RasPi-Fog : Developer Tutorial

Shreshth Tuli[1]

June 2018

1 Introduction

This document includes the documentation of the code and insight on how different sections of the software can be tweaked or extended. All explanations and suggestions are based on the assumption that the reader has basic understanding of PHP and Java, and is well versed with programming concepts.
The document is divided into five major sections that deal with :

- Master and Worker web interface (PHP)
- Data Analyzer (Java)
- Android Interface (MIT AI)
- Functionalities of the Software
- Further scope of development

2 Web Interface

All communication between the Fog nodes works using HTTP REST APIs, specifically GET and POST.

2.1 Master

The Master performs the major tasks which include but are not limited to:

1. Take user login information and check with the Database
2. Maintain Database of registered users
3. Maintain configurations including Worker nodes' IP addresses
4. Take data for analysis
5. Distribute data with other parameters among worker nodes (or itself)

2.1.1 index.php

The code snippet below of "index.php" shows the HTML form for taking the login information from the user and sending it using the POST method.

```
<html>
<head><title>HealthKeeper - Login</title></head>
<body>

<!-- Title -->
<h1>HealthKeeper - Login Page</h1>

<!-- Login Form -->
<form id='login' action='index.php' method='post'
accept-charset='UTF-8'>
<fieldset >
<legend>Login</legend>
<label for='username' >Username:</label>
<br>
<input type='text' name='username' id='username'
maxlength="50" />
<br>
<label for='password' >Password:</label>
<br>
<input type='password' name='password' id='password'
maxlength="50" />
<br><br>
<input type='submit' name='Submit' value='Submit' />
</fieldset>
</form>
```

The next snippet shows the database settings to connect with the MySQL database. As described in the "End User Tutorial", the settings include parameters like name of database (users), name of table (registrations). Using the *mysqli_connect* command, the web-server is able to connect to the required database. The database connection is indicated whether successful or not using 2 echo commands.

Next the login information is checked using the SQL query to find entry with the entered credentials. If the number of rows is 1 then login is successful and the page navigates to "session.php" with GET tag of username. If login is not successful, then "Username or password incorrect" is displayed on the page.

```
<?php
session_start();

// Database settings
$host = "localhost";
$user = "root";
$pass = "";
```

```php
$db = "users";

// Connection to database
$dbConnected = mysqli_connect($host, $user, $pass, $db);
$dbSelected = mysqli_select_db($dbConnected, "users");
$dbSuccess = true;

// Show Database connection on screen
if($dbConnected){
        echo "MySQL Connection OK<br />";
        if($dbSelected){
                echo "DB Connection OK<br />";
        }
        else{
                echo "DB Connection FAIL<br />";
                $dbSuccess = false;
        }
}
else{
        echo "MySQL Connection FAIL<br />";
        $dbSuccess = false;
}

// Check login credentials
if(isset($_POST['username']) && isset($_POST['password'])
&& $dbSuccess){
        $username = $_POST['username'];
        $password = $_POST['password'];
        $sql = "select * from registrations where
    username='".$username."' AND
        password='".$password."' limit 1";
        $result = mysqli_query($dbConnected,$sql);
        if(mysqli_num_rows($result)==1){
                // Go to session page if login successful
                echo "Login Successful!";
                header('Location: session.php/?username='.
        $_POST['username']);
        }
        else{
                // Show incorrect credentials otherwise
                echo "Username or password incorrect!";
                exit();
        }
}
?>

</body>
</html>
```

2.1.2 manager.php

The "manager.php", handles all the worker IPs for the "session.php". It maintains the "config.txt", in which every line contains one IP address of a worker. "config.txt" also holds whether Master could be given the task of analysis or not. The following code snipper in the "manager.php", shows that if "Remove all Workers" is clicked, then "config.txt" is overwritten to a default state with no worker IP and Master enable to do the task.

```
<?php

// Remove all worker nodes
if(isset($_POST['remove'])){
        file_put_contents("config.txt", "EnableMaster".PHP_EOL);
        echo "All Workers removed<br/>";
}
```

The next snippet shows how the config.txt is parsed. The method $fgets$, return a next line each time it is called. The while loop stores each line in the $\$content$ variable skipping the first line which contains the checkbox value. As per the "Enable Master" checkbox, the string "EnableMaster" or "DesableMaser" is added to $\$content$.

```
{
        // Read IPs from config.txt
        $file = fopen("config.txt", "r");
        $content = "";
        $line = fgets($file);
        while(($line = fgets($file)) !== false){
                $content=$content.$line;
        }
        fclose($file);

        // Alter first line of config.txt as per
        // Enable master set or not
        if(isset($_POST['enable'])){
                file_put_contents("config.txt",
        "EnableMaster".PHP_EOL.$content);
        }
        else{
                file_put_contents("config.txt",
        "DisableMaster".PHP_EOL.$content);
        }
```

Whenever a new IP address is added, the POST tag $\$\_POST[`ip']$ is set and the IP is appended to the configuration file as shown in the snippet below.

```
        // If new IP added, add to config.txt
        if(isset($_POST['ip']) && $_POST['ip']!=""){
        $file = fopen("config.txt", "a");
        $k = $_POST['ip']."\n";
```

```
        echo "Worker IP added : ".$_POST['ip']."<br/>";
        fwrite($file, $k);
        fclose($file);

        }
```

The rest of the code displays the IPs set in the configuration file and lays down the form for adding new worker IP and the IP address of the local system (master).

```
        // Display IPs already set
        echo "Set Worker IPs here <br/>";
        $file = fopen("config.txt", "r");
        $line = fgets($file);
        while(($line = fgets($file)) !== false){
                echo "Worker IP : ".$line."<br/>";
        }
        fclose($file);
        echo "<br/>"."Add Worker IP<br/>";
        echo "
        <form id='ipinfo' method='post'>
        <input type='checkbox' name='enable' value='Yes'
        checked />
        Enable Master as Worker <br/>
        <input type='text' name='ip' id='ip'  maxlength=\"500\" /> <br/>
        <input type='submit' name='add' value='Add Worker' /> <br/><br/>
        <input type='submit' name='remove' value='Remove all workers' />
        </form>";
        }
}
$localIP = getHostByName(getHostName());
echo "Master IP address : ".$localIP;
?>

</body>
</html>
```

2.1.3 home.php

The "home.php" is like a main menu page where the user can navigate to manager page or the session page. The code below shows how this is achieved.

```
<html>
<head><title>HealthKeeper - Home</title></head>
<body>

<!-- Title -->
<h1>HealthKeeper - Home Page</h1>
<?php
session_start();
```

```
if(isset($_GET['username'])){
        $username = $_GET['username'];
        echo "<h2>Hello ".htmlspecialchars($username)."</h2> ";
}
else{
        echo "<h2>Hello</h2> ";
}
?>

<!-- Navigation Form -->
<form id='login' action='home.php' method='post' accept-charset='UTF-8'>
<input type='submit' name='manager' value='Go To Manager Page' />
    
<input type='submit' name='session' value='Go To Session Page' />
<br><br>
<input type='submit' name='back' value='Go Back' />
</fieldset>
</form>

<?php
$user = $_GET['username'];

if(isset($_POST['manager'])){
        header("Location: ../manager.php/?username=".$user);
}
elseif(isset($_POST['session'])) {
        header('Location: ../session.php/?username='.$user);
}
elseif(isset($_POST['back'])) {
        header('Location: ../');
}
?>
</body>
</html>
```

2.1.4 session.php

The "session.php" script performs most of the software functionalities. The initial part of the code welcomes user by using the username tag of the GET request.

```
<html>
<head><title>HealthKeeper - Login</title></head>
<body>
<h1>HealthKeeper - Session Page</h1>

<?php
session_start();

```

```php
// Welcome user
if(isset($_GET['username'])){
        $username = $_GET['username'];
        echo "<h2>Hello ".htmlspecialchars($username)."</h2> ";
}
else{
        echo "<h2>Hello</h2> ";
}
?>
```

Then, there is an HTML form for input of data from the user.

```html
<!-- Submit for analysis -->
<form id='data' method='get'>
<input type='text' name='data' id='data'  maxlength="500" />
<input type='submit' name='analyze' value='analyze' />
</form>
```

The next few lines, as shown in the code snippet below display the input data from the HTML form.

```php
<?php

// If data entered, show the data values stored in data.txt
if(isset($_GET['data'])){
        $file = fopen("data.txt", "a");
        $content = $_GET['data'];
        echo "Data Values Stored : ";
        echo $content;
        if(0==filesize("data.txt")){
                fwrite($file,$content);
        }else{
                fwrite($file,",".$content);
        }
        fclose($file);
}
?>
```

Next, if the "Analyze" button is clicked, then the content of "config.txt" is parsed to obtain list of worker IP addresses. The variable $\$toMaster$ stores if the task is to be given to the Master or not. It is initialized to false if first line in "config.txt" is "DisableMaster" and true otherwise.

The next few lines form the code for the "Load Balancing scheme" of the software. The variable $\$ips$ is an array of the IP addresses and $\$loads$ is their corresponding loads. The PHP method $file\_get\_contents()$ allows us to get a string form of the webpage passed as the argument. Using a for loop, the $\$loads$ array is populated by accessing the "load.php" of the corresponding IP address. If the load of any worker is $< 80\%$, then the variable $\$toMaster$ is set to false. In effect, when master is enabled as worker, this sets $\$toMaster$ to true only

when all worker nodes have more that 80% load.

If the task is not given to the master, i.e. $\$toMaster$ is false, then the IP of the worker with minimum load gets the task. The task is sent using the GET method to the "worker.php" script of the worker with that IP address.

```php
<?php

// If Analyze is clicked
if(isset($_GET['analyze'])){
        $content = $_GET['data'];

        // Parse config.txt for IPs
        $file = fopen("config.txt", "r");
        $line = fgets($file);

        // Initialize toMaster
        // true if work given to master, else false
        $toMaster = true;
        if($line == "DisableMaster"){
                $toMaster = false;
        }
        $ips = array();
        while(($line = fgets($file)) !== false){
          array_push($ips, $line);
        }

        // Initialize loads array to store loads of workers
        $loads = array();

    // For each IP, get load from load.php
        foreach($ips as $ip){
                $ip = preg_replace('/\s+/', '', $ip);
                $dataFromExternalServer=
        file_get_contents("http://".$ip."/HealthKeeper/load.php");
                $dataFromExternalServer =
        preg_replace('/\s+/', '', $dataFromExternalServer);
                $my_var = 0.0 + $dataFromExternalServer;
                echo "<br/>Woker load with IP ".$ip.": ".$my_var;
                array_push($loads, $my_var);
                // If any load < 80% then toMaser = false
                if($my_var <= 0.8){
                        $toMaster = false;
                }
        }
        $result = "";
        if(!$toMaster){
                // Work given to worker with least load
                $min = 100;
                $minindex = 0;
```

```
                foreach($loads as $load){
                        if ($min > $load){
                                $min = $load;
                        }
                }
                foreach($loads as $load ){
                        if($min == $load){
                                break;
                        }
                        $minindex = $minindex+1;
                }
                $ipworker = $ips[$minindex];
                $ipworker = preg_replace('/\s+/', '', $ipworker);
                // Send data
                echo "<br/><br/>Work sent to Worker ".($minindex+1)."
        with IP address : ".$ipworker."<br/><br/>";
                // Get result and store in $result variable
                $result = file_get_contents
                ('http://'.$ipworker.'/HealthKeeper/worker.php/?data='.
                $_GET['data']);
        }
        else {
                // Work done by master
                $minindex = 0;
                $ipworker = "localhost";
                echo "<br/><br/>Work Done by Master<br/><br/>";
                $result = file_get_contents('http://'.$ipworker.
                '/HealthKeeper/RPi/Worker/worker.php/?data='.
                $_GET['data']);
        }
        echo $result;
```

The remaining part of the code parser the data file with all data from the beginning and displays graph using JavaScript. It also removes all data by overwriting the data file with empty string.

```
        // Graph
        $file1 = fopen("data.txt", "r");
        $result = fgets($file1);
        $allArray = explode(",", $result);
        $dataPoints = array();
        $criticalPoints = array();
        foreach ($allArray as $value) {
    array_push($dataPoints, array("y" => (int)$value, "label" => "-"));
    array_push($criticalPoints, array("y" => 88, "label" => "-"));
        }
        fclose($file1);

}

```

```
?>
<script>
window.onload = function () {

var chart = new CanvasJS.Chart("chartContainer", {
        title: {
                text: "Sleep Apnea Graph"
        },
        axisY: {
                title: "Oxygen Level"
        },
        data: [{
                markerType: "none",
                type: "line",
                dataPoints:
        <?php echo json_encode($dataPoints, JSON_NUMERIC_CHECK); ?>},
                {
           markerType: "none",
                type: "line",
                dataPoints:
        <?php echo json_encode($criticalPoints, JSON_NUMERIC_CHECK); ?>
        }]
});
chart.render();

}
</script>
<div id="chartContainer" style="height: 370px; width: 50%;"></div>
<script src="https://canvasjs.com/assets/script/canvasjs.min.js"></script>
<form id='data' method='post'>
<input type='submit' name='reset' value='Reset All Data' />
</form>
<?php

// Reset all data
if(isset($_POST['reset'])){
        file_put_contents("data.txt", "");
        echo "All Data removed<br/>";
}
?>

</body>
</html>
```

2.2 Worker

The Worker node, displays CPU load, analyzes data and displays results.

2.2.1 load.php

The "load.php" uses the PHP method : *sys_getloadavg*() to get the system CPU load and displays it.

```
<?php
// Display CPU load
$load = sys_getloadavg();
echo $load[0];
?>
```

2.2.2 manager.php

The "manager.php" sets the Master IP address for data verification. It also displays the current set IP address of the Master which is saved in the configuration file (config.txt).

```
<html>
<head><title>HealthKeeper - Manager</title>
</head>
<body>
<?php
if(isset($_POST['ip'])){
        // Set Master IP
        $file = fopen("config.txt", "w+");
        $k = "Master IP : ".$_POST['ip'];
        echo "Master IP Set to : ".$_POST['ip'].PHP_EOL;
        fwrite($file, $k);
        fclose($file);

}
else{
        echo "Set Master IP here".PHP_EOL;
        $file = fopen("config.txt", "r");
        $k = fgets($file);
        echo $k.PHP_EOL;
        // Form to input Master IP
        echo "
        <form id='ip' method='post'>
        <input type='text' name='ip' id='ip'  maxlength=\"500\" />
        </form>";
        echo    "<form id='Change IP' method='post'>
        <input type='submit' name='Change IP' value='Change IP' />
        </form>";
        fclose($file);

```

```
}
?>
</body>
</html>
```

2.2.3 worker.php

The "worker.php" script receives data to be analyzed using the GET method. It saves the data in "data.txt" and sets the first line to "Analysis Done = false". The Java based analysis application parses this data, saves results in "result.txt" and changes the first line of "data.txt" to "Analysis Done = true". This script once data has been written waits for the first line to change to "Analysis Done = true", and when so parses the result file. The result file's first line contains two parameters for analysis:

1. Number of times the oxygen level went below 88
2. Least oxygen level

The current analysis scheme determines the disease severity on the basis of the following thresholds of the first parameter:

- < 5 : None
- between 5 and 15 : Mild
- between 15 and 30 : Moderate
- > 30 : Highly severe

```
<html>
<head><title>HealthKeeper - Worker</title></head>
<body>

<?php
if(isset($_GET['data'])){

        // Write Data to file
        $content = $_GET['data'];
        $file = fopen("data.txt", "w+");
        fwrite($file, "Analysis Done = false".PHP_EOL);
        fwrite($file,$content.PHP_EOL);
        fclose($file);

        // Wait for analysis done
        $file = fopen("data.txt", "r");
        $k = fgets($file);
        while(!preg_match("/Analysis Done = true/", $k)){
                fclose($file);
```

```
                $file = fopen("data.txt", "r");
                $k = fgets($file);
                usleep(500000);
        }
        fclose($file);

        // Read results and display
        $file1 = fopen("result.txt", "r");
        $result = fgets($file1);
        $array = explode(",", $result);
        $count = (int)$array[0];
        $min = (int)$array[1];
        echo "For 1 hour of sleep data<br />";
        echo "AHI (Apnea-hypopnea index) = ".$count."<br />";
        echo "Minimum Oxygen Level reached : ".$min."<br />";
        $sev = "";
        if($count < 5){
                $sev = "None";
        }
        elseif($count < 15){
                $sev = "mild";
        }
        elseif($count < 30){
                $sev = "moderate";
        }
        else{
                $sev = "Highly severe!";
        }
        echo "Disease severity : ".$sev;
}
?>
</body>
</html>
```

3 Data Analyzer

The "Sleep Apnea Analyzer" code is developed in Java. The code shown below is the complete code for the data analysis. There is one fileReader for "data.txt" and two fileWriters, one for "result.txt" and other for "data.txt". The whole working code is inside a while(true) loop so that it works indefinitely. The data fileReader waits for the first line of data to turn to "Analysis Done = false" at time gaps of 500 milliseconds, which means that the worker php script has written new data for analysis. It checks this using the bufferedReader.readLine() method.

Next, when the new data is found in the data file, it parses it and splits the string with "," as a delimiter, and converts all strings to integers. The analysis

is done using in the following way:

- There is a *dip* boolean variable, which stores whether there is a dip in oxygen level, a *count* which stores the number of times *dip* changes to true and a *min* which stores minimum oxygen level
- Whenever the oxygen level goes below 88, *dip* turns to true and stays true till oxygen level comes above 88.
- This count becomes the AHI (Apnea - Hypopnea Index), used to determine the disease severity

Once done, the result is written to the result file, and the first line of data fie is changed to "Analysis Done = true" indicating the worker script to take results from the result file.

```
1  import java.io.*;
2  import java.util.*;
3  import static java.lang.Integer.parseInt;
4
5  public class analyzer{
6
7          public static void main(String[] args) throws
               Exception {
8                  int i;
9                  FileReader fileReader = new
                       FileReader("data.txt");
10                 FileWriter resultfile = new
                       FileWriter("result.txt");
11                 FileWriter datafile;
12                 BufferedWriter resultwriter;
13                 BufferedWriter datawriter;
14                 BufferedReader bufferedReader;
15                 String line = "empty";
16                 String line2 = "empty";
17                 int count;
18                 int min;
19                 boolean dip;
20                 BufferedWriter writer;
21                 String[] datastring;
22                 Integer[] data;
23
24                 while(true){
25
26                         // Wait for analysis false
27                         bufferedReader = new
                               BufferedReader(fileReader);
```

```
28                          while(true){
29                                  fileReader = new
                                        FileReader("data.txt");
30                                  bufferedReader = new
                                        BufferedReader(fileReader);
31                                  line =
                                        bufferedReader.readLine();
32                                  System.out.println(line);
33                                  if(line.equals("Analysis
                                        Done = false")){
34                                          break;
35                                  }
36                                  Thread.sleep(500);
37                          }
38
39                          line2 =
                                bufferedReader.readLine();
40                          System.out.println(line);
41                          System.out.println(line2);
42                          bufferedReader.close();
43
44                          // parse data
45                          datastring = line2.split(",");
46                          data = new
                                Integer[datastring.length];
47                          i=0;
48                  for(String str:datastring){
49                          data[i]=Integer.parseInt(str);
50                          i++;
51                  }
52
53                  // Analyze data
54                  count = 0;
55                  min = 100;
56                  dip = false;
57                  for(int j = 0; j < data.length; j++){
58                          if(data[j] <= 88 && !dip){
59                                  count++;
60                                  dip = true;
61                          }
62                          else if(data[j] > 88 && dip){
63                                  dip = false;
64                          }
65                          if(min > data[j]){
66                                  min = data[j];
67                          }
```

```
68                  }
69
70          // Write results to file
71                  resultfile = new
                        FileWriter("result.txt");
72                  resultwriter = new
                        BufferedWriter(resultfile);
73                  datafile = new FileWriter("data.txt");
74                  datawriter = new
                        BufferedWriter(datafile);
75                  resultwriter.write(count+","+min);
76                  resultwriter.newLine();
77                  resultwriter.write(line2.substring(1,line2.length())+"\n");
78                  datawriter.write("Analysis Done = true");
79                  datawriter.newLine();
80                  datawriter.write(line2.substring(1,line2.length())+"\n");
81                  resultwriter.close();
82                  datawriter.close();
83                  System.out.println("Data Analysis
                        Done!");
84                  }
85          }
86  }
```

4 Android Interface

The android app "HealthKeeper.apk" allows the android device to act as an intermediary between the Oximeter sensor and the Master server. Rather than sending data manually through the GET HTML form, this app records and sends data automatically. The app has been developed on an open source platform : "MIT App Inventor" which can be seen at this link. The source file for "HealthKeeper.apk" can be found here.

The application is divided into two screens namely : Welcome screen and Session screen. As shown in the End User tutorial, the Welcome screen asks user to pair the Bluetooth Oximeter with the Android device and enter the Master server's IP address. The block based code for the Welcome screen is shown in Figure 1.

4.1 Welcome Screen

The *masterip* is the global variable which stores the IP address entered by the user. The *list* global variable stores a list of *masterip* and Bluetooth device address. The *PairSensor* is a list picker to select from the available bluetooth

devices. After picking, the BluetoothClient connects to the selected address and displays it on the screen. When the "Connect" button is clicked, the list objects are appended and sent to the Session screen.

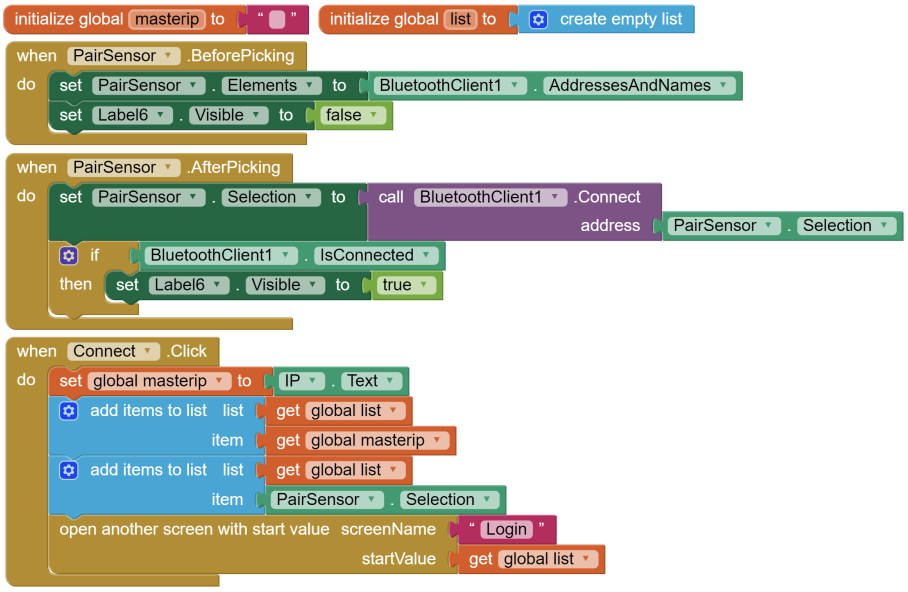

Figure 1: Welcome Screen block code

4.2 Session Screen

The Session screen is the main screen that handles all interaction with the master server. The blocks below show the variable initializations. The *data* variable stores the data received from sensor in list form, *loop* variable is a boolean for timer, and *url* variable stores the web address of the master node's index page.

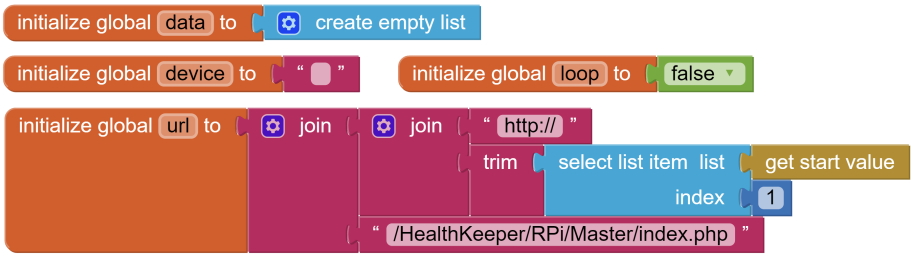

Figure 2: Session Screen block code

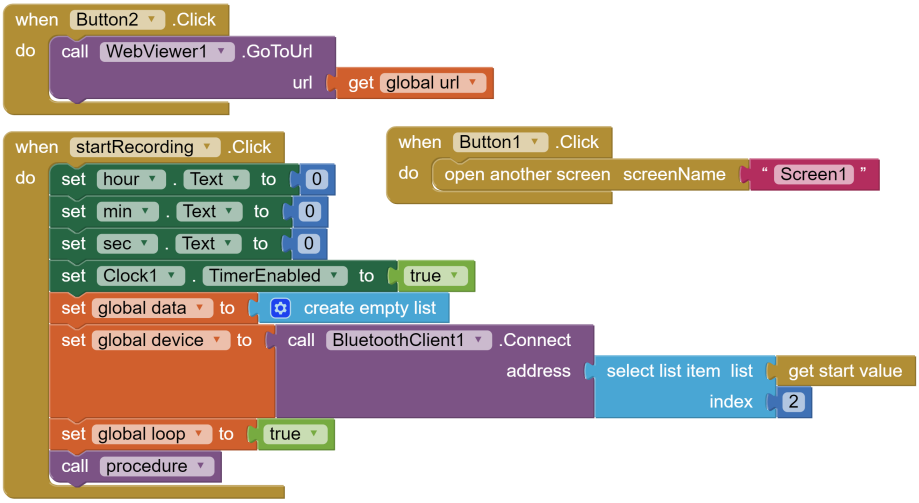

Figure 3: Session Screen block code

The "Refresh" button connects the "WebViewer" to the *url*. When the *StartRecording* button is clicked, the timer is reset, the BluetoothClient connects to the address and starts collecting data from the sensor till the *StopRecording* button is clicked. The procedure that executes is shown in Figure 4.

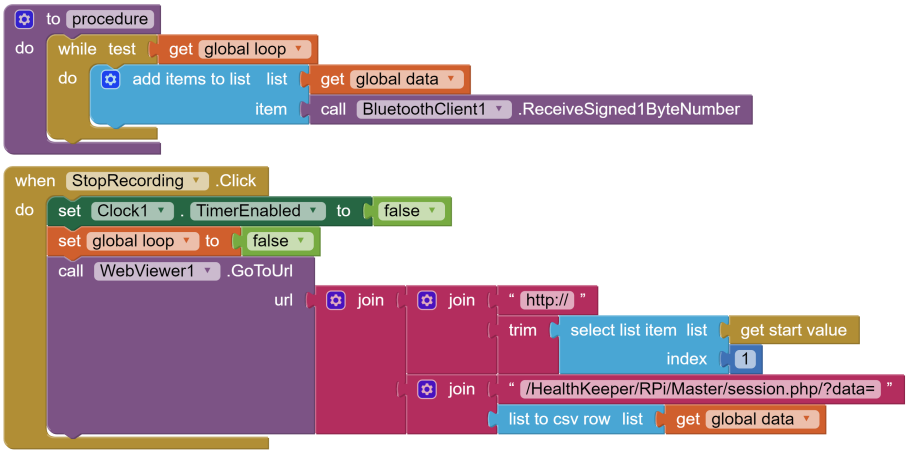

Figure 4: Session Screen block code

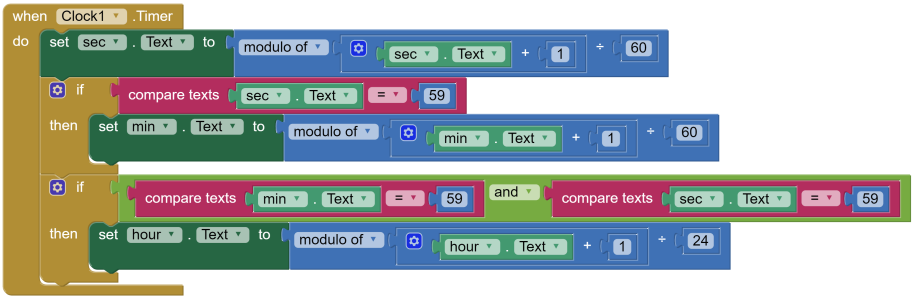

Figure 5: Session Screen block code

When the *StopRecording* button is clicked, the data is submitted to the master node via the GET method. When *Analyze* button is clicked, the data is submitted for analysis and results displayed on the WebViewer. The timer function is shown in Figure 5 and analysis in Figure 6.

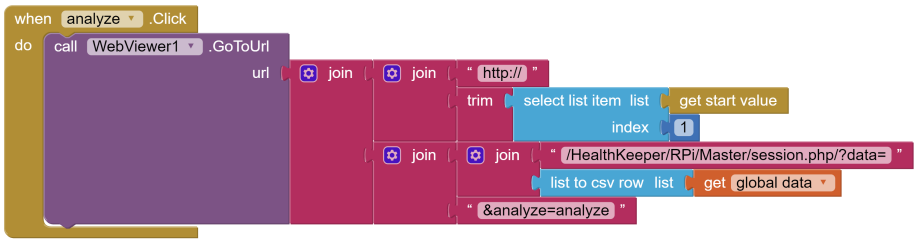

Figure 6: Session Screen block code

5 Functionalities of the Software

The software has the following major functionalities which may be expanded further:

1.

6 Further Scope of developments

This software form a base for setting up Fog Computing Environment which is not OS or architecture specific. Though the framework is complete by itself and fully functions to perform real-time, sensor based Sleep Apnea analysis, but has a large scope of improvement and further developments:

1. **Load Balancing Scheme** : As discussed earlier, the current load balancing scheme is naive and can be greatly improved. Currently, the load balancing scheme focuses on task distribution based on CPU load, whichever

device has the least load, the task is given to it. There can be a cumulative ranking parameter which takes into account many other factors as per requirement. These factors can be and are not limited to : Network Bandwidth, Memory Load, etc. Different weights can given to these parameters are per the scenario.

2. **Data Integrity** : Health Analysis data is important for patients as their treatment and lives depend on it, thus it is important to save this data from fraudulent manipulation or sabotage from hackers. Thus, to maintain data integrity many techniques like Blockchain can be implemented to ensure that data is secure.

3. **Data Privacy** : Some applications require data to be secure as well as kept private. The current system is prone to attacks and unwanted display of data to others. Privacy policies like encrypting the data can be used to ensure that the data can not be seen by others.

4. **Data Authenticity** : The current system allows any device to connect to master and share or view data. This can be used by hackers to forge DDoS or similar attacks. As Fog platforms also contains low range devices with limited threshold management, such attacks even at a low scale can destroy such devices. Thus a signature based validation technique can be used to ensure user and data authenticity.