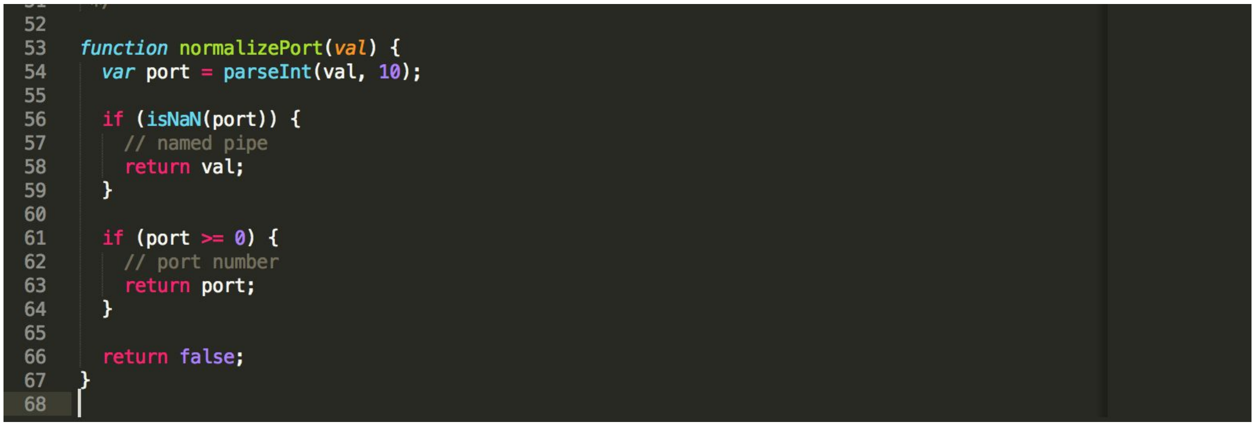
The End of Arduino IDE code!

The Cloud

The app utilizes the Azure IoT Hub and Web Application services. The Sparkfun thing takes data readings and sends information to the IoT Hub. The IoT Hub has endpoints that allow the web application to access this data. I used the code from Demo4 for the server side of the app. The Hub basically serves as a middleman between the application and the device and makes the communication smooth and simple. The server reads data from the hub and broadcasts it to all of the clients.



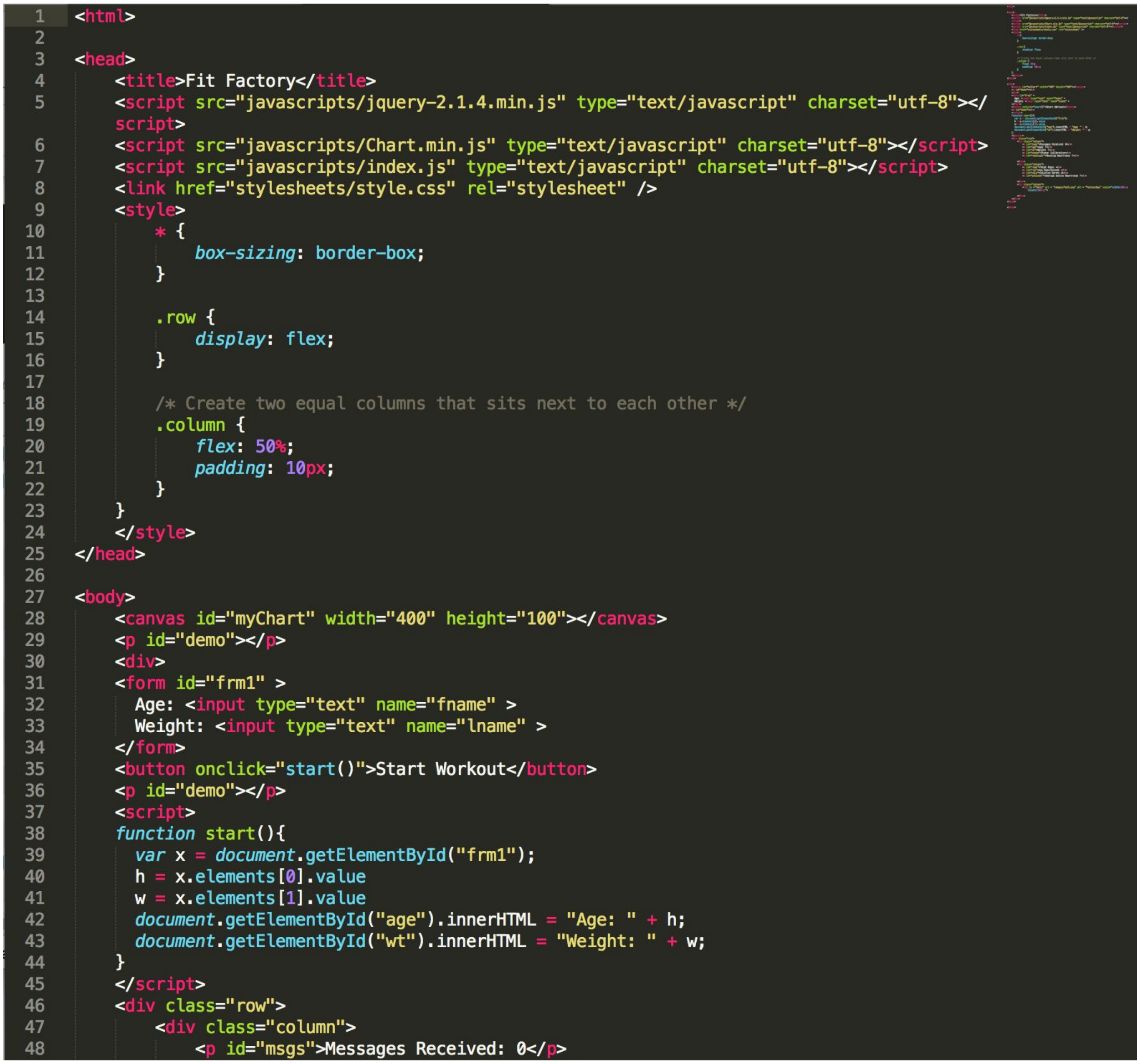
```
const express = require('express');
const http = require('http');
const WebSocket = require('ws');
const moment = require('moment');
const path = require('path');
const iotHubClient = require('./IoThub/iot-hub.js');

const app = express();

app.use(express.static(path.join(__dirname, 'public')));
app.use(function (req, res/*, next*/) {
  res.redirect('/');
});

const server = http.createServer(app);
const wss = new WebSocket.Server({ server });

// Broadcast to all.
wss.broadcast = function broadcast(data) {
  wss.clients.forEach(function each(client) {
    if (client.readyState === WebSocket.OPEN) {
      try {
        console.log('sending data ' + data);
        client.send(data);
      } catch (e) {
        console.error(e);
      }
    }
  });
};

var iotHubReader = new iotHubClient(process.env['Azure.IoT.IoTHub.ConnectionString'], process
iotHubReader.startReadMessage(function (obj, date) {
  try {
    console.log(date);
    date = date || Date.now()
    wss.broadcast(JSON.stringify(Object.assign(obj, { time: moment.utc(date).format('YYYY:MM:
  } catch (err) {
    console.log(obj);
    console.error(err);
  }
});

var port = normalizePort(process.env.PORT || '3000');
server.listen(port, function listening() {
  console.log('Listening on %d', server.address().port);
});

```



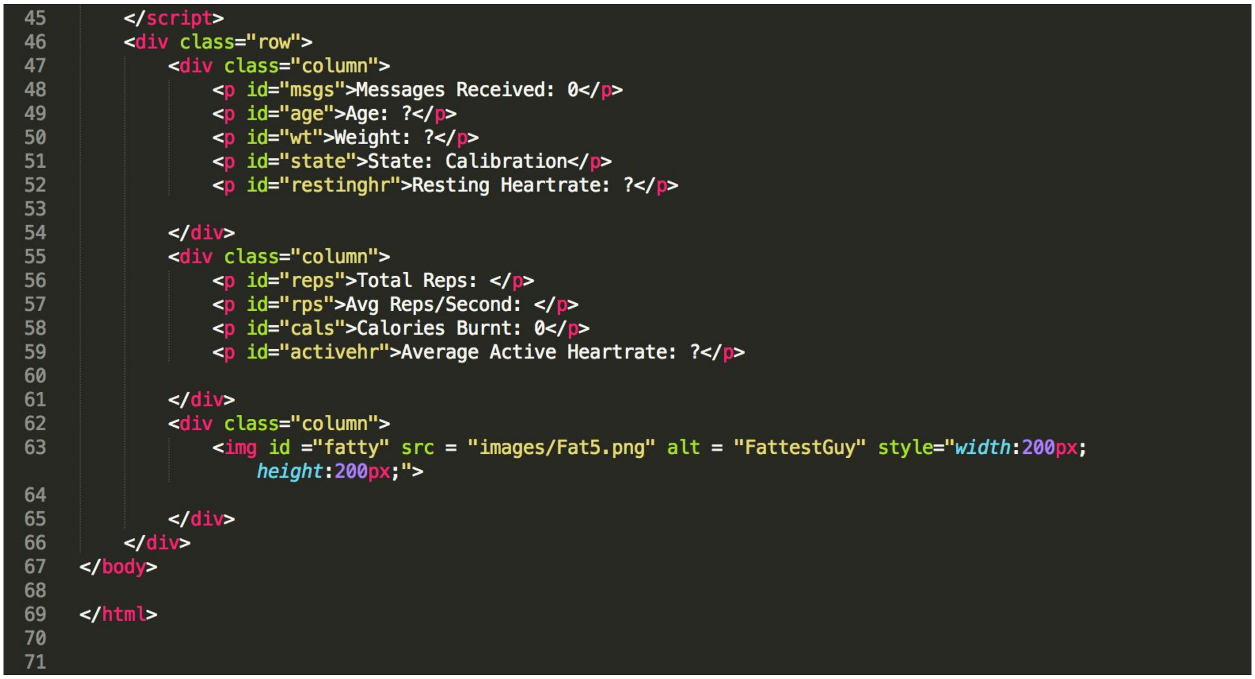
```

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

```

The Application

The HTML document formats the information and creates some simple functionality to the page like inputs and buttons. It also features a picture of a fat person to start that gets thinner as a user gets closer to the goal. This gives a simple way to visualize the workout progress.

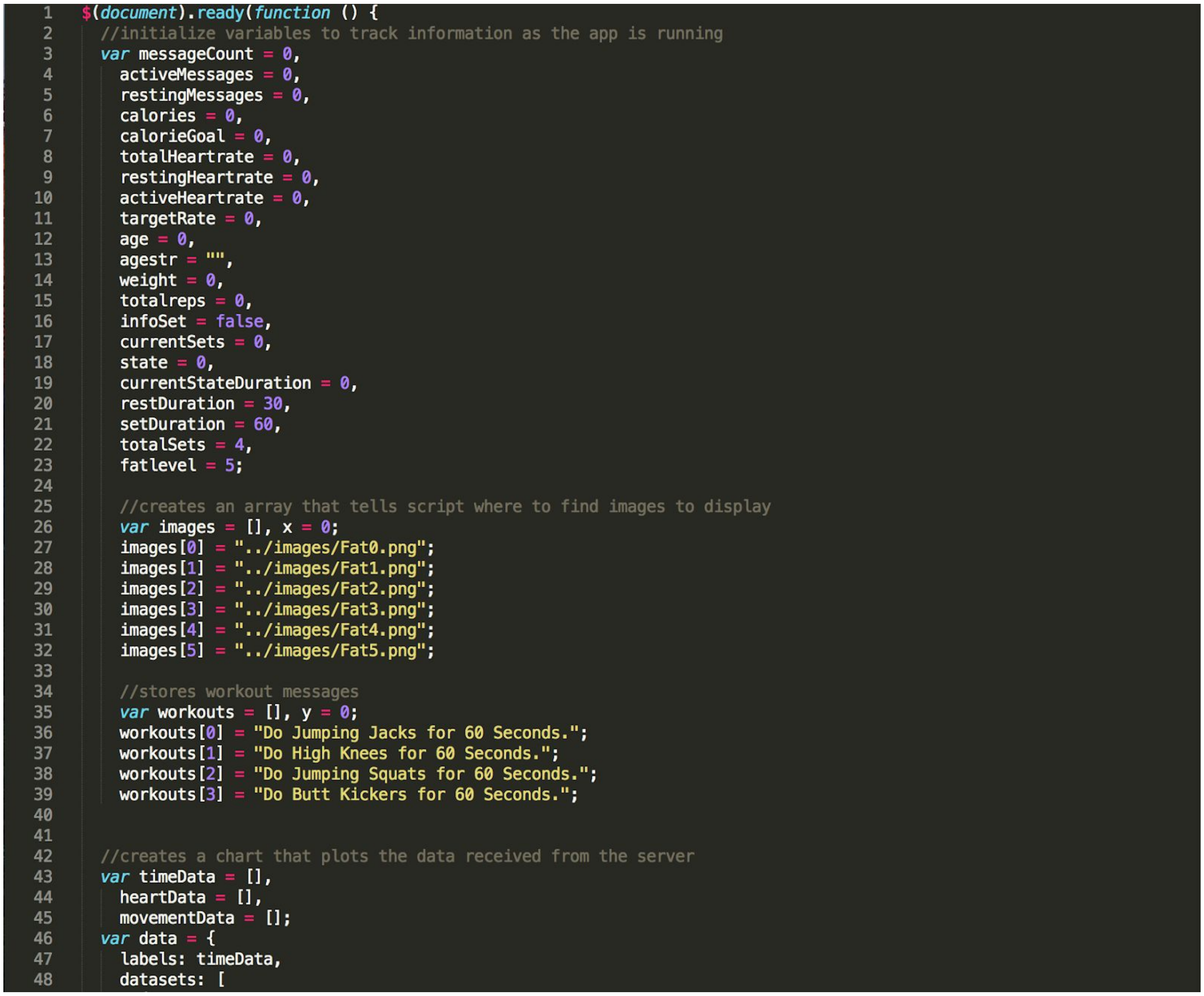


```
<html>

<head>
    <title>Fit Factory</title>
    <script src="javascripts/jquery-2.1.4.min.js" type="text/javascript" charset="utf-8"></script>
    <script src="javascripts/Chart.min.js" type="text/javascript" charset="utf-8"></script>
    <script src="javascripts/index.js" type="text/javascript" charset="utf-8"></script>
    <link href="stylesheets/style.css" rel="stylesheet" />
    <style>
        * {
            box-sizing: border-box;
        }

        .row {
            display: flex;
        }

        /* Create two equal columns that sits next to each other */
        .column {
            flex: 50%;
            padding: 10px;
        }
    }
    </style>
</head>

<body>
    <canvas id="myChart" width="400" height="100"></canvas>
    <p id="demo"></p>
    <div>
    <form id="frm1" >
      Age: <input type="text" name="fname" >
      Weight: <input type="text" name="lname" >
    </form>
    <button onclick="start()">Start Workout</button>
    <p id="demo"></p>
    <script>
    function start(){
      var x = document.getElementById("frm1");
      h = x.elements[0].value
      w = x.elements[1].value
      document.getElementById("age").innerHTML = "Age: " + h;
      document.getElementById("wt").innerHTML = "Weight: " + w;
    }
    </script>
    <div class="row">
        <div class="column">
            <p id="msgs">Messages Received: 0</p>
```

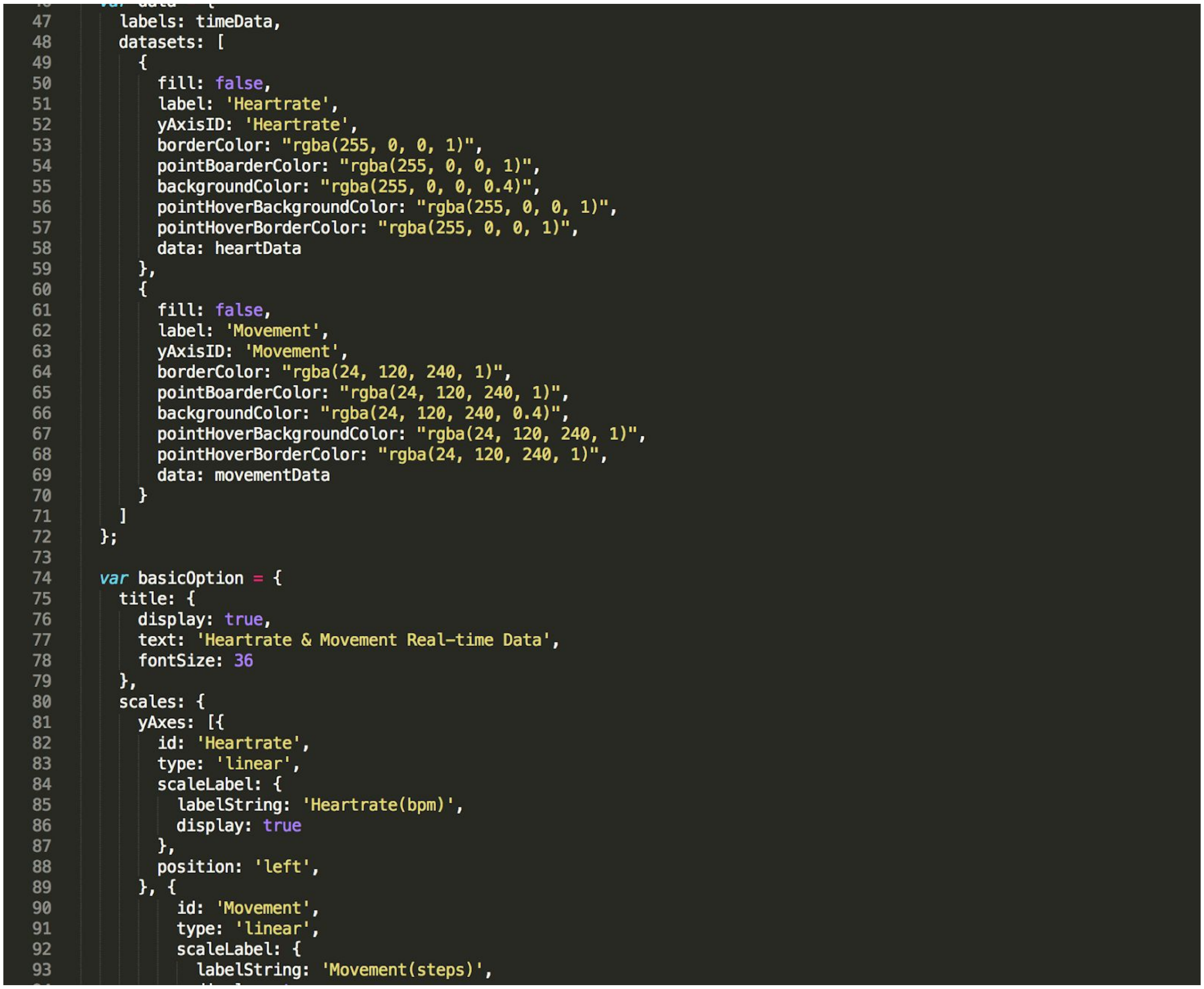


```html
    </script>
    <div class="row">
        <div class="column">
            <p id="msgs">Messages Received: 0</p>
            <p id="age">Age: ?</p>
            <p id="wt">Weight: ?</p>
            <p id="state">State: Calibration</p>
            <p id="restinghr">Resting Heartrate: ?</p>

        </div>
        <div class="column">
            <p id="reps">Total Reps: </p>
            <p id="rps">Avg Reps/Second: </p>
            <p id="cals">Calories Burnt: 0</p>
            <p id="activehr">Average Active Heartrate: ?</p>

        </div>
        <div class="column">
            <img id ="fatty" src = "images/Fat5.png" alt = "FattestGuy" style="width:200px; height:200px;">

        </div>
    </div>
</body>

</html>


```

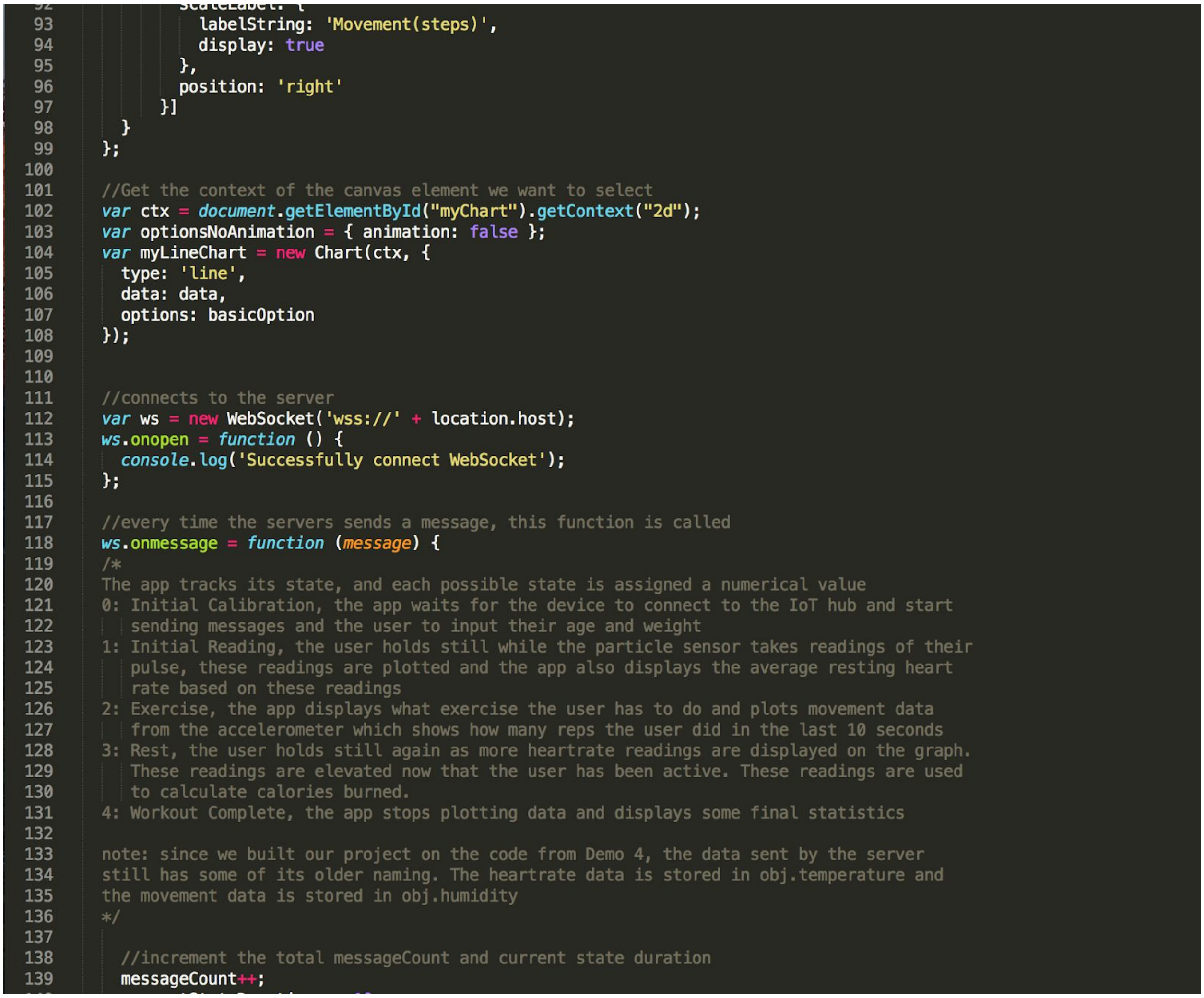
The Javascript code is where most of the calculations are performed. Most of the code is executed every time a message is received from the server. It shifts between various states and the display changes accordingly. In the screenshotted code below, there are comments that discuss this in more detail. The key idea here was to keep updating the information so that the user can have a live feed of data during the workout.



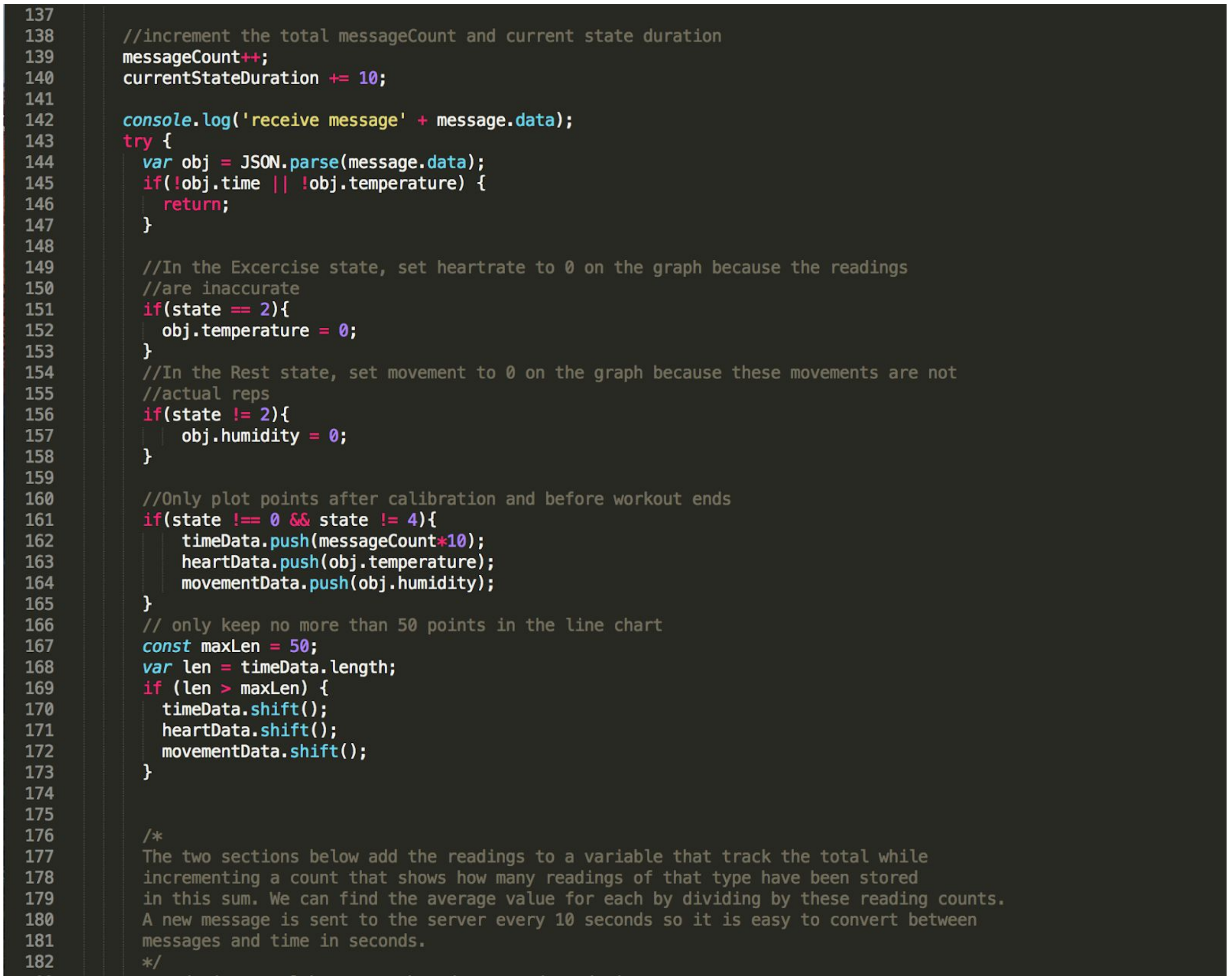
```
$(document).ready(function () {
  //initialize variables to track information as the app is running
  var messageCount = 0,
    activeMessages = 0,
    restingMessages = 0,
    calories = 0,
    calorieGoal = 0,
    totalHeartrate = 0,
    restingHeartrate = 0,
    activeHeartrate = 0,
    targetRate = 0,
    age = 0,
    agestr = "",
    weight = 0,
    totalreps = 0,
    infoSet = false,
    currentSets = 0,
    state = 0,
    currentStateDuration = 0,
    restDuration = 30,
    setDuration = 60,
    totalSets = 4,
    fatlevel = 5;

    //creates an array that tells script where to find images to display
    var images = [], x = 0;
    images[0] = "../images/Fat0.png";
    images[1] = "../images/Fat1.png";
    images[2] = "../images/Fat2.png";
    images[3] = "../images/Fat3.png";
    images[4] = "../images/Fat4.png";
    images[5] = "../images/Fat5.png";

    //stores workout messages
    var workouts = [], y = 0;
    workouts[0] = "Do Jumping Jacks for 60 Seconds.";
    workouts[1] = "Do High Knees for 60 Seconds.";
    workouts[2] = "Do Jumping Squats for 60 Seconds.";
    workouts[3] = "Do Butt Kickers for 60 Seconds.";


  //creates a chart that plots the data received from the server
  var timeData = [],
    heartData = [],
    movementData = [];
  var data = {
    labels: timeData,
    datasets: [
```



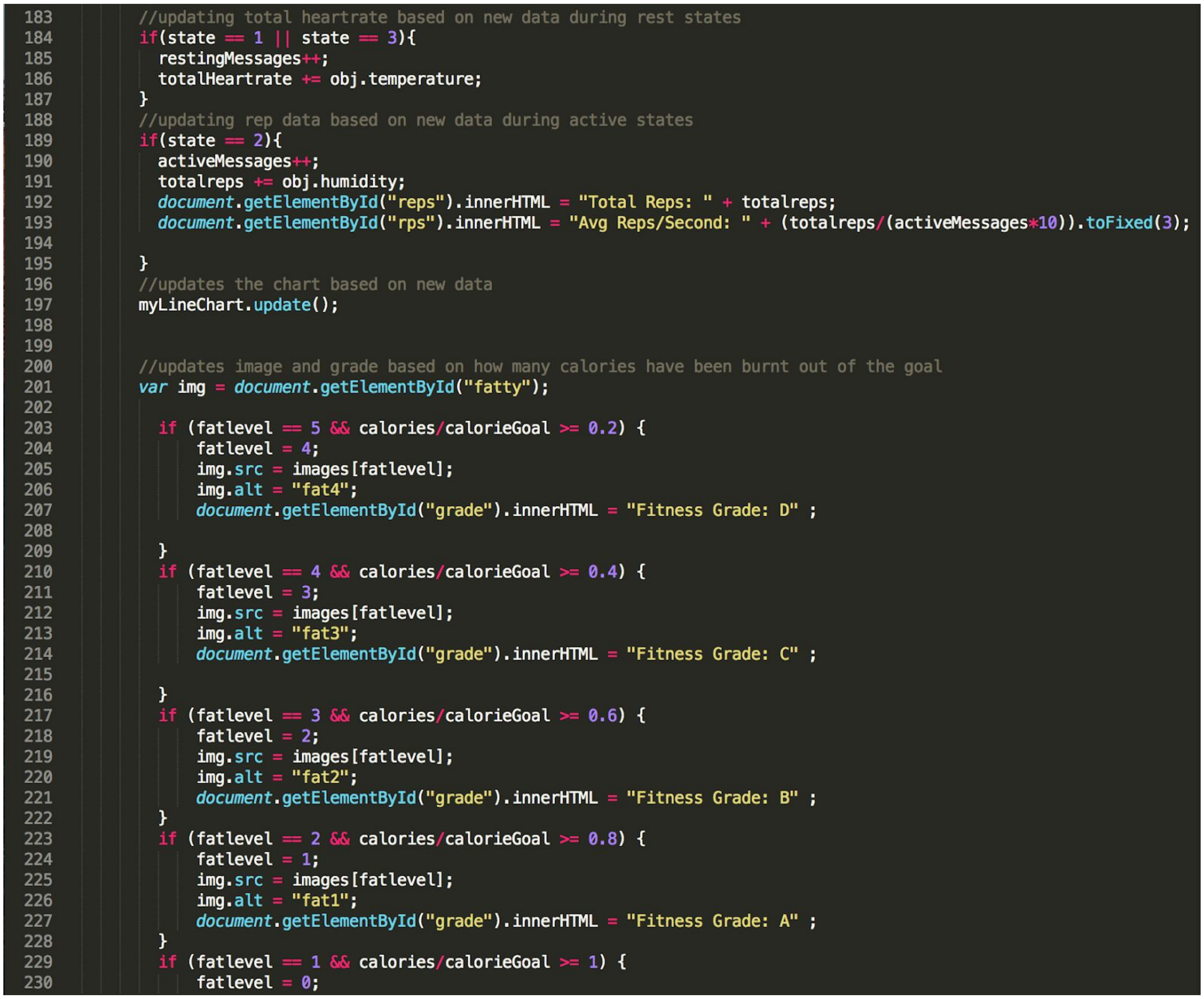
```
var data = {
  labels: timeData,
  datasets: [
    {
      fill: false,
      label: 'Heartrate',
      yAxisID: 'Heartrate',
      borderColor: "rgba(255, 0, 0, 1)",
      pointBoarderColor: "rgba(255, 0, 0, 1)",
      backgroundColor: "rgba(255, 0, 0, 0.4)",
      pointHoverBackgroundColor: "rgba(255, 0, 0, 1)",
      pointHoverBorderColor: "rgba(255, 0, 0, 1)",
      data: heartData
    },
    {
      fill: false,
      label: 'Movement',
      yAxisID: 'Movement',
      borderColor: "rgba(24, 120, 240, 1)",
      pointBoarderColor: "rgba(24, 120, 240, 1)",
      backgroundColor: "rgba(24, 120, 240, 0.4)",
      pointHoverBackgroundColor: "rgba(24, 120, 240, 1)",
      pointHoverBorderColor: "rgba(24, 120, 240, 1)",
      data: movementData
    }
  ]
};

var basicOption = {
  title: {
    display: true,
    text: 'Heartrate & Movement Real-time Data',
    fontSize: 36
  },
  scales: {
    yAxes: [{
      id: 'Heartrate',
      type: 'linear',
      scaleLabel: {
        labelString: 'Heartrate(bpm)',
        display: true
      },
      position: 'left',
    }, {
        id: 'Movement',
        type: 'linear',
        scaleLabel: {
          labelString: 'Movement(steps)',
```



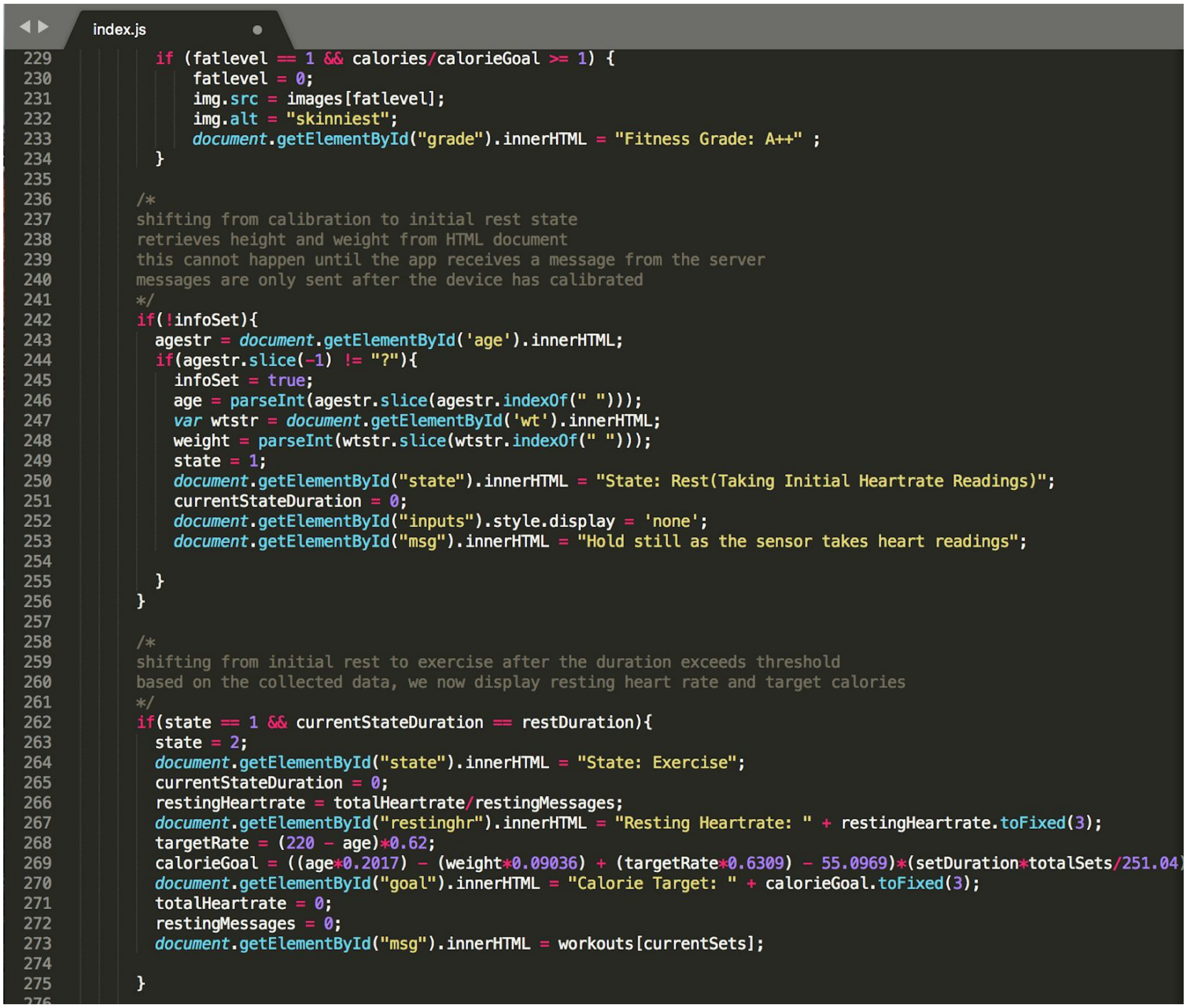
```
        scaleLabel: {
          labelString: 'Movement(steps)',
          display: true
        },
        position: 'right'
      }]
  }
};

//Get the context of the canvas element we want to select
var ctx = document.getElementById("myChart").getContext("2d");
var optionsNoAnimation = { animation: false };
var myLineChart = new Chart(ctx, {
  type: 'line',
  data: data,
  options: basicOption
});


//connects to the server
var ws = new WebSocket('wss://' + location.host);
ws.onopen = function () {
  console.log('Successfully connect WebSocket');
};

//every time the servers sends a message, this function is called
ws.onmessage = function (message) {
/*
The app tracks its state, and each possible state is assigned a numerical value
0: Initial Calibration, the app waits for the device to connect to the IoT hub and start
   sending messages and the user to input their age and weight
1: Initial Reading, the user holds still while the particle sensor takes readings of their
   pulse, these readings are plotted and the app also displays the average resting heart
   rate based on these readings
2: Exercise, the app displays what exercise the user has to do and plots movement data
   from the accelerometer which shows how many reps the user did in the last 10 seconds
3: Rest, the user holds still again as more heartrate readings are displayed on the graph.
   These readings are elevated now that the user has been active. These readings are used
   to calculate calories burned.
4: Workout Complete, the app stops plotting data and displays some final statistics

note: since we built our project on the code from Demo 4, the data sent by the server
still has some of its older naming. The heartrate data is stored in obj.temperature and
the movement data is stored in obj.humidity
*/

  //increment the total messageCount and current state duration
  messageCount++;
```



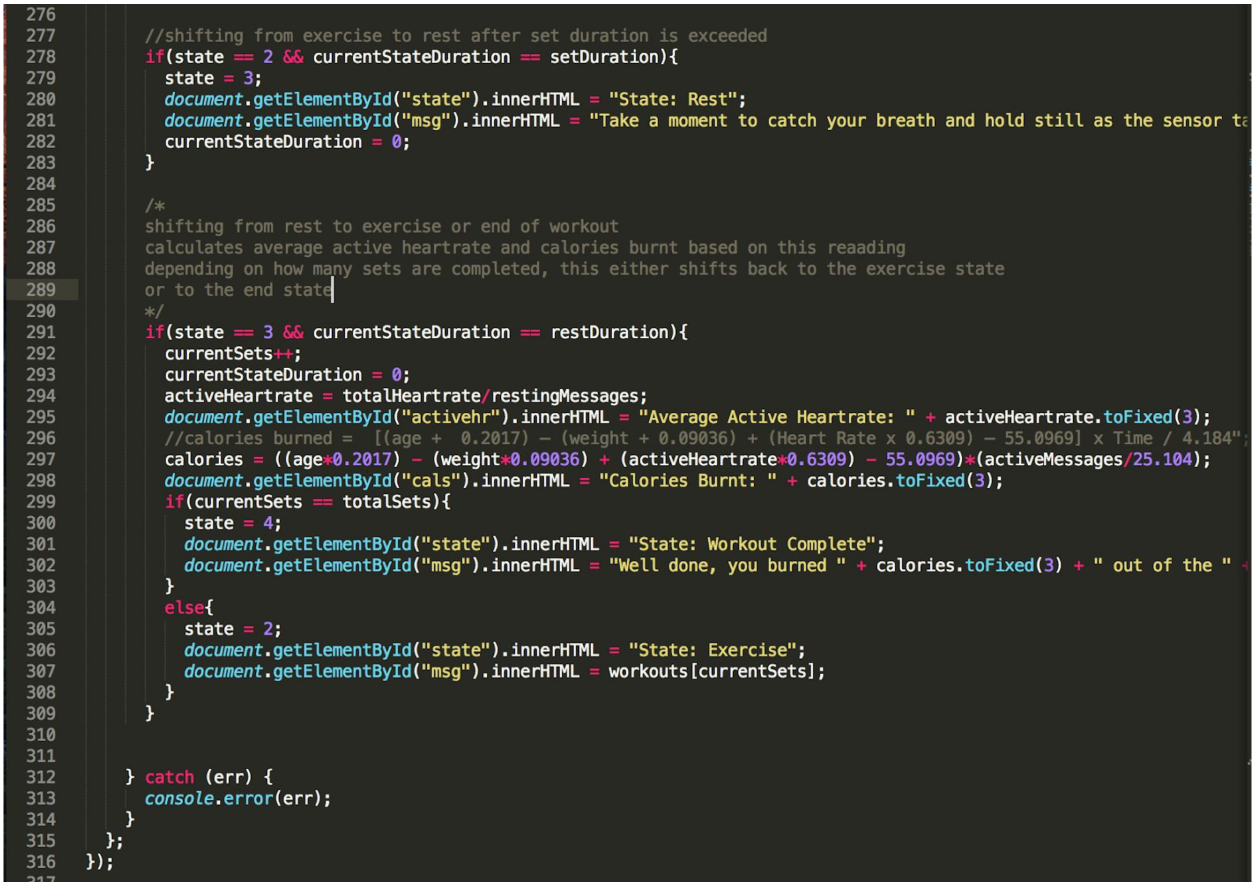
```

//increment the total messageCount and current state duration
messageCount++;
currentStateDuration += 10;

console.log('receive message' + message.data);
try {
  var obj = JSON.parse(message.data);
  if(!obj.time || !obj.temperature) {
    return;
  }

  //In the Excercise state, set heartrate to 0 on the graph because the readings
  //are inaccurate
  if(state == 2){
    obj.temperature = 0;
  }
  //In the Rest state, set movement to 0 on the graph because these movements are not
  //actual reps
  if(state != 2){
      obj.humidity = 0;
  }

  //Only plot points after calibration and before workout ends
  if(state !== 0 && state != 4){
      timeData.push(messageCount*10);
      heartData.push(obj.temperature);
      movementData.push(obj.humidity);
  }
  // only keep no more than 50 points in the line chart
  const maxLen = 50;
  var len = timeData.length;
  if (len > maxLen) {
    timeData.shift();
    heartData.shift();
    movementData.shift();
  }


  /*
  The two sections below add the readings to a variable that track the total while
  incrementing a count that shows how many readings of that type have been stored
  in this sum. We can find the average value for each by dividing by these reading counts.
  A new message is sent to the server every 10 seconds so it is easy to convert between
  messages and time in seconds.
  */
```



```javascript
      //updating total heartrate based on new data during rest states
      if(state == 1 || state == 3){
        restingMessages++;
        totalHeartrate += obj.temperature;
      }
      //updating rep data based on new data during active states
      if(state == 2){
        activeMessages++;
        totalreps += obj.humidity;
        document.getElementById("reps").innerHTML = "Total Reps: " + totalreps;
        document.getElementById("rps").innerHTML = "Avg Reps/Second: " + (totalreps/(activeMessages*10)).toFixed(3);

      }
      //updates the chart based on new data
      myLineChart.update();


      //updates image and grade based on how many calories have been burnt out of the goal
      var img = document.getElementById("fatty");

        if (fatlevel == 5 && calories/calorieGoal >= 0.2) {
            fatlevel = 4;
            img.src = images[fatlevel];
            img.alt = "fat4";
            document.getElementById("grade").innerHTML = "Fitness Grade: D" ;

        }
        if (fatlevel == 4 && calories/calorieGoal >= 0.4) {
            fatlevel = 3;
            img.src = images[fatlevel];
            img.alt = "fat3";
            document.getElementById("grade").innerHTML = "Fitness Grade: C" ;

        }
        if (fatlevel == 3 && calories/calorieGoal >= 0.6) {
            fatlevel = 2;
            img.src = images[fatlevel];
            img.alt = "fat2";
            document.getElementById("grade").innerHTML = "Fitness Grade: B" ;
        }
        if (fatlevel == 2 && calories/calorieGoal >= 0.8) {
            fatlevel = 1;
            img.src = images[fatlevel];
            img.alt = "fat1";
            document.getElementById("grade").innerHTML = "Fitness Grade: A" ;
        }
        if (fatlevel == 1 && calories/calorieGoal >= 1) {
            fatlevel = 0;
```

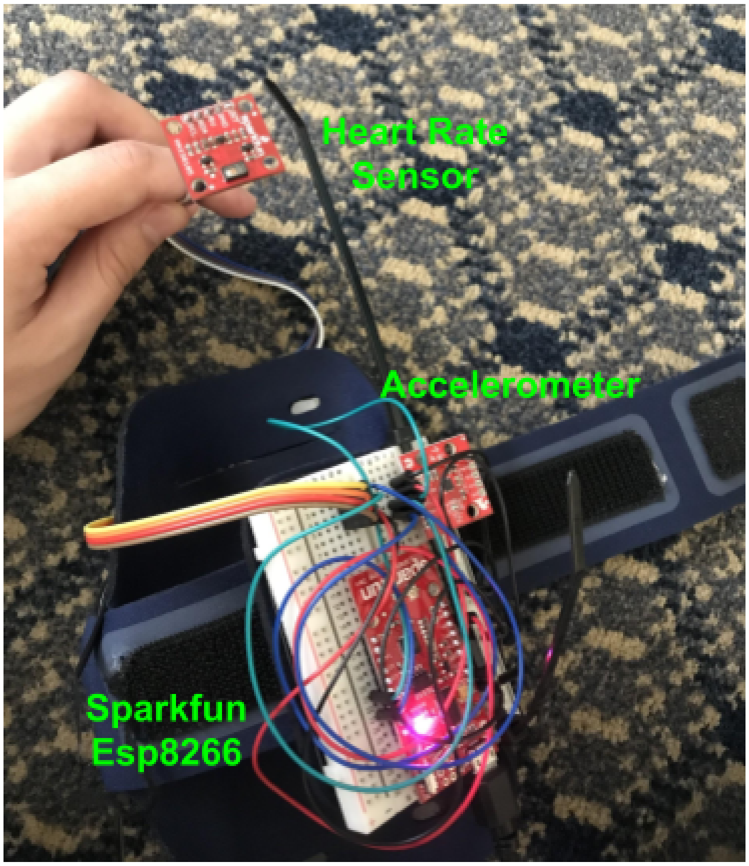


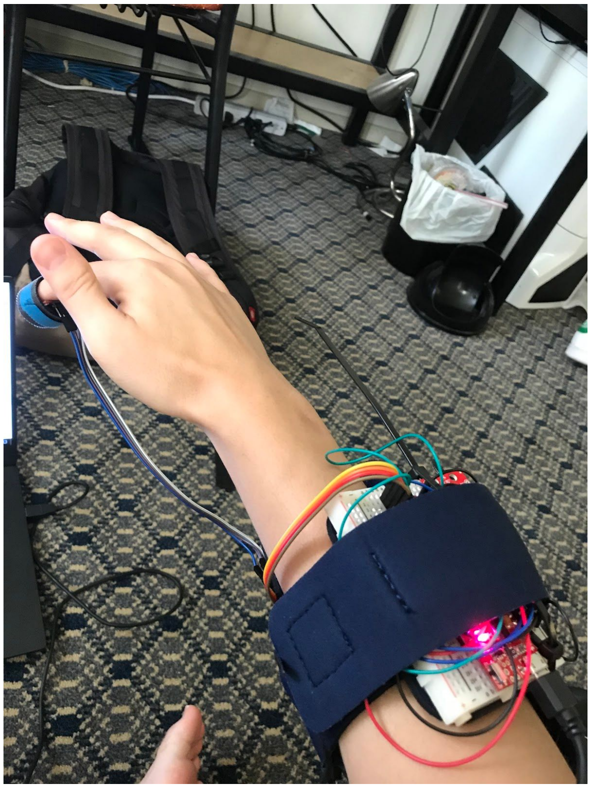
```
  if (fatlevel == 1 && calories/calorieGoal >= 1) {
      fatlevel = 0;
      img.src = images[fatlevel];
      img.alt = "skinniest";
      document.getElementById("grade").innerHTML = "Fitness Grade: A++" ;
  }

/*
shifting from calibration to initial rest state
retrieves height and weight from HTML document
this cannot happen until the app receives a message from the server
messages are only sent after the device has calibrated
*/
if(!infoSet){
  agestr = document.getElementById('age').innerHTML;
  if(agestr.slice(-1) != "?"){
    infoSet = true;
    age = parseInt(agestr.slice(agestr.indexOf(" ")));
    var wtstr = document.getElementById('wt').innerHTML;
    weight = parseInt(wtstr.slice(wtstr.indexOf(" ")));
    state = 1;
    document.getElementById("state").innerHTML = "State: Rest(Taking Initial Heartrate Readings)";
    currentStateDuration = 0;
    document.getElementById("inputs").style.display = 'none';
    document.getElementById("msg").innerHTML = "Hold still as the sensor takes heart readings";

  }
}

/*
shifting from initial rest to exercise after the duration exceeds threshold
based on the collected data, we now display resting heart rate and target calories
*/
if(state == 1 && currentStateDuration == restDuration){
  state = 2;
  document.getElementById("state").innerHTML = "State: Exercise";
  currentStateDuration = 0;
  restingHeartrate = totalHeartrate/restingMessages;
  document.getElementById("restinghr").innerHTML = "Resting Heartrate: " + restingHeartrate.toFixed(3);
  targetRate = (220 - age)*0.62;
  calorieGoal = ((age*0.2017) - (weight*0.09036) + (targetRate*0.6309) - 55.0969)*(setDuration*totalSets/251.04)
  document.getElementById("goal").innerHTML = "Calorie Target: " + calorieGoal.toFixed(3);
  totalHeartrate = 0;
  restingMessages = 0;
  document.getElementById("msg").innerHTML = workouts[currentSets];

}
```



```
      //shifting from exercise to rest after set duration is exceeded
      if(state == 2 && currentStateDuration == setDuration){
        state = 3;
        document.getElementById("state").innerHTML = "State: Rest";
        document.getElementById("msg").innerHTML = "Take a moment to catch your breath and hold still as the sensor ta
        currentStateDuration = 0;
      }

      /*
      shifting from rest to exercise or end of workout
      calculates average active heartrate and calories burnt based on this reaading
      depending on how many sets are completed, this either shifts back to the exercise state
      or to the end state
      */
      if(state == 3 && currentStateDuration == restDuration){
        currentSets++;
        currentStateDuration = 0;
        activeHeartrate = totalHeartrate/restingMessages;
        document.getElementById("activehr").innerHTML = "Average Active Heartrate: " + activeHeartrate.toFixed(3);
        //calories burned =  [(age +  0.2017) — (weight + 0.09036) + (Heart Rate x 0.6309) — 55.0969] x Time / 4.184";
        calories = ((age*0.2017) - (weight*0.09036) + (activeHeartrate*0.6309) - 55.0969)*(activeMessages/25.104);
        document.getElementById("cals").innerHTML = "Calories Burnt: " + calories.toFixed(3);
        if(currentSets == totalSets){
          state = 4;
          document.getElementById("state").innerHTML = "State: Workout Complete";
          document.getElementById("msg").innerHTML = "Well done, you burned " + calories.toFixed(3) + " out of the "
        }
        else{
          state = 2;
          document.getElementById("state").innerHTML = "State: Exercise";
          document.getElementById("msg").innerHTML = workouts[currentSets];
        }
      }


    } catch (err) {
      console.error(err);
    }
  };
});
```

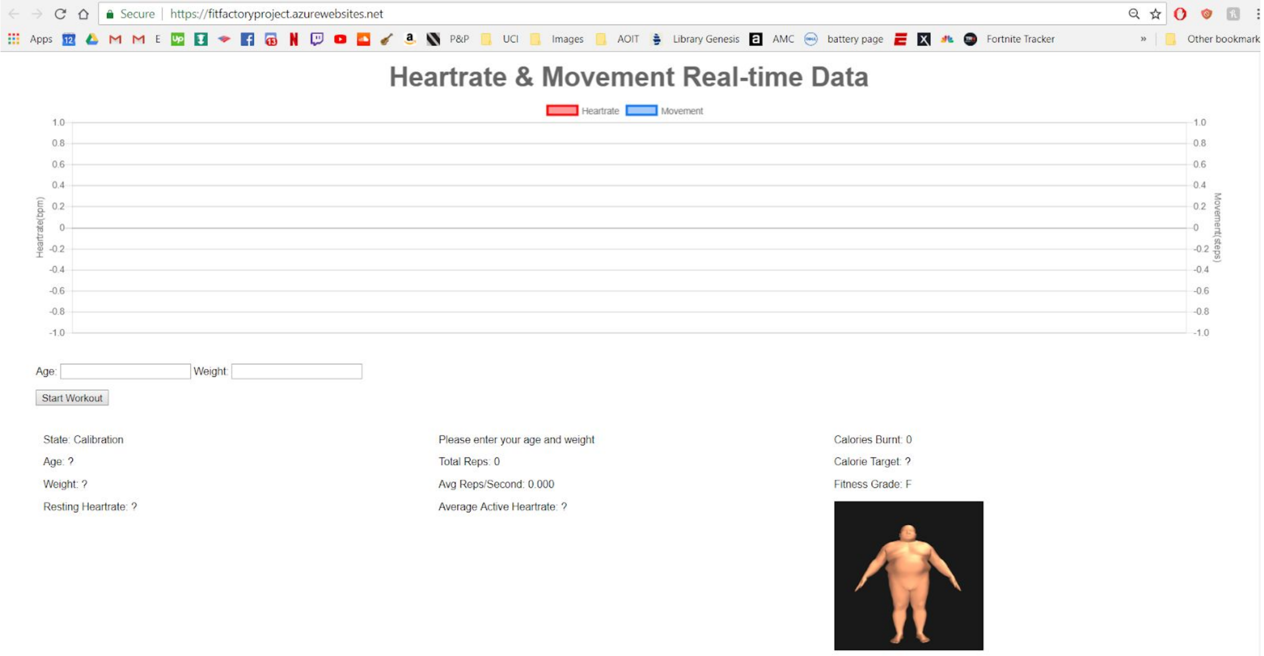
This Web Application, named Fit Factory, is a workout analysis application that uses Sparkfun ESP8266 board connected to a MAX30105 particle sensor and an LSM6DS3 accelerometer to track the user's heart rate and movements.





I configured a standard workout phone sleeve to hold the sparkfun and the connected sensors to fit on my arm while performing the workout.

The Web Application begins with this empty graph (which later plots the user's heart rate and actions performed during 10 second intervals), and information about the workout below. I also include an image of a fat orange human figure to represent the starting place for the workout. The goal was to have this fat figure get skinnier as the workout progressed.



Heartrate & Movement Real-time Data

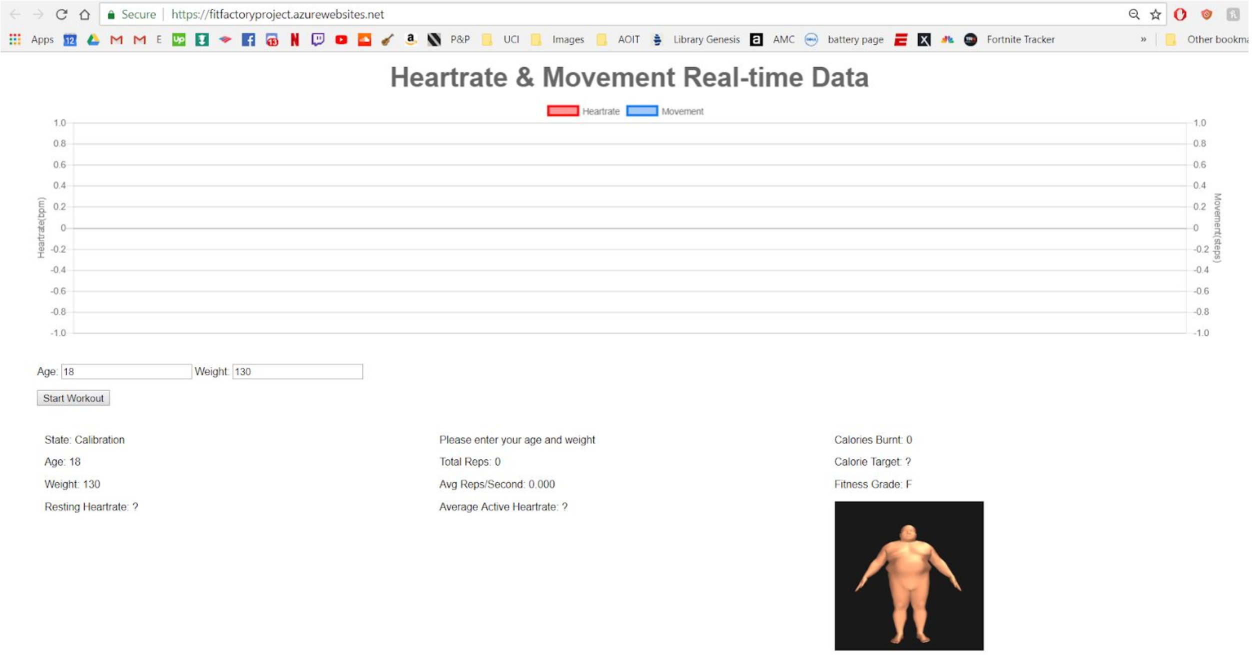
Age: Weight:

State: Calibration
Age: ?
Weight: ?
Resting Heartrate: ?

Please enter your age and weight
Total Reps: 0
Avg Reps/Second: 0.000
Average Active Heartrate: ?

Calories Burnt: 0
Calorie Target: ?
Fitness Grade: F

One of the variables at the bottom "State", serves as a guide as to the stage in the workout that the user is currently in. During the "Calibration" period the user must enter their Age and Weight, important variables when calculating calories burnt over a workout. In order to begin the workout, the heart rate sensor must be properly calibrated, so the workout does not proceed until it receives its first message from the Sparkfun (this means that calibration has been a success).



Heartrate & Movement Real-time Data

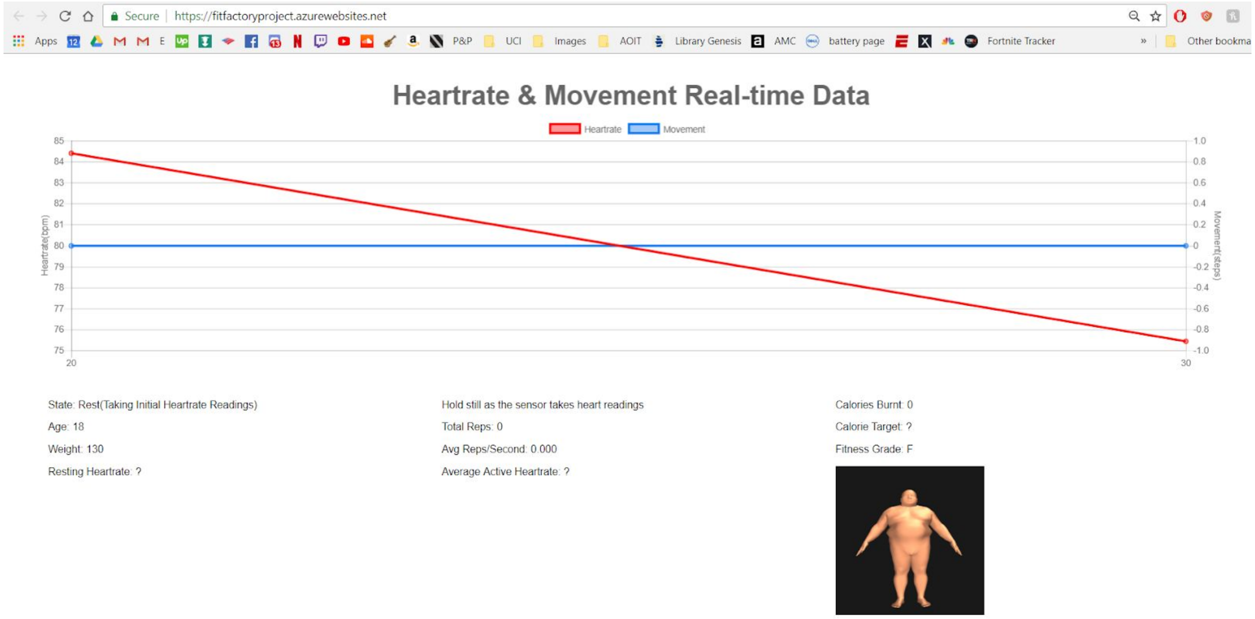
Age: 18 Weight: 130

State: Calibration
Age: 18
Weight: 130
Resting Heartrate: ?

Please enter your age and weight
Total Reps: 0
Avg Reps/Second: 0.000
Average Active Heartrate: ?

Calories Burnt: 0
Calorie Target: ?
Fitness Grade: F

Once calibration is complete, the user’s resting heart rate is found using three heart rate readings during the initial rest period. During this time, the application ignores any incoming data about the user’s movements.



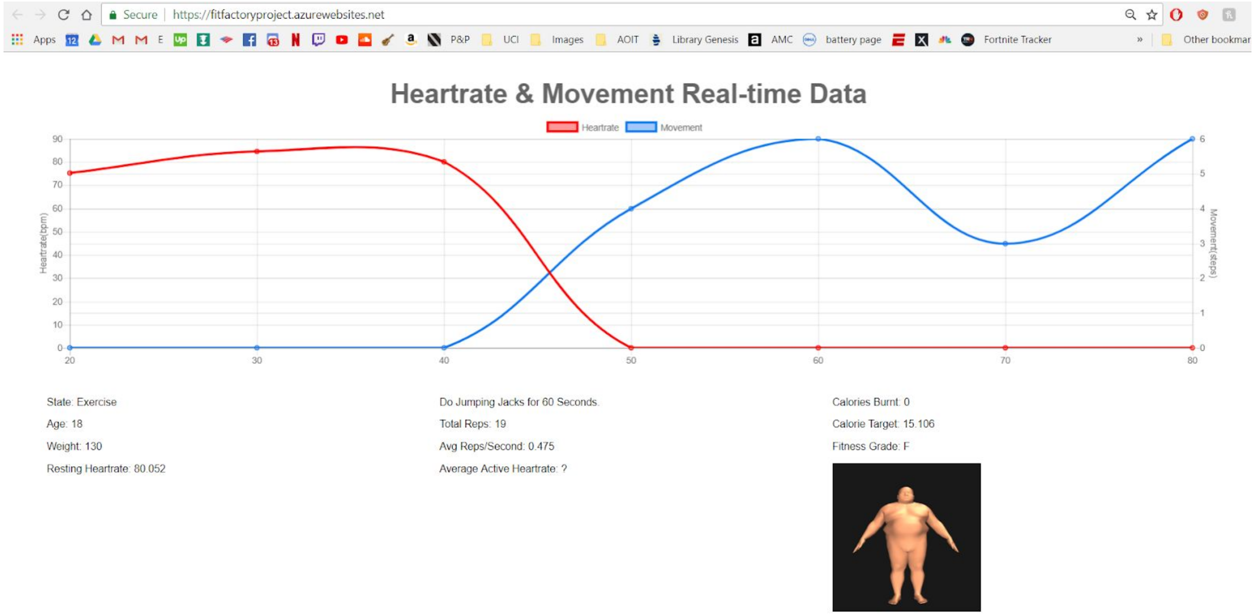
Heartrate & Movement Real-time Data

State: Rest(Taking Initial Heartrate Readings)
Age: 18
Weight: 130
Resting Heartrate: ?

Hold still as the sensor takes heart readings
Total Reps: 0
Avg Reps/Second: 0.000
Average Active Heartrate: ?

Calories Burnt: 0
Calorie Target: ?
Fitness Grade: F

Once the resting heart rate is found, the user is given their “Calorie Goal” and is prompted during their first “Exercise” period to do a certain exercise.



Heartrate & Movement Real-time Data

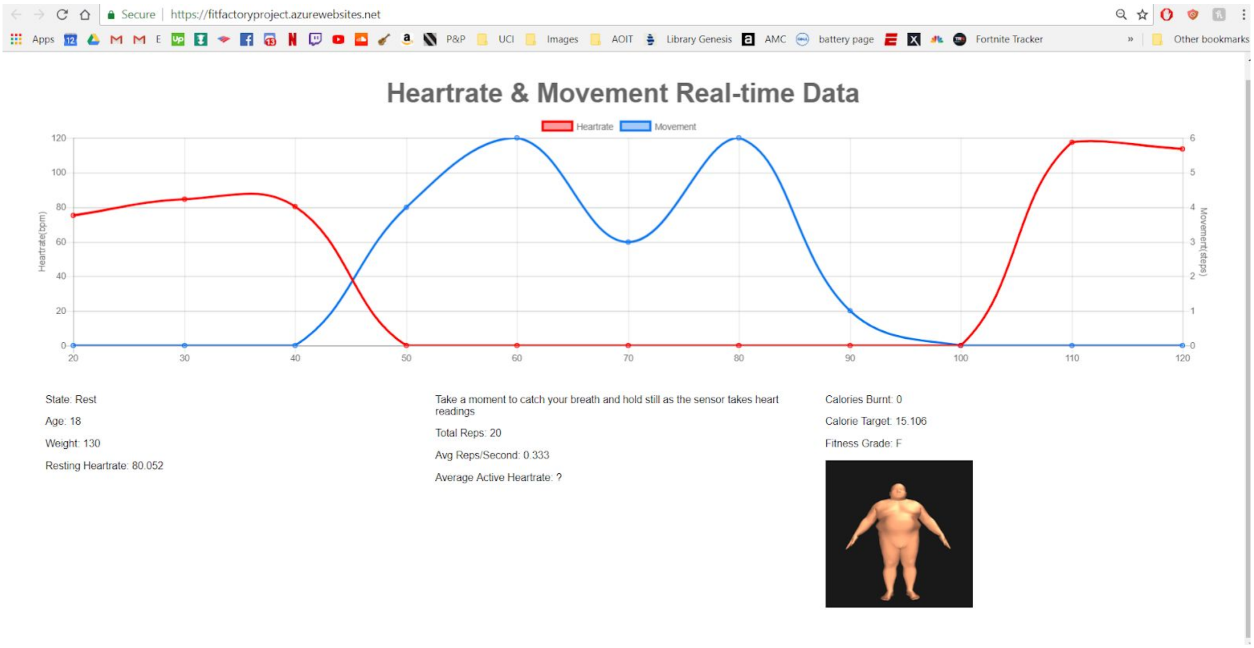
State: Exercise
Age: 18
Weight: 130
Resting Heartrate: 80.052

Do Jumping Jacks for 60 Seconds.
Total Reps: 19
Avg Reps/Second: 0.475
Average Active Heartrate: ?

Calories Burnt: 0
Calorie Target: 15.106
Fitness Grade: F

During this time, the heart rate readings ended up being very inaccurate due to the movement of the user, so I ignored incoming information about heart rate during each exercise period (hence the flat red line starting at t = 50). However, movement readings are recorded during the exercise states.

After the first exercise is complete, the user is prompted to rest and record their heart rate readings during this “Rest” state. After these readings are taken, the variables “Average Active Heartrate”, “calories burnt” , and “Grade” are all updated. The image of the fat figure is also changed here depending on how much progress the user has made toward their “Calorie Goal”. The application swaps between exercise and rest states until four sets have been completed.


Secure | https://fitfactoryproject.azurewebsites.net
Apps P&P UCI Images AOIT Library Genesis AMC battery page Fortnite Tracker Other bookmarks
Heartrate & Movement Real-time Data
Heartrate
Movement
Heartrate(bpm)
Movement(steps)
120
100
80
60
40
20
0
6
5
4
3
2
1
0
20
30
40
50
60
70
80
90
100
110
120

State: Rest

Age: 18

Weight: 130

Resting Heartrate: 80.052

Take a moment to catch your breath and hold still as the sensor takes heart readings

Total Reps: 20

Avg Reps/Second: 0.333

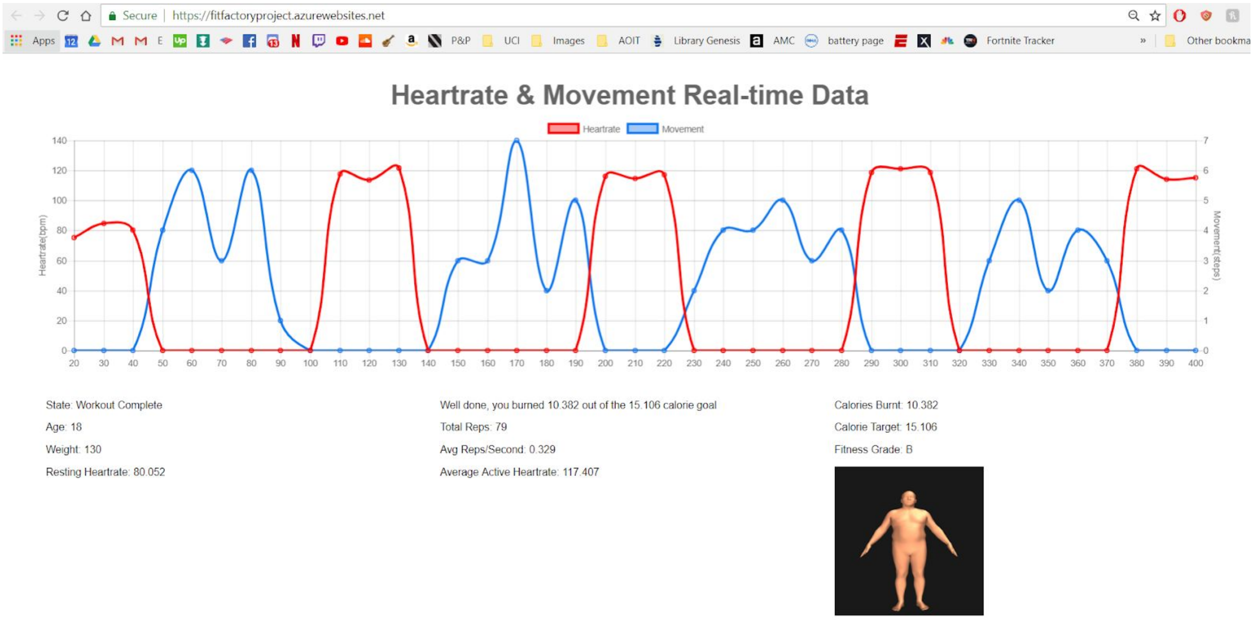
Average Active Heartrate: ?

Calories Burnt: 0

Calorie Target: 15.106

Fitness Grade: F

Once the workout is complete, the user is given a grade based on how many calories they have burned vs. their given calorie goal. Here our user burned 10.3 out of their 15.1 calorie goal, thus receiving a B grade. The figure is also much thinner than it initially was, but still could have been skinnier if the user had worked harder.



State: Workout Complete

Age: 18

Weight: 130

Resting Heartrate: 80.052

Well done, you burned 10.382 out of the 15.106 calorie goal

Total Reps: 79

Avg Reps/Second: 0.329

Average Active Heartrate: 117.407

Calories Burnt: 10.382

Calorie Target: 15.106

Fitness Grade: B