Project 2: HTML GUI Development

Kaylin Punotai

4 March 2021

Part 1. Implementation/Assumption Notes

- Python code and HTML GUI both connect to Tornado webserver to initiate communication
- HTML GUI handles all user inputs, including getting sensor readings, requesting statistics, and toggling temperature units
- Python receives GUI requests and returns data to the GUI
- GUI displays alarms when temp or humidity exceeds user-defined alarm values and when server connection is broken

- Assumptions:
- Temperature range = [-20, 100] F
- Humidity range = [0, 100] %
- PseudoSensor code (from Week 1) randomizes data within these ranges

Part 2. Code: pseudoSensor.py

This code is from Week 1 and generates random humidity and temperature readings

```python
import random

class PseudoSensor:

    h_range = [0, 20, 20, 40, 40, 60, 60, 80, 80, 90, 70, 70, 50, 50, 30, 30, 10, 10]

    t_range = [-20, -10, 0, 10, 30, 50, 70, 80, 90, 80, 60, 40, 20, 10, 0, -10]

    h_range_index = 0

    t_range_index = 0

    humVal = 0

    tempVal = 0


    def __init__(self):

        self.humVal = self.h_range[self.h_range_index]

        self.tempVal = self.t_range[self.t_range_index]


    def generate_values(self):

        self.humVal = self.h_range[self.h_range_index] + random.uniform(0, 10);

        self.tempVal = self.t_range[self.t_range_index] + random.uniform(0, 10);

        self.h_range_index += 1

        if self.h_range_index > len(self.h_range) - 1:

            self.h_range_index = 0

        self.t_range_index += 1

        if self.t_range_index > len(self.t_range) - 1:

            self.t_range_index = 0

        return self.humVal, self.tempVal

```

Each index has a starter magnitude

Generated values are randomized based on the starter magnitude at each index

Part 2. Code: TempHumServer.py

This is the python code that opens the Tornado webserver connection and generates sensor readings

TempHumServer.py

```
1   import sys, time
2   from datetime import datetime
3
4   # Sensor readings from Week 1
5   from psuedoSensor import PseudoSensor
6   ps = PseudoSensor()
7
8   # Tornado webserver
9   import tornado.httpserver
10  import tornado.websocket
11  import tornado.ioloop
12  import tornado.web
13  import socket
14
15  class WSHandler(tornado.websocket.WebSocketHandler):
16      # When HTML is opened
17      def open(self):
18          print "Connected"
19
20      # When HTML is closed
21      def on_close(self):
22          print "Disconnected"
23
24      def check_origin(self, origin):
25          return True
26
```

Server connections/disconnections are written to the console

```python
    # Communication between HTML and python code
    def on_message(self, message):
        msg = str(message)
        returnMsg = ""

        if msg.startswith("readOne"):       # eg: readOne
            returnMsg = "read@" + self.readSensor(1)  # eg: time0,t0,h0;

        elif msg.startswith("readTen"):     # eg: readTen
            returnMsg = "read@" + self.readSensor(10) # eg: time0,t0,h0;time1,t1,h1;time2,t2,h2;

        elif msg.startswith("getStats"):    # eg: getStats:t0,h0;t1,h0;t2,h2;
            splitMsg = msg.split(":")
            stats = self.getStats(splitMsg[1])
            returnMsg = "getStats@" + stats  # eg: t0,t1,t2,h0,h1,h2

        elif msg.startswith("changeUnits"): # eg: changeUnits:tempRead,t0;tempMax,t1;tempMin,t2;:F
            units = msg[-1]                 # eg: F or C
            convTemps = ""
            splitMsg = msg.split(":")[1]
            splitMsg2 = splitMsg.split(";")
            for tempPair in splitMsg2:
                if (len(tempPair) > 0):
                    tempPairsplit = tempPair.split(",")     # eg: tempRead,t0
                    tconv = self.convertTemp(tempPairsplit[1], units)
                    convTemps = convTemps + tempPairsplit[0] + "," + tconv + ";"
            returnMsg = "changeUnits@" + convTemps      # eg: tempRead,t0;tempMax,t1;tempMin,t2;

        else:
            returnMsg = "error"

        # Return message is sent to HTML
        self.write_message(returnMsg)
```

The python code and HTML GUI communicate through messages

Each button on the GUI has python actions, which return a message that will be used to update the GUI

If there are any communication issues, errors will be raised

readSensor calls the pseudoSensor to generate a random temperature and humidity

```
61      # Get data from random generator and return readings
62      def readSensor(self, iterations):
63          readings = ""
64          for i in range(iterations):
65              h,t = ps.generate_values()
66              currentTime = datetime.now()
67              readings = readings + "%s,%s,%s;" % (str(currentTime), str(t), str(h))
68              # 1 second delay between readings
69              time.sleep(1)
70          return readings
71
```

```
72      # Calculate max, min, and average temps and humidities
73      def getStats(self, msg):
74          temp = 0.0
75          hum = 0.0
76          tempSum = 0.0
77          humSum = 0.0
78          tempMax = -20.0
79          humMax = 0.0
80          tempMin = 100.0
81          humMin = 100.0
82          numReadings = 0
83          returnMsg = ""
84
85          splitMsg = msg.split(";")   # ex: t0,h0;t1,h0;t2,h2;
86          for reading in splitMsg:
87              if len(reading) > 0:
88                  # temperature stats
89                  tempstr = reading.split(",")[0]
90                  if tempstr.startswith("-"):
91                      tempstr1 = tempstr.strip("-")
92                      temp = -1*float(tempstr1)
93                  elif len(tempstr) > 0:
94                      temp = float(tempstr)
95                  else:
96                      pass
97                  # Check max
98                  if temp > tempMax:
99                      tempMax = temp
100                 # Check min
101                 if temp < tempMin:
102                     tempMin = temp
103                 # Add for average
104                 tempSum += temp
```

getStats returns calculated stats based on the dataset sent by the GUI

Getting temp stats

```
105
106             # Humidity stats
107             humstr = reading.split(",")[1]
108             hum = float(humstr)
109             # Check max
110             if hum > humMax:
111                 humMax = hum
112             # Check min
113             if hum < humMin:
114                 humMin = hum
115             # Add for average
116             humSum += hum
117             numReadings += 1
118
119         avgTemp = tempSum/numReadings
120         avgHum = humSum/numReadings
121         return "%s,%s,%s,%s,%s,%s" % (str(tempMax), str(tempMin), str(avgTemp), str(humMax), str(hu
```

Getting humidity stats

At the end, averages are calculated based on the sums and number of readings

Then all stats values are returned

```
122
123     # Calculate conversion between Fahrenheit and Celsius
124     def convertTemp(self, temp, units):
125         if temp.startswith("-"):
126             tempstr = temp.strip("-")
127             t = -1*float(tempstr)
128         elif len(temp) > 0:
129             t = float(temp)
130         else:
131             t = 1000
132
133         if t == 1000:
134             tconv = ""
135         else:
136             if units == "F":
137                 tconv = (9*t)/5+32
138             elif units == "C":
139                 tconv = (t-32)*5/9
140
141         return str(tconv)
```

convertTemp changes any temperature sent to python to F or C

If the temperature sent to the function is N/A (eg. If it is blank), then I temporarily set it to 1000

So, any value that is at 1000 is kept as blank when it is returned to prevent any errors

```
143  # Application initializer
144  application = tornado.web.Application([
145      (r'/ws', WSHandler),
146  ])
147
148  # Tornado server connection initializer
149  if __name__ == "__main__":
150      http_server = tornado.httpserver.HTTPServer(application)
151      http_server.listen(8888)
152      myIP = socket.gethostbyname(socket.gethostname())
153      print '*** Websocket Server Started at %s***' % myIP
154      tornado.ioloop.IOLoop.instance().start()
```

The application and server are initialize with this code

Part 2. Code: TempHumUI.html

This HTML is code for inputs and display

TempHumUI.html > ...

```
1   <!doctype html>
2   <html>
3       <head>
4           <title>Pseudo Temperature/Humidity Sensor</title>
5           <meta charset="utf-8" />
6           <style type="text/css">
7               body {
8                   text-align: center;
9                   min-width: 500px;
10              }
11          </style>
12          <script src="http://code.jquery.com/jquery.min.js"></script>
13          <script>
14              // At initialization, the UI hides all elements except the Tornado connect button
15              $(document).ready(function () {
16                  $("#disconnectButton").hide();
17                  $("div#settings").hide();
18                  $("div#readings").hide();
19                  $("div#stats").hide();
20                  $("div#history").hide();
21
22                  var ws;
23                  var tempRead;
```

CSS was kept default

When the HTML is first initialized, all elements are hidden until it is connected to the server

```
25          // Connect button onClick event, which opens the websocket
26          $("#connectButton").click(function(evt) {
27              evt.preventDefault();
28
29              // create websocket instance
30              ws = new WebSocket("ws://localhost:8888/ws");
31
32              // websocket close event
33              ws.onclose = function(evt) {
34                  $("#connectionStatus").text("Disconnected");
35                  $("#connectionStatus").css("color", "red");
36                  $("#disconnectButton").hide();
37                  $("#connectButton").show();
38                  $("div#settings").hide();
39                  $("div#readings").hide();
40                  $("div#stats").hide();
41                  $("div#history").hide();
42              };
43
44              // websocket open event
45              ws.onopen = function(evt) {
46                  $("#connectionStatus").text("Connected");
47                  $("#connectionStatus").css("color", "green");
48                  $("#connectButton").hide();
49                  $("#disconnectButton").show();
50                  $("div#settings").show();
51                  $("div#readings").show();
52                  $("div#stats").show();
53                  $("div#history").show();
54              };
```

When the connect button is clicked, a new websocket is created

When the websocket is closed, the GUI reverts back to its disconnected state

When the websocket is opened, all elements are displayed and the connect button is replaced by the disconnect button

```
55
56              // Incoming messages from python
57              ws.onmessage = function(evt) {
58                  var msg = String(evt.data);
59                  var msgSplit = msg.split("@"); // eg. read@xxx or getStats@xxx
60                  var action = msgSplit[0];
61                  var data = msgSplit[1];
62
63                  // Actions vary based on the message obtained from python
64                  switch(action) {
65                      case "read":
66                          addData(data);
67                          break;
68                      case "getStats":
69                          updateStats(data);
70                          break;
71                      case "changeUnits":
72                          updateTemps(data);
73                          break;
74                      case "error":
75                          alert("An error occurred with the Python code connection");
76                          break;
77                      default:
78                          alert("An unknown error occurred with the Python code connection");
79                          break;
80                  }
81
82              };
83
84              // Alert when server is not available
85              ws.onerror = function(evt) {
86                  alert("Tornado webserver is not available");
87              };
88          });
```

The onMessage event corresponds to messages sent from Python to HTML

The data retrieved from Python are manipulated further by the HTML depending on the original request

Communication errors are caught and will show alarms

When the server is not available, an alarm will show

```
90          // Disconnect click event: UI refreshes
91          $("#disconnectButton").click(function(evt) {
92              evt.preventDefault();
93              location.reload();
94          });
95
96          // Close event: window closes
97          $("#closeButton").click(function(evt) {
98              evt.preventDefault();
99              close();
100         });
101
102         // Read 1 event: one reading is obtained from pseudosensor
103         $("#readOne").click(function(evt) {
104             ws.send("readOne");
105         });
106
107         // Read 10 event: ten readings are obtained from pseudosensor
108         $("#readTen").click(function(evt) {
109             ws.send("readTen");
110         });
111
112         // getStats event: stats are calculated from last 10 entries
113         $("#getStats").click(function(evt) {
114             var datalist = getData();
115             ws.send("getStats:" + datalist);
116         });
117
```

The disconnect button disconnects from the server and refreshes the GUI

The close button closes the browser window, which also disconnects from the server

The Read 1 button retrieves 1 temp/humidity reading

Read 10 button retrieves 10 temp/humidity readings

The Get statistics button calculates statistics for temp and hum

```
118         // Fradio change event: all temperatures are converted to Fahrenheit
119         $("#Fradio").change(function(evt) {
120             var units = "°F";
121             $("#tempUnitsLabel1").text(units);
122             $("#tempUnitsLabel2").text(units);
123             $("#tempUnitsLabel3").text(units);
124             $("#tempUnitsLabel4").text(units);
125             $("#tempUnitsLabel5").text(units);
126             var dataTable = document.getElementById("dataTable");
127             var numRows = dataTable.rows.length;
128             var i;
129             for (i = 1; i < numRows; i++) {
130                 dataTable.rows[i].cells[2].innerHTML = units;
131             }
132             var datalist = getTempFields();
133             ws.send("changeUnits:" + datalist + ":F");
134         });
135
136         // Cradio change event: all temperatures are converted to Celsius
137         $("#Cradio").change(function(evt) {
138             var units = "°C";
139             $("#tempUnitsLabel1").text(units);
140             $("#tempUnitsLabel2").text(units);
141             $("#tempUnitsLabel3").text(units);
142             $("#tempUnitsLabel4").text(units);
143             $("#tempUnitsLabel5").text(units);
144             var dataTable = document.getElementById("dataTable");
145             var numRows = dataTable.rows.length;
146             var i;
147             for (i = 1; i < numRows; i++) {
148                 dataTable.rows[i].cells[2].innerHTML = units;
149             }
150             var datalist = getTempFields();
151             ws.send("changeUnits:" + datalist + ":C");
152         });
153
```

There are radio buttons to toggle between Fahrenheit and Celsius on all temperature fields, including the reading, alarm imput, statistics, and data table temperatures

Unit labels also change

```
154         // Python sends readings as raw data, which are then split and logged in UI data ta
155         function addData(dataraw)       // datastr eg: time0,t0,h0;time1,t1,h1;time2,t2,h2;
156         {
157             var datastr = String(dataraw);
158             var dataSplit = datastr.split(";");
159 
160             var i;
161             // Each reading is added as a new row entry in data table
162             for (i = 0; i < (dataSplit.length-1); i++) {
163                 var entrySplit = dataSplit[i].split(",");
164                 var time = entrySplit[0];
165                 var temp = entrySplit[1];
166                 var hum = entrySplit[2];
167                 var dataTable = document.getElementById("dataTable");
168                 var newRow = dataTable.insertRow(-1);
169                 var timeCell = newRow.insertCell(0);
170                 var tempCell = newRow.insertCell(1);
171                 var tempUnitsCell = newRow.insertCell(2);
172                 timeCell.innerHTML = time;
173                 var humCell = newRow.insertCell(3);
174                 humCell.innerHTML = hum;
175                 var humUnitsCell = newRow.insertCell(4);
176                 humUnitsCell.innerHTML = "%";
177 
178                 // Convert temp to Celsius if checked, since python data is Fahrenheit by d
179                 var tempUnits = document.getElementById("Fradio");
180                 if (tempUnits.checked) {
181                     tempUnitsCell.innerHTML = "°F";
182                 }
183                 else {
184                     tempUnitsCell.innerHTML = "°C";
185                     temp = (temp-32)*5/9;
186                 }
187                 tempCell.innerHTML = temp;
188 
189                 // Reading displays only show final readings in a set
190                 document.getElementById("tempRead").value = temp;
191                 document.getElementById("humRead").value = hum;
192             }
```

addData takes the readings send from Python and places them in the data table

Python data is in Fahrenheit by default, so if the Celsius radio is checked, the data is converted to C

The last reading is displayed

The last reading is checked for alarms

If the last temp or humidity reading exceeds alarm values, an alarm will pop up and the reading will turn red

```
194             // Check for alarms
195             var temp = document.getElementById("tempRead").value;
196             var hum = document.getElementById("humRead").value;
197             var tempAlarm = document.getElementById("tempAlarm").value;
198             var humAlarm = document.getElementById("humAlarm").value;
199             // If any readings exceed alarms, an alert will show and the box will turn red
200             if (temp > tempAlarm) {
201                 alert("Temperature Alarm!");
202                 document.getElementById("tempRead").style.backgroundColor = "red";
203             }
204             else {
205                 document.getElementById("tempRead").style.backgroundColor = "transparent";
206             }
207             if (hum > humAlarm) {
208                 alert("Humidity Alarm!");
209                 document.getElementById("humRead").style.backgroundColor = "red";
210             }
211             else {
212                 document.getElementById("humRead").style.backgroundColor = "transparent";
213             }
214         }
215
```

```
216         // Retrieve last 10 data pairs from the data table
217         function getData() {
218             var i;
219             var temp;
220             var hum;
221             var datalist = "";
222             var dataTable = document.getElementById("dataTable");
223             var numRows = dataTable.rows.length;
224             // If there are fewer than 10 entries, all entries will be retrieved
225             if (numRows < 11) {
226                 for (i = 1; i < numRows; i++) {
227                     temp = dataTable.rows[i].cells[1].innerHTML;
228                     hum = dataTable.rows[i].cells[3].innerHTML;
229                     datalist = datalist + temp + "," + hum + ";";
230                 }
231             }
232             // if there are more than 10 entries, only the last 10 entries will be retrieve
233             else {
234                 for (i = 0; i < 10; i++) {
235                     row = numRows - i - 1;
236                     temp = dataTable.rows[row].cells[1].innerHTML;
237                     hum = dataTable.rows[row].cells[3].innerHTML;
238                     datalist = datalist + temp + "," + hum + ";";
239                 }
240             }
241             return datalist;
242         }
```

getData retrieves the last 10 data pairs, which will be sent to Python to calculate stats

if <10 entries, all entries will be retrieved

If >10 entries, only the last 10 entries will be retrieved

```
244    // After python calculates stats, the stats table is updated
245    function updateStats(statData) {
246        var datastr = String(statData);
247        var dataSplit = datastr.split(",");
248        var tempMax = dataSplit[0];
249        var tempMin = dataSplit[1];
250        var tempAvg = dataSplit[2];
251        var humMax = dataSplit[3];
252        var humMin = dataSplit[4];
253        var humAvg = dataSplit[5];
254        document.getElementById("tempMax").innerHTML = tempMax;
255        document.getElementById("tempMin").innerHTML = tempMin;
256        document.getElementById("tempAvg").innerHTML = tempAvg;
257        document.getElementById("humMax").innerHTML = humMax;
258        document.getElementById("humMin").innerHTML = humMin;
259        document.getElementById("humAvg").innerHTML = humAvg;
260    }
261
```

updateStats takes the stats data from Python and updates the GUI stats table

```
261
262         // Get all temperature fields that will be converted to different units
263         function getTempFields() {
264             var datalist = "";
265             var tempAlarm = document.getElementById("tempAlarm").value;
266             var tempRead = document.getElementById("tempRead").value;
267             var tempMax = document.getElementById("tempMax").innerHTML;
268             var tempMin = document.getElementById("tempMin").innerHTML;
269             var tempAvg = document.getElementById("tempAvg").innerHTML;
270             // If these values exist, they will be added to the datalist
271             if (tempAlarm.length > 0) {
272                 datalist = datalist + "tempAlarm," + tempAlarm + ";";
273             }
274             if (tempRead.length > 0) {
275                 datalist = datalist + "tempRead," + tempRead + ";";
276             }
277             if (tempMax.length > 0) {
278                 datalist = datalist + "tempMax," + tempMax + ";";
279             }
280             if (tempMin.length > 0) {
281                 datalist = datalist + "tempMin," + tempMin + ";";
282             }
283             if (tempAvg.length > 0) {
284                 datalist = datalist + "tempAvg," + tempAvg + ";";
285             }
286
287             var dataTable = document.getElementById("dataTable");
288             var numRows = dataTable.rows.length;
289             var i;
290             for (i = 1; i < numRows; i++) {
291                 var temp = dataTable.rows[i].cells[1].innerHTML;
292                 datalist = datalist + i + "," + temp + ";";
293             }
294
295             return datalist;
296         }
297
```

getTempFields gets all temperature element values, which will be sent to Python to convert units

```
298         // After python converts the temperatures, they are updated in the UI
299         function updateTemps(tempData) { // eg: tempRead,t0;tempMax,t1;tempMin,t2;
300             var datastr = String(tempData);
301             var datalist = datastr.split(";");
302             var i;
303             for (i = 0; i < datalist.length; i++) {
304                 var dataSplit = datalist[i].split(",");
305                 switch (dataSplit[0]) {
306                     case "tempAlarm":
307                         document.getElementById("tempAlarm").value = dataSplit[1];
308                         break;
309                     case "tempRead":
310                         document.getElementById("tempRead").value = dataSplit[1];
311                         break;
312                     case "tempMax":
313                         document.getElementById("tempMax").innerHTML = dataSplit[1];
314                         break;
315                     case "tempMin":
316                         document.getElementById("tempMin").innerHTML = dataSplit[1];
317                         break;
318                     case "tempAvg":
319                         document.getElementById("tempAvg").innerHTML = dataSplit[1];
320                         break;
321                     default:
322                         if (dataSplit[0].length > 0) {
323                             var row = parseInt(dataSplit[0]);
324                             document.getElementById("dataTable").rows[row].cells[1].innerHT
325                         }
326                         break;
327                 }
```

updateTemps takes the converted temps from Python and updates the corresponding temp elements

Beginning of HTML

```
334     <body>
335         <h1>Pseudo Temperature/Humidity Sensor</h1>
336         <p>By Kaylin Punotai</p>
337         <div id="tornadoConnect">
338             <!-- Connection status is red when disconnected and green when connected -->
339             <label id="connectionStatus" style="color: red">Disconnected</label>
340             <br/>
341             <input type="submit" id="connectButton" value="Connect"/>
342             <input type="submit" id="disconnectButton" value="Disconnect"/>
343             <input type="submit" id="closeButton" value="Close"/>
344         </div>
```

Connection status label changes color and text when connected or disconnected to the server

Connect, disconnect, and close buttons will show at the top of the UI

```
345         <div id="settings"> <!-- User-defined settings: alarms and temp units -->
346             <br/>
347             <table>
348                 <tr>
349                     <th>Set Alarms:</th>
350                     <th>Change Units:</th>
351                 </tr>
352                 <tr>
353                     <td>
354                         <label for="tempAlarm">Temperature:</label>
355                         <input type="text" id="tempAlarm" value="50"/>
356                         <label for="tempAlarm" id="tempUnitsLabel1">°F</label>
357                     </td>
358                     <td>
359                         <input type="radio" name="unitRadios" id="Fradio" value="F" checked/>
360                         <label for="Fradio">°F</label>
361                     </td>
362                 </tr>
363                 <tr>
364                     <td>
365                         <label for="humAlarm">Humidity:</label>
366                         <input type="text" id="humAlarm" value="20"/>
367                         <label for="humAlarm">%</label>
368                     </td>
369                     <td>
370                         <input type="radio" name="unitRadios" id="Cradio" value="C"/>
371                         <label for="Cradio">°C</label>
372                     </td>
373                 </tr>
374             </table>
375             <br/>
376             <input type="button" id="readOne" value="Read 1"/>
377             <input type="button" id="readTen" value="Read 10"/>
378             <input type="button" id="getStats" value="Get Statistics"/>
379             <br/>
380         </div>
```

The “settings” section keeps the user-defined items, which are the alarm settings and temperature units

Buttons for Read 1, Read 10, and Get Statistics

```
381     <div id="readings"> <!-- Displays the last temp and humidity readings -->
382         <br/>
383         <label>Sensor Reading</label>
384         <br/>
385         <label for="tempRead">Temperature:</label>
386         <input type="text" id="tempRead" readonly/>
387         <label for="tempRead" id="tempUnitsLabel2">°F</label>
388         <br/>
389         <label for="humRead">Humidity:</label>
390         <input type="text" id="humRead" readonly/>
391         <label for="humRead">%</label>
392         <br/>
393     </div>
```

Readings displays the last temp and hum readings as readonly textboxes

```
394     <div id="stats"> <!-- Displays max, min, and avg for temp and humidity -->
395         <br/>
396         <table id="statsTable">
397             <caption>Statistics</caption>
398             <tr>
399                 <th></th>
400                 <th>Temperature</th>
401                 <th></th>
402                 <th>Humidity</th>
403                 <th></th>
404             </tr>
405             <tr>
406                 <td>Max</td>
407                 <td id="tempMax"></td>
408                 <td id="tempUnitsLabel3">°F</td>
409                 <td id="humMax"></td>
410                 <td>%</td>
411             </tr>
412             <tr>
413                 <td>Min</td>
414                 <td id="tempMin"></td>
415                 <td id="tempUnitsLabel4">°F</td>
416                 <td id="humMin"></td>
417                 <td>%</td>
418             </tr>
419             <tr>
420                 <td>Average</td>
421                 <td id="tempAvg"></td>
422                 <td id="tempUnitsLabel5">°F</td>
423                 <td id="humAvg"></td>
424                 <td>%</td>
425             </tr>
426         </table>
427         <br/>
428     </div>
```

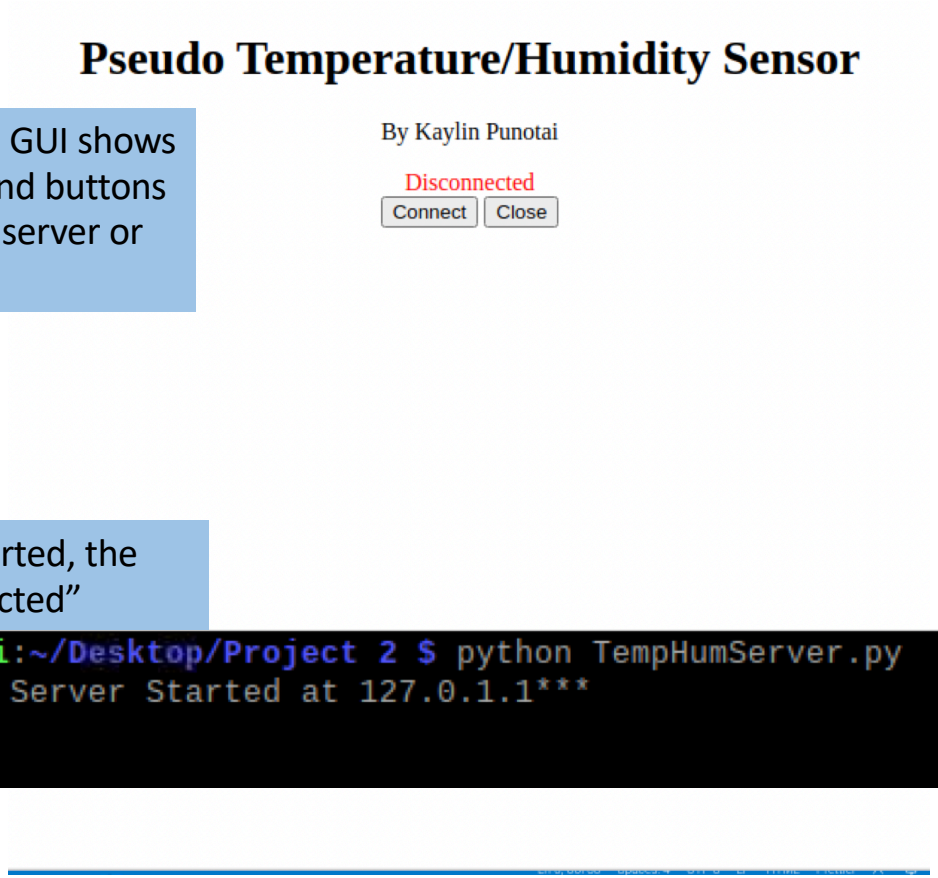
"stats" contains a table that displays the statistics of the last 10 entries

```
429         <div id="history"> <!-- data table of all sensor readings -->
430             <br/>
431             <table id="dataTable">
432                 <caption>History</caption>
433                 <tr>
434                     <th>Time</th>
435                     <th>Temperature</th>
436                     <th></th>
437                     <th>Humidity</th>
438                     <th></th>
439                 </tr>
440             </table>
441             <br/>
442         </div>
443     </body>
444 </html>
```

This data table holds every sensor reading obtained in a session

Part 3. HTML UI Demonstration

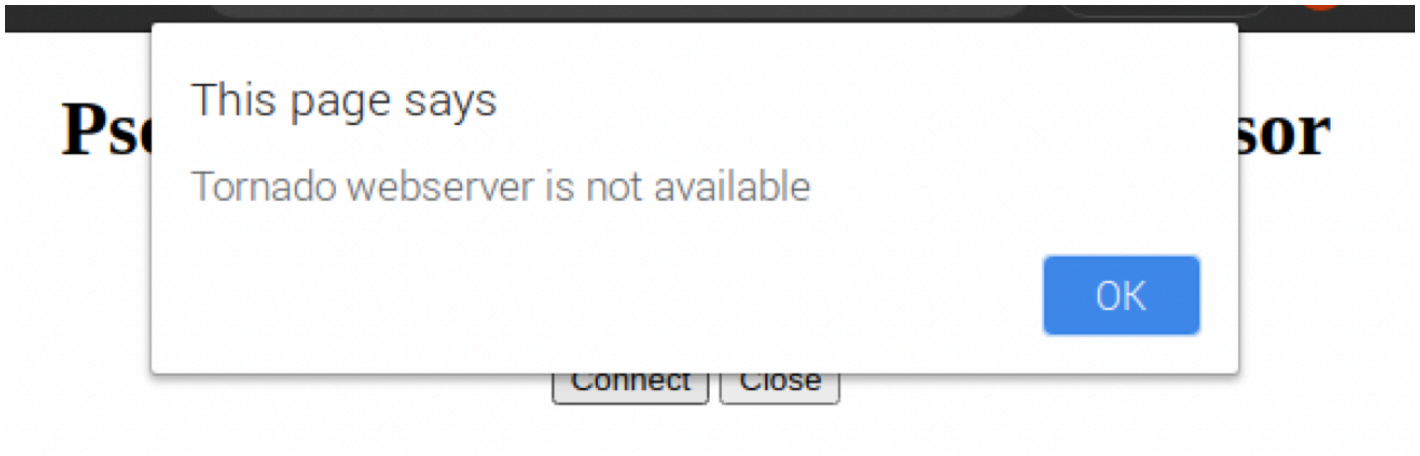
Upon startup, the GUI shows “Disconnected” and buttons to connect to the server or close the window



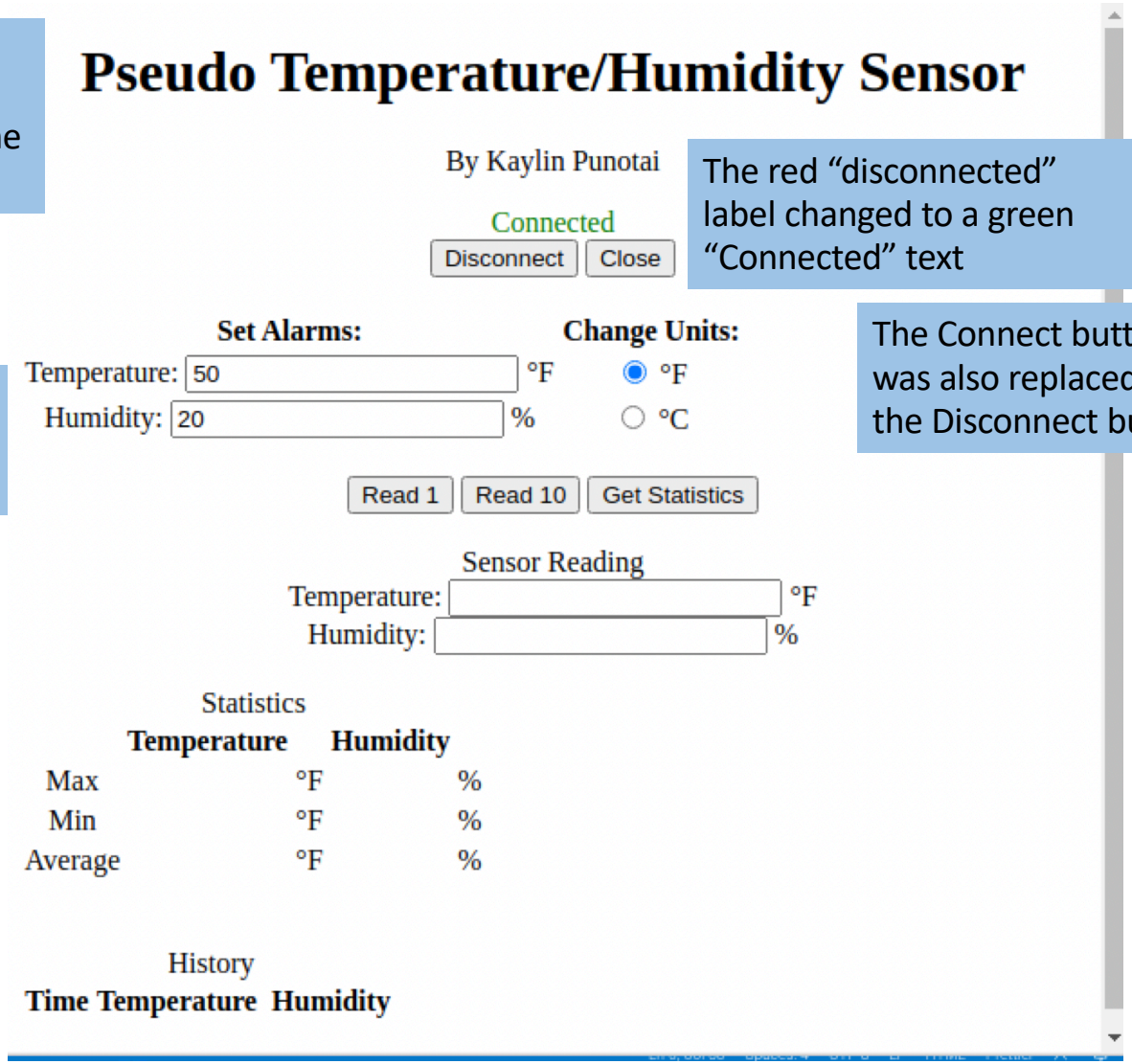
When the server is started, the console prints “Connected”

```
pi@raspberrypi:~/Desktop/Project 2 $ python TempHumServer.py
*** Websocket Server Started at 127.0.1.1***
Connected
```

If Python has not been run to activate the Server and the HTML tries to connect to the server, an error will pop up



When the GUI successfully connects to the server, the rest of the elements are shown



The red “disconnected” label changed to a green “Connected” text

The Connect button was also replaced by the Disconnect button

Temperature and humidity alarm inputs have default settings

Clicking ”Read 1”

Pseudo Temperature/Humidity Sensor

By Kaylin Punotai

Connected

Disconnect Close

Set Alarms: **Change Units:**

Temperature: 50 °F °F

Humidity: 60 % °C

Read 1 Read 10 Get Statistics

Readings are updated

Sensor Reading

Temperature: -19.7090862981 °F

Humidity: 4.18079478081 %

Statistics

	Temperature	Humidity
Max	°F	%
Min	°F	%
Average	°F	%

History

Time	Temperature	Humidity
2021-03-03 23:33:14.776256	-19.7090862981 °F	4.18079478081 %

The history table shows timestamp and readings

Clicking “Get Statistics”:

Pseudo Temperature/Humidity Sensor

By Kaylin Punotai

Connected

Disconnect Close

Set Alarms: **Change Units:**

Temperature: 50 °F °F

Humidity: 60 % °C

Read 1 Read 10 Get Statistics

Sensor Reading

Temperature: -19.7090862981 °F

Humidity: 4.18079478081 %

Statistics

	Temperature	Humidity
Max	-19.7090862981 °F	4.18079478081 %
Min	-19.7090862981 °F	4.18079478081 %
Average	-19.7090862981 °F	4.18079478081 %

Statistics table is updated

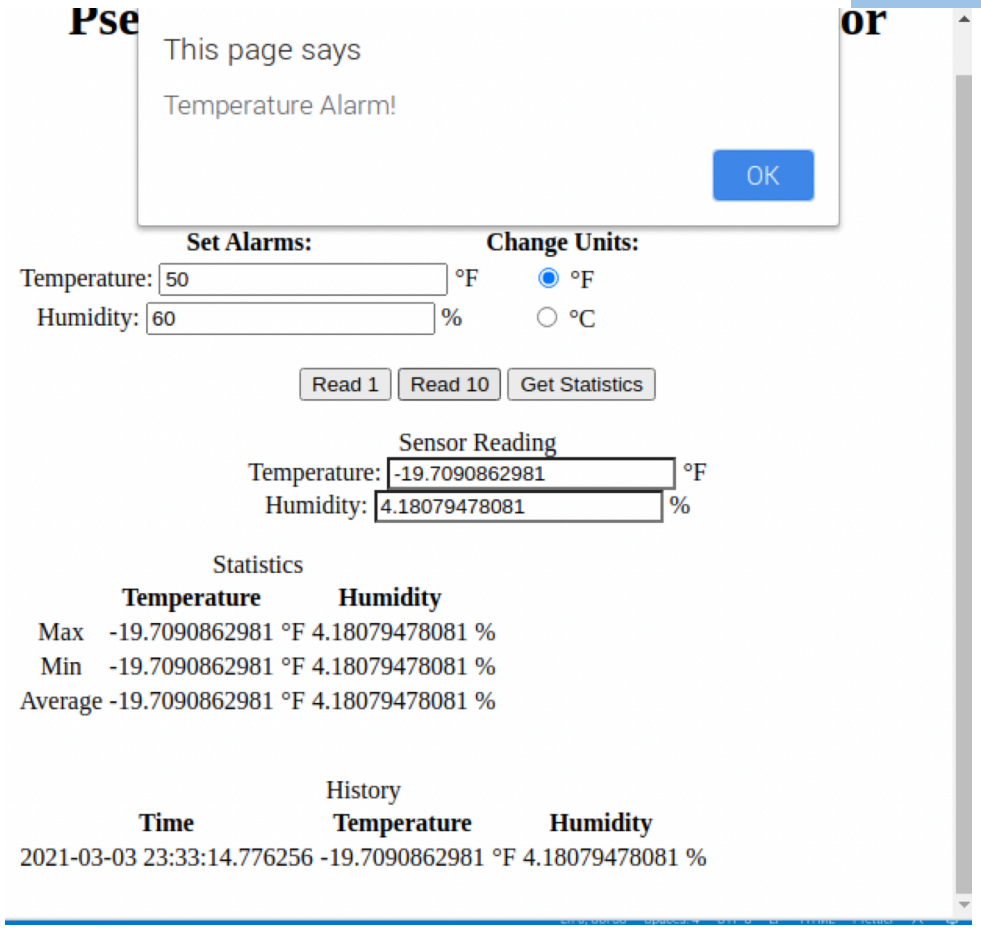
Makes sense that they are all equal to the single reading

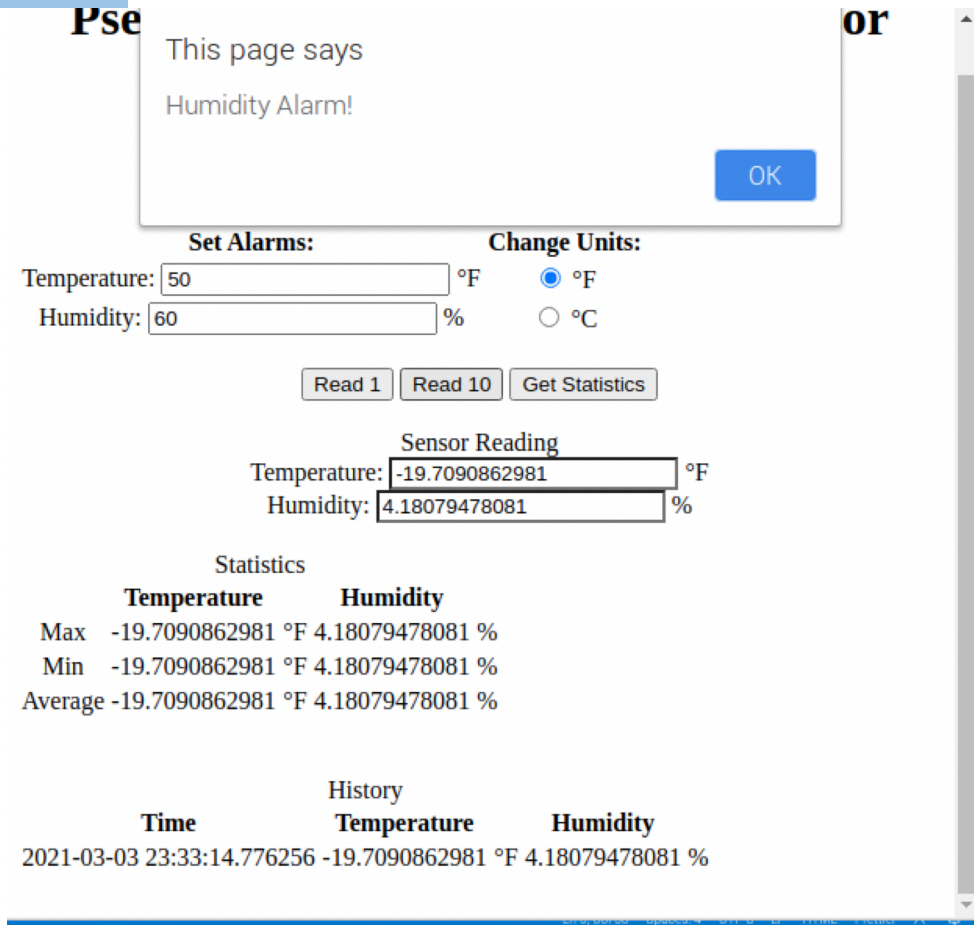
History

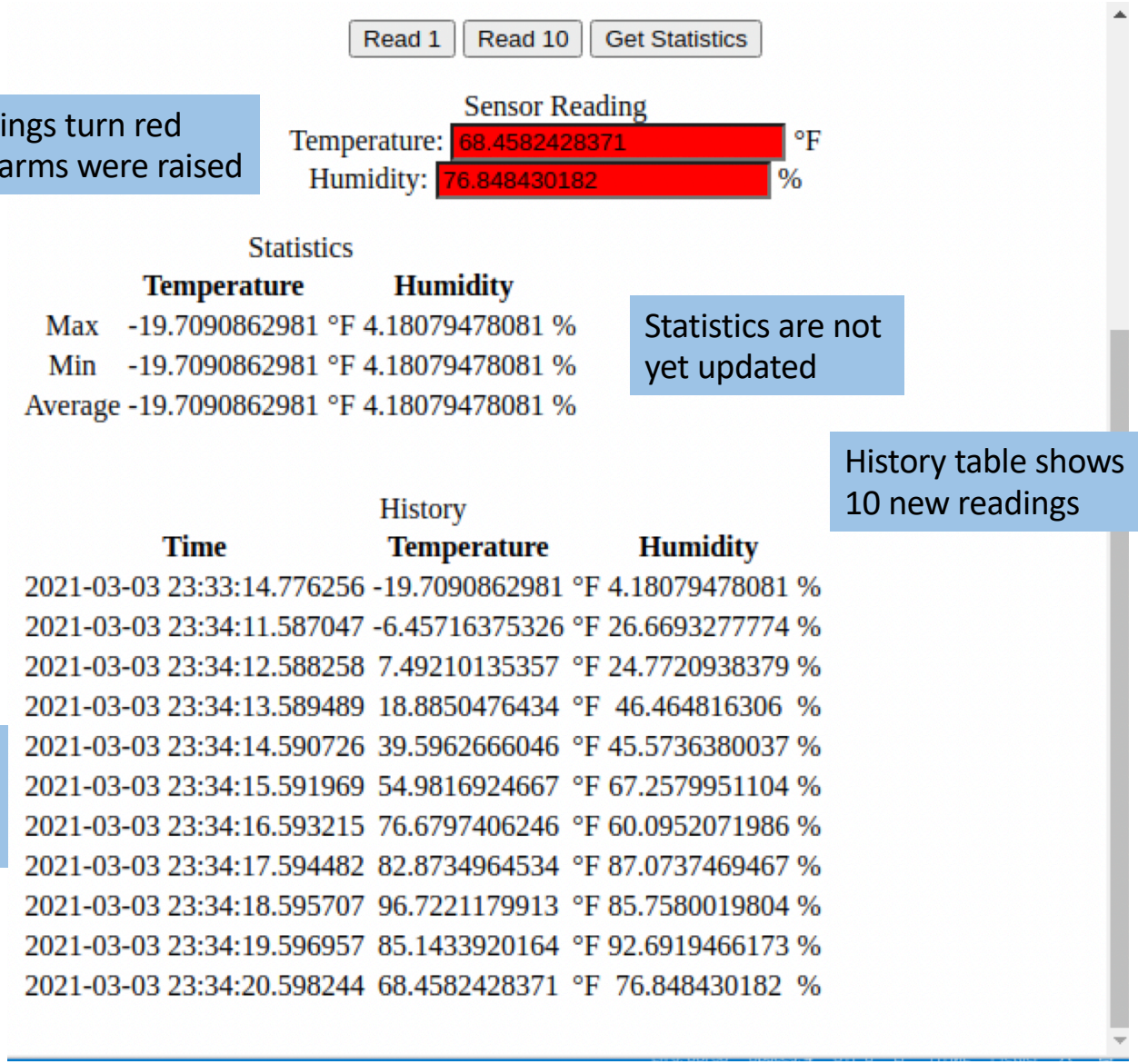
Time	Temperature	Humidity
2021-03-03 23:33:14.776256	-19.7090862981 °F	4.18079478081 %

Clicking “Read 10”:

Temp and Hum alarms raised







The sensor readings turn red because their alarms were raised

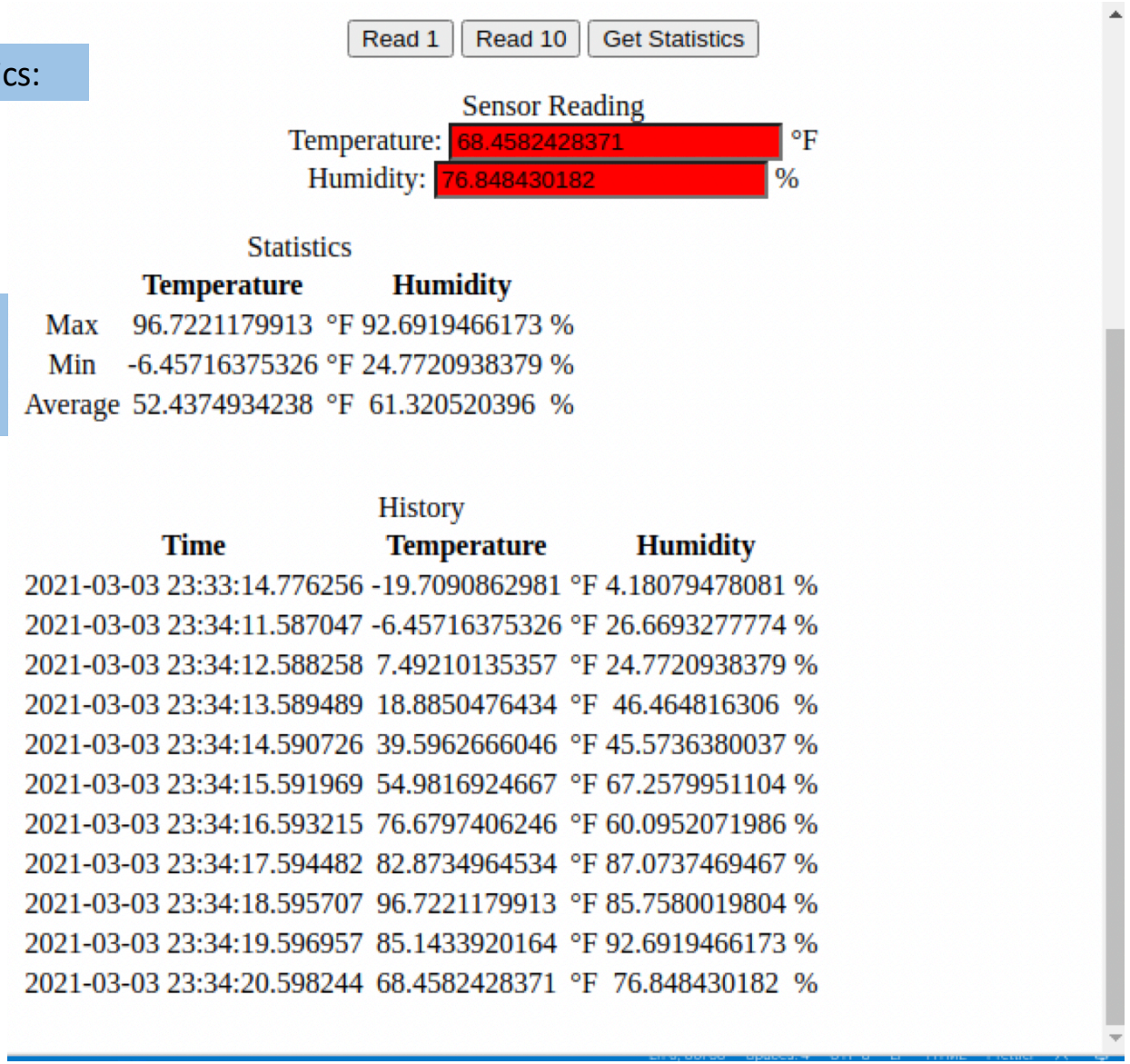
Statistics are not yet updated

History table shows 10 new readings

For ”Read 10”, there is a 1 sec delay per reading

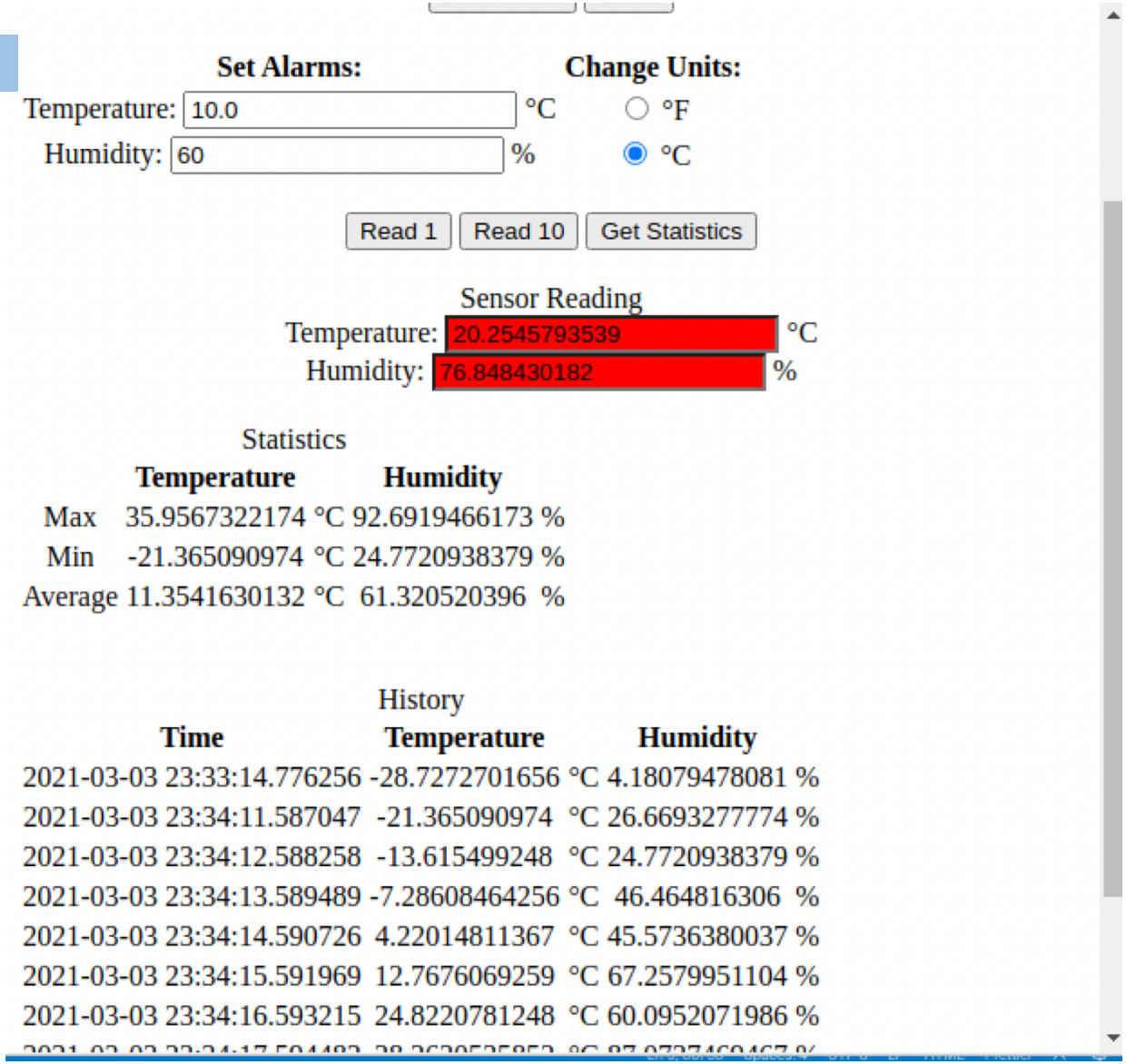
Clicking “Get Statistics:

Stats are now updated for the last 10 entries



Clicking C radio button:

All temperature fields are converted to Celsius



Clicking “Disconnect”:

Pseudo Temperature/Humidity Sensor

By Kaylin Punotai

Disconnected

Connect Close

The HTML is disconnected from the server, and the page goes back to its original blank state

pi@raspberrypi: ~/Desktop/Project 2

File Edit Tabs Help

```
pi@raspberrypi:~/Desktop/Project 2 $ python TempHumServer.py
*** Websocket Server Started at 127.0.1.1***
Connected
Disconnected
```

The console prints the disconnection