IoT Experimental Learning
Week 3 Journal

1. **Describe the experience and what you hope to gain from participating in the experience.**
    - This week's assignment was a bit easier to get through mostly due to having all of the prerequisites in place and configured correctly. I was able to complete the assignment without issue.
        1. During this week I performed research and development in order to integrate NodeRed into the existing project
        2. I was also able to engage with other classmates through discussions to pass on information I gathered during this research by post two discussions on;
            1. Discussion Post: Week 3Primer.
            2. Discussion Post: Module 3 Gist for those having issues with this module.
        3. Provided support to other classmates (Donna Stuart) for week 3 issues

2. **Provide an overview of tasks and key activities (training, discussions, labs, assessments, etc.) in which you were engaged during the week.**

   For week 3 I accomplished the following tasks in chronological order;

❖ **Monday February 3, 2020**, completed reviewing this week's assignment for the project.
- ➢ I performed an internet search and read though the NodeRed Specification at https://nodered.org/docs/getting-started/raspberrypi
- ➢ Completed Discussion Post as Week3 Primer to share research

❖ **Wednesday February 5, 2020**, complete code assignment for Course Model 3
- ➢ Completed Node Red Flow as seen in Figure 1
    - Also refactored python code to include new python class to encapsulate the MQTT publishing.

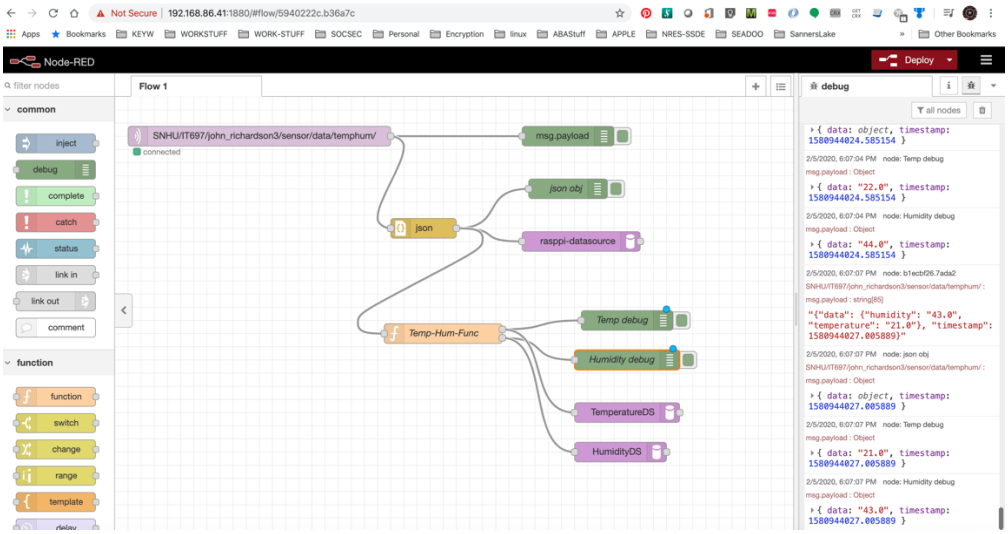

Figure 1 Completed Node Red Flow (Week 3)

- Individual areas of running program as follows;
    - NodeRed function that creates two new mqtt message payloads which splits out the original payload values for consumption by other NodeRed components or dashboards

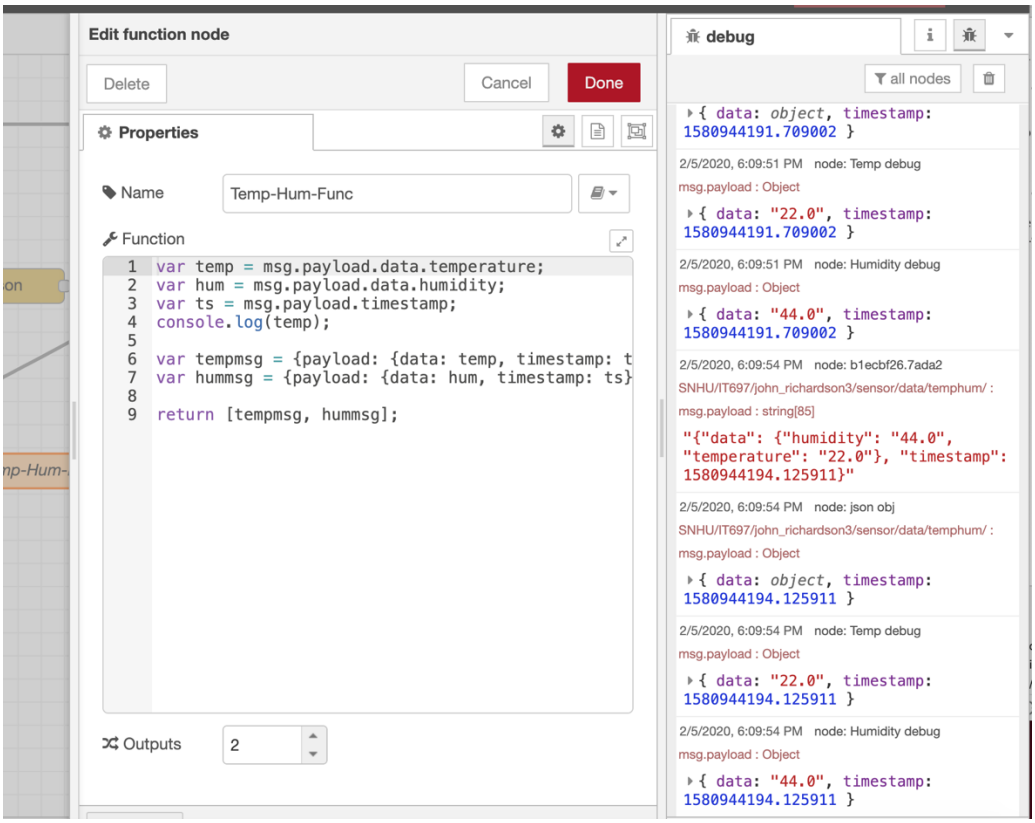

Figure 2 jasvascript contained in the NodeRed function

- Running program in terminal and NodeRed Console

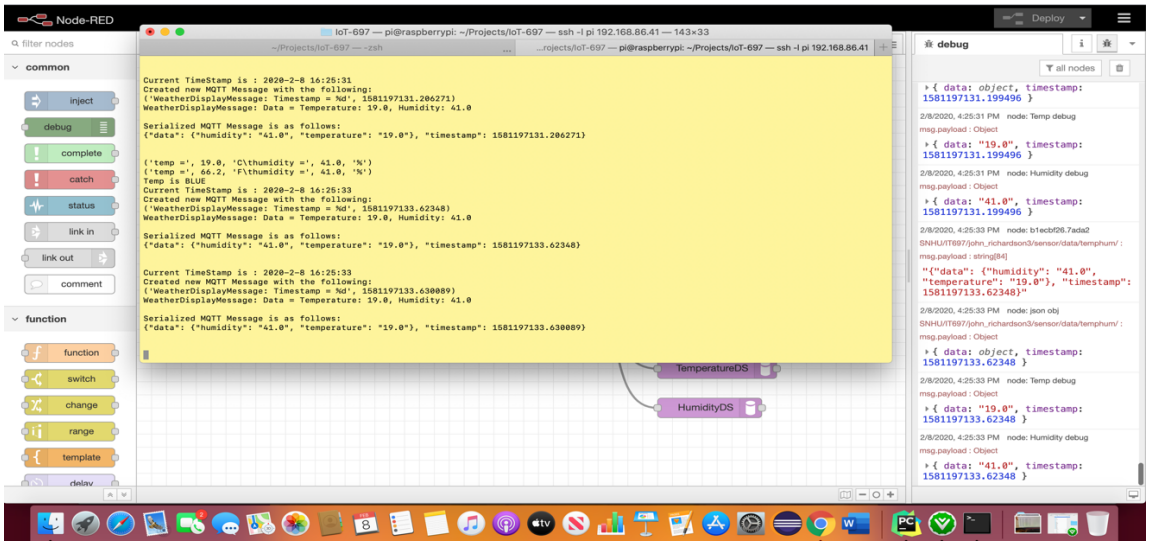

Figure 3 Running Program

- NodeRed dashboard capture and display of the NodeRed data sources created in the program flow

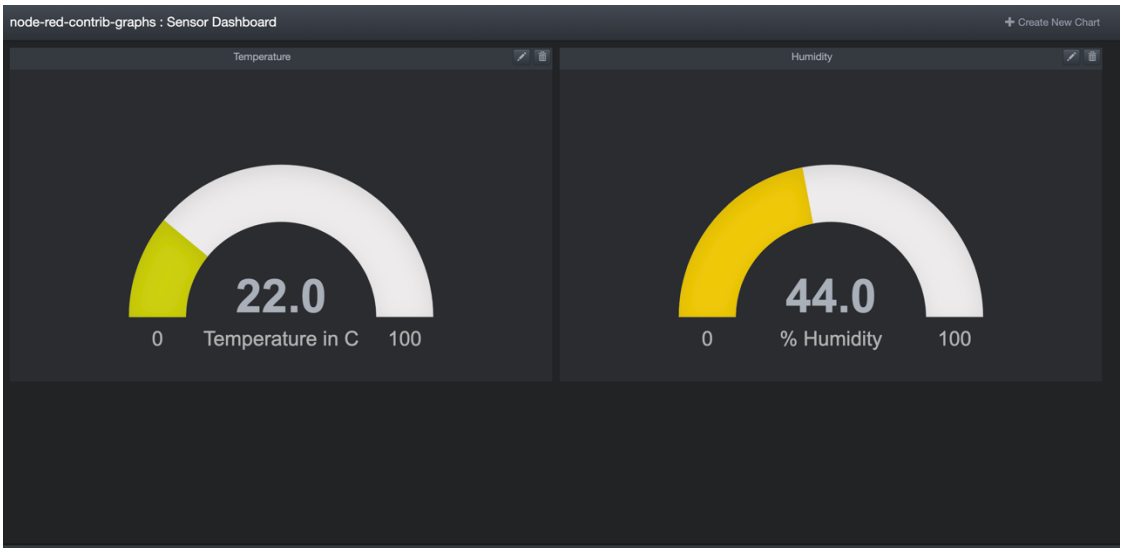

Figure 4 NodeRed dashboard showing data from data source

❖ **Saturday February 08, 2020**,

- ➢ Created GitHub Gist to share Code with class:
  - https://gist.github.com/johnnyrich0617/a7cff6c6119d888b2d0f3878bccc3eac
- ➢ Created Discussion Post to shar Gist with class
- ➢ Provided assistance to classmate Donna Stuart on code runtime issues

❖ Sunday February 09, 2020,

- Created Week 3 Timesheet
- Created Week 3 Journal

❖ Week 3 Code:

weather_display_message.json (json schema for project)

```
{
  "timestamp" : "The timestamp as epoch",
  "data" : {
    "temperature" : "The temperature as a string",
    "humidity" : "The humidity as a string"
  }
}
```

data_temhum_payload.py (Payload encapsulation for messaging)

```
class DataPayload:

    def __init__(self, temp, hum):
        self.temperature = temp
        self.humidity = hum

    def get_data_str(self):
        return "Temperature: " + self.temperature + ", Humidity: " + self.humidity
```

weather_display_message.py (Full message for publishing)

```
import time
import data_temphum_payload as payload
import jsonpickle


class WeatherDisplayMessage:

    def __init__(self, temp, humidity):
        self.data = payload.DataPayload(temp, humidity)
        self.timestamp = time.time()
        timeObj = time.localtime(self.timestamp)
        print('Current TimeStamp is : %d-%d-%d %d:%d:%d' % (
            timeObj.tm_year, timeObj.tm_mon, timeObj.tm_mday,
            timeObj.tm_hour, timeObj.tm_min, timeObj.tm_sec))

    def serialize(self):
        return jsonpickle.encode(self, unpicklable=False)

    def print_raw_content(self):
        print("WeatherDisplayMessage: Timestamp = %d", self.timestamp)
        print("WeatherDisplayMessage: Data = " + self.data.get_data_str())
```

```
    def print_serialized_obj(self):
        print(self.serialize())
```

HomeWeatherDisplayPublisher.py (Encapsulation code for MQTT publishing)

```
import paho.mqtt.client as mqttClient
import weather_display_message as wdm


class HomeWeatherPublisher:

    def __init__(self, mqtt_host, port, mqtt_client_id, topic):
        self.mqtt_host = mqtt_host
        self.mqtt_client_id = mqtt_client_id
        self.port = port
        self.topic = topic
        self.mqtt_client = mqttClient.Client(self.mqtt_client_id)

    def connect(self):
        print("HomeWeatherPublisher::Connecting to client with id = ",
self.mqtt_client_id)
        print("HomeWeatherPublisher::Connecting to host " + self.mqtt_host)
        self.mqtt_client.connect(host=self.mqtt_host, port=self.port)

    def stop_publishing(self):
        self.mqtt_client.disconnect()
        self.mqtt_client.loop_stop()

    def set_callbacks(self, on_connection, on_publish):
        self.mqtt_client.on_connect = on_connection
        self.mqtt_client.on_publish = on_publish

    def publish_msg(self, temp, humidity):
        mqtt_msg = wdm.WeatherDisplayMessage(temp=temp, humidity=humidity)
        print("Created new MQTT Message with the following:")
        mqtt_msg.print_raw_content()
        print("")
        print("Serialized MQTT Message is as follows:")
        mqtt_msg.print_serialized_obj()
        print("")
        print("")
        self.mqtt_client.publish(topic=self.topic, payload=mqtt_msg.serialize(),
qos=1)

    def is_connect(self):
        global CONNECTED
        return CONNECTED

    def get_client(self):
        return self.mqtt_client

    def get_topic(self):
        return self.topic
```

HomeWeatherDisplayMQTT.py (Main program flow)

```
# Home_Weather_Display.py
#
# This is an project for using the Grove RGB LCD Display and the Grove DHT Sensor from
the GrovePi starter kit
#
# In this project, the Temperature and humidity from the DHT sensor is printed on the
RGB-LCD Display
#
#
# Note the dht_sensor_type below may need to be changed depending on which DHT sensor
you have:
#  0 - DHT11 - blue one - comes with the GrovePi+ Starter Kit
#  1 - DHT22 - white one, aka DHT Pro or AM2302
#  2 - DHT21 - black one, aka AM2301
#
# For more info please see: http://www.dexterindustries.com/topic/537-6c-displayed-in-
home-weather-project/
#
'''
The MIT License (MIT)
GrovePi for the Raspberry Pi: an open source platform for connecting Grove Sensors to
the Raspberry Pi.
Copyright (C) 2017  Dexter Industries
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
'''

from grovepi import *
from grove_rgb_lcd import *
import time
from math import isnan
import paho.mqtt.client
import HomeWeatherDisplayPublisher as hwdp
import weather_display_message as wdm


# USE WITH OUT PUBLISHER
def local_on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to Local MQTT Mosquitto Broker....")
        global LOCAL_CONNECTED  # Use global variable
        LOCAL_CONNECTED = True  # Signal connection
    else:
        print("Connection to Local MQTT Mosquitto Broker failed..." +rc)
```

```
def public_on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to Public MQTT Mosquitto Broker...")
        global PUBLIC_CONNECTED  # Use global variable
        PUBLIC_CONNECTED = True  # Signal connection
    else:
        print("Connection to Public MQTT Mosquitto Broker failed..." +rc)


def local_on_publish(client, userdata, mid):
    print("LOCAL CLIENT PUBLISHED MESSAGE WITH ID: " + mid)


def public_on_publish(client, userdata, mid):
    print("PUBLIC CLIENT PUBLISHED MESSAGE WITH ID: " + mid)


'''
Setup the sensors
'''
# connect the DHt sensor to port 7
dht_sensor_port = 7
# use 0 for the blue-colored sensor and 1 for the white-colored sensor
dht_sensor_type = 0
# set the default back lighting to BLUE
setRGB(0, 0, 255)
# clear the LED Screen
setText_norefresh(" ")

'''
GLOBALS
'''
# global variables for connection state and other data
LOCAL_CONNECTED = False
PUBLIC_CONNECTED = False

'''
PUBLISHER Metadata
'''
PUBLISH_TOPIC = "SNHU/IT697/john_richardson3/sensor/data/temphum/"
local_broker_address = "localhost"
public_broker_address = "test.mosquitto.org"
port = 1883

'''
CONNECT AND MANAGE CONNECTION
THIS IS THE CODE USING ENCAPSULATED MQTT PUBLISHERS
LOCAL_PUBLISHER
PUBLIC_PUBLISHER
'''
local_publisher = hwdp.HomeWeatherPublisher(local_broker_address, port, "LOCALCLIENT",
PUBLISH_TOPIC)
public_publisher = hwdp.HomeWeatherPublisher(public_broker_address, port,
"PUBLICCLIENT", PUBLISH_TOPIC)

local_publisher.set_callbacks(on_connection=local_on_connect,
on_publish=local_on_publish)
public_publisher.set_callbacks(on_connection=public_on_connect,
on_publish=public_on_connect)
```

```
local_publisher.connect()
public_publisher.connect()

local_mqtt_client = local_publisher.get_client()
public_mqtt_client = public_publisher.get_client()

local_mqtt_client.loop_start()
public_mqtt_client.loop_start()
'''
END MQTT CONNECTION MANAGEMENT
'''

# Wait for connections
while not LOCAL_CONNECTED and not PUBLIC_CONNECTED:
    time.sleep(0.1)
    print("Sleeping...Waiting on connection")

while True:
    try:
        # get the temperature and Humidity from the DHT sensor
        [ temp,hum ] = dht(dht_sensor_port,dht_sensor_type)
        print("temp =", temp, "C\thumidity =", hum,"%")

        # check if we have nans
        # if so, then raise a type error exception
        if isnan(temp) is True or isnan(hum) is True:
            raise TypeError('nan error')

    # Convert the acquired temperature to Fahrenheit
        tempf = (temp * 1.8) + 32

    # Print the converted temperature and humidity
    # to the shell/console
        print('temp =', tempf, 'F\thumidity =', hum, '%')

    # If the temperature in Fahrenheit is greater then 60
    # change the LeD back lighting to RED
    # and print this condition
        if tempf > 68:
            print('Temp is RED........')
            setRGB(255,0,0)
        else:
        # If its not greater then 68
        # set the backlighting to BLUE and
        # print color condition to console
            print('Temp is BLUE.......')
            setRGB(0,0,255)

        t = str(temp)
        h = str(hum)
        tf = str(tempf)

        '''
        Publish messages to the broker using publisher objects
        '''
        local_publisher.publish_msg(temp=t, humidity=h)
        public_publisher.publish_msg(temp=t, humidity=h)
        # write to the sensor display LCD
        setText_norefresh("Temp:" + tf + "F\n" + "Humidity:" + h + "%")

    except (IOError, TypeError) as e:
```

```
        print(str(e))
        setText_norefresh(" ")
        local_publisher.stop_publishing()
        public_publisher.stop_publishing()

        # and since we got a type error
        # then reset the LCD's text

    except KeyboardInterrupt as e:
        print(str(e))
        setText_norefresh(" ")
        local_publisher.stop_publishing()
        public_publisher.stop_publishing()
        # since we're exiting the program
        # it's better to leave the LCD with a blank text
        break

    # wait some time before re-updating the LCD
    time.sleep(2)
```