INFT3970 Major Project Final Report
Distributed Monitoring System using ESP-12E

Team Encore
Thursday 10:00AM - 10:30AM
Jay Rovacsek c3146220@uon.edu.au
Dean Morton c3252227@uon.edu.au
Josh Brown c3283797@uon.edu.au
Jacob Litherland c3263482@uon.edu.au
Lee Marron c3263482@uon.edu.au
Edward Lonsdale c3252144@uon.edu.au

October 15, 2018

Contents

1 Executive Summary **2**

2 Introduction **3**
2.1 Project Objectives . . . 3
2.2 Business Summary . . . 3

3 Proposed Solution **4**
3.1 Outline . . . 4
3.2 Technical Requirements . . . 4
3.3 Limitations or Constraints . . . 4

4 Technical Documentation **5**
4.1 Database Documentation . . . 5
4.1.1 Relational Model . . . 5
4.1.2 Data Requirements . . . 5
4.1.3 Data Dictionary . . . 5
4.1.4 Database Schema . . . 5
4.2 ESP12-E Documentation . . . 6
4.2.1 Temperature . . . 6
4.2.2 Humidity . . . 6
4.2.3 Motion . . . 6
4.2.4 Issues in Development . . . 6
4.3 Web Application Documentation . . . 7
4.3.1 API Documentation . . . 7
4.3.2 Issues in Development . . . 7
4.4 General Documentation . . . 8
4.4.1 Sequence Diagrams . . . 8
4.4.2 Network Diagram . . . 8
4.5 Install Proceedures . . . 9

5 User Documentation **10**
5.1 Installation . . . 10
5.2 Accessing Dashboards . . . 10
5.3 Account Administration . . . 10

A Appendix **12**
A.1 Golang ESP12-E Implementation . . . 12

1 Executive Summary

2 Introduction

2.1 Project Objectives

2.2 Business Summary

3 Proposed Solution

3.1 Outline

3.2 Technical Requirements

3.3 Limitations or Constraints

4 Technical Documentation

4.1 Database Documentation

4.1.1 Relational Model

4.1.2 Data Requirements

4.1.3 Data Dictionary

4.1.4 Database Schema

4.2 ESP12-E Documentation

4.2.1 Temperature

4.2.2 Humidity

4.2.3 Motion

4.2.4 Issues in Development

One of the first challenges of this project was to ensure we had a viable path to producing a result we deemed desirable, a rused prototype was created on the ESP12-E using arduino code. The code used methods causing blocks on continued execution and for one or two items being reported this was okay, however to ensure the platform had room to breathe we investigated two routes: non-blocking code using timers in arduino code, and golang utilising the Gobot framework[1].

The Gobot route seemed very promising, prewritten libraries for items such as the XC-4444[2] existed, however a mixture of team members unfamiliar with the Go langauge[3] and a number of resources pointing to the DHT-11[4] being severely undersupported[5]. in the Golang space lead us to investigate the alternative option: timers in arduino code.

A copy of the Golang Arduino Implementation can be found here.

4.3 Web Application Documentation

4.3.1 API Documentation

4.3.2 Issues in Development

4.4 General Documentation

4.4.1 Sequence Diagrams

4.4.2 Network Diagram

4.5 Install Proceedures

5 User Documentation

5.1 Installation

5.2 Accessing Dashboards

5.3 Account Administration

References

[1] Various. (2018). The gobot framework, [Online]. Available: https://gobot.io/.

[2] ——, (2018). Jaycar xc-4444 datasheet, [Online]. Available: https://www.jaycar.com.au/medias/sys_master/images/9105858396190/XC4444-dataSheetMain.pdf.

[3] ——, (2018). The golang language, [Online]. Available: https://golang.org/.

[4] ——, (). Jaycar resources; dht11 datasheet, [Online]. Available: https://www.jaycar.com.au/medias/sys_master/images/9091897786398/XC4520-dataSheetMain.zip.

[5] ——, (2018). Issues implementing dht-11 and dht-22, [Online]. Available: https://github.com/hybridgroup/gobot/issues/361.

A Appendix

A.1 Golang ESP12-E Implementation

```
1  package main
2
3  import (
4      "C"
5      "dht"
6      "fmt"
7      "net/http"
8      "time"
9
10     "gobot.io/x/gobot"
11     "gobot.io/x/gobot/drivers/gpio"
12     "gobot.io/x/gobot/platforms/firmata"
13 )
14
15 const (
16     SensorDHT11 = iota
17     SensorDHT22
18 )
19
20 var firmataAdaptor = firmata.NewTCPAdaptor("192.168.1.95:3030")
21
22 func main() {
23
24     work := func() {
25         gobot.Every(1*time.Second, func() {
26             tryFlashLED()
27         })
28     }
29
30     led := gpio.NewLedDriver(firmataAdaptor, "2")
31
32     robot := gobot.NewRobot("bot",
33         []gobot.Connection{firmataAdaptor},
34         []gobot.Device{led},
35         work,
36     )
37
38     robot.Start()
39 }
40
41 func tryFlashLED() {
42     var available = getAvailability(`http://inft3970.azurewebsites.net
       :80/api/Availability`)
43
44     fmt.Println(available)
45
46     if available {
47         flashLED(`100ms`, 10)
48         return
```

```
49      }
50
51      flashLED('500', 2)
52      return
53  }
54
55  func flashLED(milliseconds string, iterations int) {
56      led := gpio.NewLedDriver(firmataAdaptor, "2")
57
58      duration, _ := time.ParseDuration(milliseconds)
59
60      for i := 1; i <= iterations; i++ {
61          led.Toggle()
62          time.Sleep(duration)
63          led.Toggle()
64      }
65  }
66
67  func getAvailability(server string) bool {
68
69      response, err := http.Get(server)
70
71      var humidity, temperature, _ = getSensorData(SensorDHT11, 2)
72      if err != nil {
73          printError(err)
74      }
75
76      fmt.Println(fmt.Sprintf("Temperature: %f, Humidity: %f"), temperature
            , humidity)
77
78      if err != nil {
79          printError(err)
80      }
81
82      if response.StatusCode == 200 {
83          return true
84      }
85      return false
86  }
87
88  func getSensorData(stype, pin int) (humidity, temperature float32, err
        error) {
89      if stype != SensorDHT11 && stype != SensorDHT22 {
90          err = fmt.Errorf("sensor type must be either %d or %d",
                SensorDHT11, SensorDHT22)
91          return
92      }
93
94      var data [5]byte
95      data, err = dht.ReadSensor(pin)
96      if err != nil {
97          return
98      }
```

```
99
100     if stype == SensorDHT11 {
101         humidity = float32(data[0])
102         temperature = float32(data[2])
103     } else {
104         humidity = float32(int(data[0])*256+int(data[1])) / 10.0
105         temperature = float32(int(data[2]&0x7F)*256+int(data[3])) / 10.0
106         if data[2]&0x80 > 0 {
107             temperature *= -1.0
108         }
109     }
110     return
111 }
112
113 func printError(err error) {
114     fmt.Println(fmt.Sprintf("An error occured: %v", err.Error()))
115 }
```