

RasPi-Fog : Developer Tutorial

Shreshth Tuli¹

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.

```
1 <html>
2 <head><title>HealthKeeper - Login</title></head>
3 <body>
4
5 <!-- Title -->
6 <h1>HealthKeeper - Login Page</h1>
7
8 <!-- Login Form -->
9 <form id='login' action='index.php' method='post'
10 accept-charset='UTF-8'>
11 <fieldset >
12 <legend>Login</legend>
13 <label for='username' >Username:</label>
14 <br>
15 <input type='text' name='username' id='username'
16 maxlength="50" />
17 <br>
18 <label for='password' >Password:</label>
19 <br>
20 <input type='password' name='password' id='password'
21 maxlength="50" />
22 <br><br>
23 <input type='submit' name='Submit' value='Submit' />
24 </fieldset>
25 </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.

```
1 <?php
2 session_start();
3
4 // Database settings
5 $host = "localhost";
6 $user = "root";
7 $pass = "";
```

```

8 $db = "users";
9
10 // Connection to database
11 $dbConnected = mysqli_connect($host, $user, $pass, $db);
12 $dbSelected = mysqli_select_db($dbConnected, "users");
13 $dbSuccess = true;
14
15 // Show Database connection on screen
16 if($dbConnected){
17     echo "MySQL Connection OK<br />";
18     if($dbSelected){
19         echo "DB Connection OK<br />";
20     }
21     else{
22         echo "DB Connection FAIL<br />";
23         $dbSuccess = false;
24     }
25 }
26 else{
27     echo "MySQL Connection FAIL<br />";
28     $dbSuccess = false;
29 }
30
31 // Check login credentials
32 if(isset($_POST['username']) && isset($_POST['password'])
33 && $dbSuccess){
34     $username = $_POST['username'];
35     $password = $_POST['password'];
36     $sql = "select * from registrations where
37 username='". $username. "' AND
38 password='". $password. "' limit 1";
39 $result = mysqli_query($dbConnected, $sql);
40 if(mysqli_num_rows($result)==1){
41     // Go to session page if login successful
42     echo "Login Successful!";
43     header('Location: session.php/?username='.
44 $_POST['username']);
45 }
46 else{
47     // Show incorrect credentials otherwise
48     echo "Username or password incorrect!";
49     exit();
50 }
51 }
52 ?>
53
54 </body>
55 </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 snippet 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.

```
1 <?php
2
3 // Remove all worker nodes
4 if(isset($_POST['remove'])){
5     file_put_contents("config.txt", "EnableMaster".PHP_EOL);
6     echo "All Workers removed<br/>";
7 }
```

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 “DisableMaster” is added to *\$content*.

```
1 {
2     // Read IPs from config.txt
3     $file = fopen("config.txt", "r");
4     $content = "";
5     $line = fgets($file);
6     while(($line = fgets($file)) !== false){
7         $content=$content.$line;
8     }
9     fclose($file);
10
11     // Alter first line of config.txt as per
12     // Enable master set or not
13     if(isset($_POST['enable'])){
14         file_put_contents("config.txt",
15             "EnableMaster".PHP_EOL.$content);
16     }
17     else{
18         file_put_contents("config.txt",
19             "DisableMaster".PHP_EOL.$content);
20     }
```

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.

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

```

5     echo "Worker IP added : ".$_POST['ip']."<br/>";
6     fwrite($file, $k);
7     fclose($file);
8
9     }

```

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).

```

1     // Display IPs already set
2     echo "Set Worker IPs here <br/>";
3     $file = fopen("config.txt", "r");
4     $line = fgets($file);
5     while(($line = fgets($file)) != false){
6         echo "Worker IP : ".$line."<br/>";
7     }
8     fclose($file);
9     echo "<br/>". "Add Worker IP<br/>";
10    echo "
11    <form id='ipinfo' method='post'>
12    <input type='checkbox' name='enable' value='Yes'
13    checked />
14    Enable Master as Worker <br/>
15    <input type='text' name='ip' id='ip'  maxlength=\"500\" /> <br/>
16    <input type='submit' name='add' value='Add Worker' /> <br/><br/>
17    <input type='submit' name='remove' value='Remove all workers' />
18    </form>";
19    }
20 }
21 $localIP = getHostByName(getHostName());
22 echo "Master IP address : ".$localIP;
23 ?>
24 <form id='login' action='home.php' method='post' accept-charset='UTF-8'>
25 <br><br>
26 <input type='submit' name='back' value='Go Back' />
27 </fieldset>
28 </form>
29
30 <?php
31 if(isset($_GET['username']))
32 {$user = $_GET['username'];}
33 else{$user = '';}
34 if(isset($_POST['back'])) {
35     header('Location: ../home.php?username='.$user);
36 }
37
38 ?>
39 </body>
40 </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.

```

1 <html>
2 <head><title>HealthKeeper - Home</title></head>
3 <body>
4
5 <!-- Title -->
6 <h1>HealthKeeper - Home Page</h1>
7 <?php
8 session_start();
9 if(isset($_GET['username'])) {
10     $username = $_GET['username'];
11     echo "<h2>Hello ".htmlspecialchars($username)."</h2> ";
12 }
13 else {
14     echo "<h2>Hello</h2> ";
15 }
16 ?>
17
18 <!-- Navigation Form -->
19 <form id='login' action='home.php' method='post' accept-charset='UTF-8'>
20 <input type='submit' name='manager' value='Go To Manager Page' />
21 &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
22 <input type='submit' name='session' value='Go To Session Page' />
23 <br><br>
24 <input type='submit' name='back' value='Go Back' />
25 </fieldset>
26 </form>
27
28 <?php
29 $user = $_GET['username'];
30
31 if(isset($_POST['manager'])) {
32     header("Location: ../manager.php/?username=".$user);
33 }
34 elseif(isset($_POST['session'])) {
35     header('Location: ../session.php/?username='.$user);
36 }
37 elseif(isset($_POST['back'])) {
38     header('Location: ../');
39 }
40 ?>
41 </body>
42 </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.

```
1 <html>
2 <head><title>HealthKeeper - Login</title></head>
3 <body>
4 <h1>HealthKeeper - Session Page</h1>
5
6 <?php
7 session_start();
8
9 // Welcome user
10 if(isset($_GET['username'])){
11     $username = $_GET['username'];
12     echo "<h2>Hello ".htmlspecialchars($username). "</h2> ";
13 }
14 else{
15     echo "<h2>Hello</h2> ";
16 }
17 ?>
```

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

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

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

```
1 <?php
2
3 // If data entered, show the data values stored in data.txt
4 if(isset($_GET['data'])){
5     $file = fopen("data.txt", "a");
6     $content = $_GET['data'];
7     echo "Data Values Stored : ";
8     echo $content;
9     if(0==filesize("data.txt")){
10         fwrite($file,$content);
11     }else{
12         fwrite($file," ".$content);
13     }
14     fclose($file);
15 }
16 ?>
```

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 “Failover” and “Load Balancing” schemes 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. The “@” operator with the *file_get_contents()* is the error operator which lets it return *FALSE* in case of errors. If any HTTP request for load returns *FALSE* then *\$my_var* is set to 100, and error message is displayed on the screen. Setting load to 100 has the effect that this node would never get the task and hence removed from load balancing IPs. The next time the worker IPs are accessed for load, this is checked again and if the particular worker node is available, it is taken into the load balancer.

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 than 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.

```
1 <?php
2
3 // If Analyze is clicked
4 if(isset($_GET['analyze'])){
5     $content = $_GET['data'];
6
7     // Parse config.txt for IPs
8     $file = fopen("config.txt", "r");
9     $line = fgets($file);
10
11     // Initialize toMaster
12     // true if work given to master, else false
13     $toMaster = true;
14     if($line == "DisableMaster"){
15         $toMaster = false;
16     }
17     $ips = array();
18     while(($line = fgets($file)) != false){
19         array_push($ips, $line);
20     }
21
22     // Initialize loads array to store loads of workers
```



```

23     $loads = array();
24
25     // For each IP, get load from load.php
26     foreach($ips as $ip){
27         $ip = preg_replace('/\s+/', '', $ip);
28         $dataFromExternalServer =
29         @file_get_contents("http://".$ip."/HealthKeeper/load.php");
30         if($dataFromExternalServer != FALSE){
31             $dataFromExternalServer =
32             preg_replace('/\s+/', '', $dataFromExternalServer);
33             $my_var = 0.0 + $dataFromExternalServer;
34             echo "<br/>Woker load with IP ".$ip.": ".$my_var;
35         } else{
36             $my_var = 100;
37             echo "<br/>Woker with IP ".$ip.
38             ": compromised - Error \"Could not connect to fog node\"";
39         }
40         array_push($loads, $my_var);
41         // If any load < 80% then toMaser = false
42         if($my_var <= 0.8){
43             $toMaster = false;
44         }
45     }
46     $result = "";
47     if(!$toMaster){
48         // Work given to worker with least load
49         $min = 100;
50         $minindex = 0;
51         foreach($loads as $load){
52             if ($min > $load){
53                 $min = $load;
54             }
55         }
56         foreach($loads as $load ){
57             if($min == $load){
58                 break;
59             }
60             $minindex = $minindex+1;
61         }
62         $ipworker = $ips[$minindex];
63         $ipworker = preg_replace('/\s+/', '', $ipworker);
64         // Send data
65         echo "<br/><br/>Work sent to Worker ".$($minindex+1).
66         "with IP address : ".$ipworker."<br/><br/>";
67         // Get result and store in $result variable
68         $result = file_get_contents
69         ('http://'.$ipworker.'/HealthKeeper/worker.php/?data='.
70         $_GET['data']);
71     }
72     else {

```

```

73         // Work done by master
74         $minindex = 0;
75         $ipworker = "localhost";
76         echo "<br/><br/>Work Done by Master<br/><br/>";
77         $result = file_get_contents('http://'.$ipworker.
78             '/HealthKeeper/RPi/Worker/worker.php/?data=' .
79             $_GET['data']);
80     }
81     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.

```

1         // Graph
2         $file1 = fopen("data.txt", "r");
3         $result = fgets($file1);
4         $allArray = explode(",", $result);
5         $dataPoints = array();
6         $criticalPoints = array();
7         foreach ($allArray as $value) {
8             array_push($dataPoints, array("y" => (int)$value, "label" => "-"));
9             array_push($criticalPoints, array("y" => 88, "label" => "-"));
10        }
11        fclose($file1);
12    }
13}
14
15?>
16
17<script>
18window.onload = function () {
19
20var chart = new CanvasJS.Chart("chartContainer", {
21    title: {
22        text: "Sleep Apnea Graph"
23    },
24    axisY: {
25        title: "Oxygen Level"
26    },
27    data: [{
28        markerType: "none",
29        type: "line",
30        dataPoints:
31        <?php
32        echo json_encode($dataPoints, JSON_NUMERIC_CHECK);
33        ?>},
34        {
35            markerType: "none",
36            type: "line",

```

```

37         dataPoints:
38             <?php
39                 echo json_encode($criticalPoints, JSON_NUMERIC_CHECK);
40             ?>
41         }
42     });
43     chart.render();
44 }
45 }
46
47 </script>
48
49 <div id="chartContainer" style="height: 370px; width: 50%;"></div>
50 <script src="https://canvasjs.com/assets/script/canvasjs.min.js">
51 </script>
52
53 <form id='data' method='post'>
54 <input type='submit' name='reset' value='Reset All Data' />
55 </form>
56
57 <?php
58
59 // Reset all data
60 if(isset($_POST['reset'])){
61     file_put_contents("data.txt", "");
62     echo "All Data removed<br/>";
63 }
64 ?>
65
66 </body>
67 </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.

```

1 <?php
2 // Display CPU load
3 $load = sys_getloadavg();
4 echo $load[0];
5 ?>

```

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).

```
1 <html>
2 <head><title>HealthKeeper - Manager</title>
3 </head>
4 <body>
5 <?php
6 if(isset($_POST['ip'])){
7     // Set Master IP
8     $file = fopen("config.txt", "w+");
9     $k = "Master IP : ".$_POST['ip'];
10    echo "Master IP Set to : ".$_POST['ip'].PHP_EOL;
11    fwrite($file, $k);
12    fclose($file);
13
14 }
15 else{
16     echo "Set Master IP here".PHP_EOL;
17     $file = fopen("config.txt", "r");
18     $k = fgets($file);
19     echo $k.PHP_EOL;
20     // Form to input Master IP
21     echo "
22     <form id='ip' method='post'>
23     <input type='text' name='ip' id='ip' maxlength=\"500\" />
24     </form>";
25     echo "    <form id='Change IP' method='post'>
26     <input type='submit' name='Change IP' value='Change IP' />
27     </form>";
28     fclose($file);
29
30 }
31 ?>
32 </body>
33 </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

```
1 <html>
2 <head><title>HealthKeeper - Worker</title></head>
3 <body>
4
5 <?php
6 if(isset($_GET['data'])){
7
8     // Write Data to file
9     $content = $_GET['data'];
10    $file = fopen("data.txt", "w+");
11    fwrite($file, "Analysis Done = false".PHP_EOL);
12    fwrite($file,$content.PHP_EOL);
13    fclose($file);
14
15    // Wait for analysis done
16    $file = fopen("data.txt", "r");
17    $k = fgets($file);
18    while(!preg_match("/Analysis Done = true/", $k)){
19        fclose($file);
20        $file = fopen("data.txt", "r");
21        $k = fgets($file);
22        usleep(500000);
23    }
24    fclose($file);
25
26    // Read results and display
27    $file1 = fopen("result.txt", "r");
28    $result = fgets($file1);
29    $array = explode(",", $result);
30    $count = (int)$array[0];
31    $min = (int)$array[1];
32    echo "For 1 hour of sleep data<br />";
33    echo "AHI (Apnea-hypopnea index) = ".$count."<br />";
34    echo "Minimum Oxygen Level reached : ".$min."<br />";
35    $sev = "";
36    if($count < 5){
```

```
37         $sev = "None";
38     }
39     elseif($count < 15){
40         $sev = "mild";
41     }
42     elseif($count < 30){
43         $sev = "moderate";
44     }
45     else{
46         $sev = "Highly severe!";
47     }
48     echo "Disease severity : ".$sev;
49 }
50 ?>
51 </body>
52 </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 file 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     public static void main(String[] args) throws
7         Exception {
8         int i;
9         FileReader fileReader = new
10             FileReader("data.txt");
11         FileWriter resultfile = new
12             FileWriter("result.txt");
13         FileWriter datafile;
14         BufferedWriter resultwriter;
15         BufferedWriter datawriter;
16         BufferedReader bufferedReader;
17         String line = "empty";
18         String line2 = "empty";
```

```

16         int count;
17         int min;
18         boolean dip;
19         BufferedWriter writer;
20         String[] datastring;
21         Integer[] data;
22
23         while(true){
24
25             // Wait for analysis false
26             bufferedReader = new
27                 BufferedReader(fileReader);
28             while(true){
29                 fileReader = new
30                     FileReader("data.txt");
31                 bufferedReader = new
32                     BufferedReader(fileReader);
33                 line =
34                     bufferedReader.readLine();
35                 System.out.println(line);
36                 try{
37                     if(line.equals("Analysis
38                         Done =
39                         false")){
40                         break;
41                     }
42                 }
43                 catch(Exception e){}
44
45                 Thread.sleep(500);
46             }
47             line2 =
48                 bufferedReader.readLine();
49             System.out.println(line);
50             System.out.println(line2);
51             bufferedReader.close();
52
53             // parse data
54             datastring = line2.split(",");
55             data = new
56                 Integer[datastring.length];
57             i=0;
58             try{
59                 for(String
60                     str:datastring){
61                     data[i]=Integer.parseInt(str);

```



```

53         i++;
54     }
55 }
56 catch(Exception e){}
57
58 // Analyze data
59 count = 0;
60 min = 100;
61 dip = false;
62 System.out.println("Number of data
63     values :"+data.length);
64 if(data.length != 1){
65     for(int j = 0; j < data.length;
66         j++){
67         if(data[j] <= 88 &&
68             !dip){
69             count++;
70             dip = true;
71         }
72         else if(data[j] > 88 &&
73             dip){
74             dip = false;
75         }
76         if(min > data[j]){
77             min = data[j];
78         }
79     }
80
81 // Write results to file
82 resultfile = new
83     FileWriter("result.txt");
84 resultwriter = new
85     BufferedWriter(resultfile);
86 datafile = new
87     FileWriter("data.txt");
88 datawriter = new
89     BufferedWriter(datafile);
90 resultwriter.write(count+" "+min);
91 resultwriter.newLine();
92 resultwriter.write(line2.substring(1,line2.length()+"\n");
93 datawriter.write("Analysis Done =
94     true");
95 datawriter.newLine();
96 datawriter.write(line2.substring(1,line2.length()+"\n");
97 resultwriter.close();
98 datawriter.close();

```

```
90
91         System.out.println("Data
92             Analysis Done!");
93     }
94     else{
95         System.out.println("Empty Data");
96         Thread.sleep(500);
97     }
98 }
99 }
```

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.

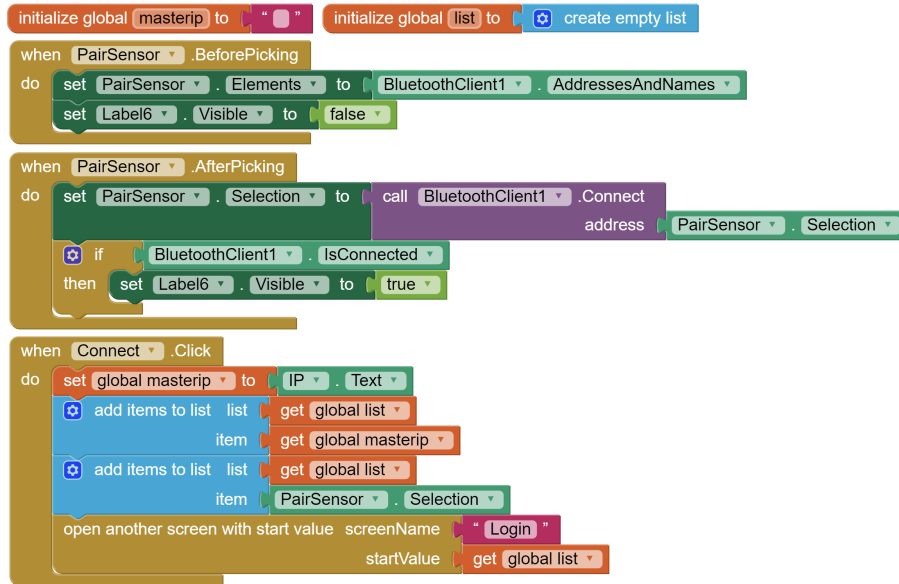


Figure 1: Welcome Screen block code

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.

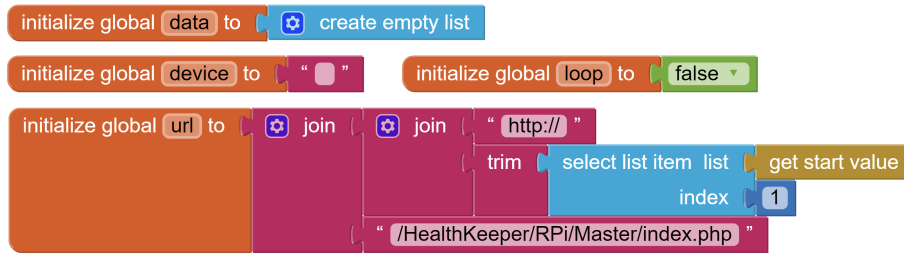


Figure 2: Session Screen block code

The “Refresh” button connects the “WebView” 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.

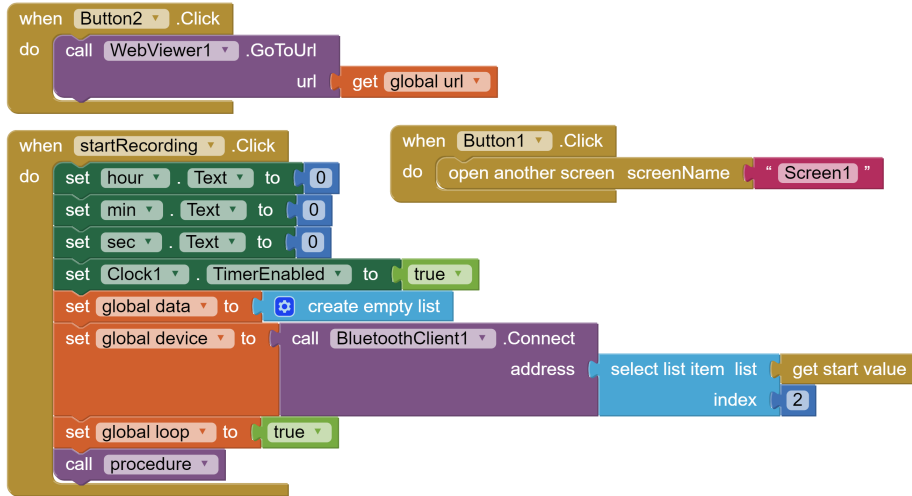


Figure 3: Session Screen block code

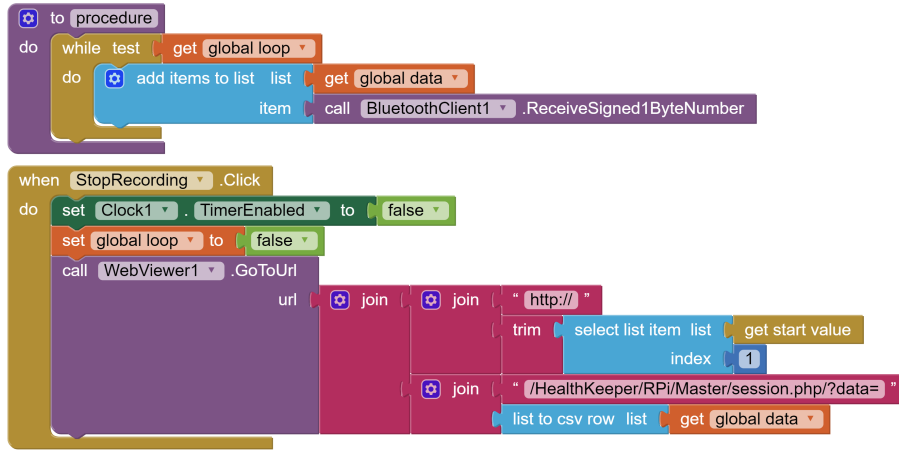


Figure 4: Session Screen block code

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.

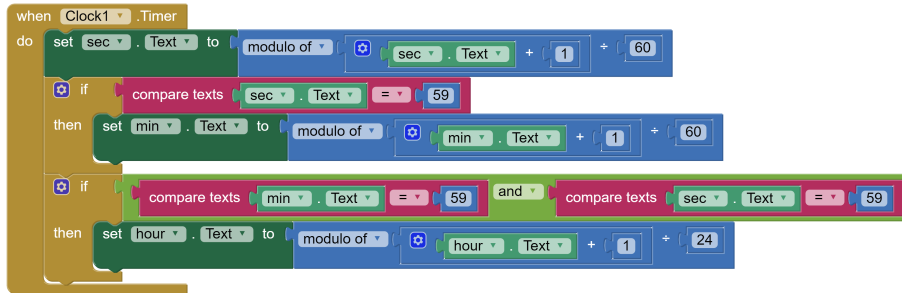


Figure 5: Session Screen block code

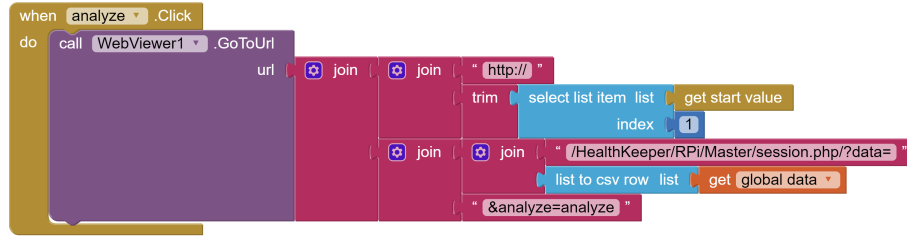


Figure 6: Session Screen block code

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.