# ECE 558 IoT Project Information and Preliminary Write-up

## Introduction

As I have mentioned in class quite a few times, I am developing a new IoT project and modifying the Multiple choice quiz app (was project 2) to advance my plan to move ECE 558 from what was an advanced Java/Android programming course to focus on interconnected devices. Even after the course update is complete, programming Android apps will still be one of the key learnings, but I intend to broaden the scope of the course to include internet-connected devices and RESTful web interfaces. To that end the new IoT project and the revised Multiple choice quiz app will make use of MQTT (MQ Telemetry Transport) and RESTful web-based API's.

While there are free (and/or free for a trial period) web-based servers for MQTT and RESTful interfaces the approach I am taking is that of a network-based (ethernet or WiFi) application with clients running on an Android device communicating with MQTT and RESTful services running on a Raspberry PI. It is my intent that programming the Raspberry PI will be done in Python. The Raspberry Pi has excellent support for Python; in fact, Python as often touted as being the scripting and programming language of choice for Raspberry Pi users.

Python has rapidly become almost a "must have" for many internships and full-time engineering positions. This came up in last week's ECE Industry Advisory Board meeting where several of our Industry partners stressed the need for improved programming skills, most notably Python, for both undergraduate and graduate students. Python, Java, and Javascript are consistently listed among the top most sought-after programming languages (https://www.computer.org/publications/tech-news/trends/programming-languages-you-should-learn-in-2020).

Although much can be learned through simulation and emulation, I strongly believe that hands-on experience working with, and debugging problems, on "real" hardware will enhance your skills and make you more "employable", besides give you projects to discuss and brag about in your search for an internship or job. Doing so will require that you have that hardware to work with even though it is an additional cost in a time when every $ counts.

## Overview of the IoT project

Project #2 is a two-person team project with an Android-based control app communicating with a **S**ingle **B**oard **C**omputer acting as the host for an array of sensors and actuators. We will use a Raspberry Pi (RPi) to host an MQTT server and two MQTT clients that interact with the sensors and actuators.

After investigating several options, I have chosen the Mosquitto MQTT server (https://mosquitto.org/) and the Paho MQTT client (https://www.eclipse.org/paho/) libraries for this project. These packages from the Eclipse community are open-source and have excellent "community" support.

Mosquitto can be hosted on several low-cost SBC's, including RPi and Arduino. We will use a Raspberry Pi 3, 3B, or 4 in this project. RPi's are available both standalone and in a variety of "Starter Kits" from

many vendors including Amazon and Adafruit. I believe it would be possible to use a Raspberry Pi Zero but I have some concerns about it's small physical size and ability to support full-size add-on boards (called HATs in RPi-speak). The RPi is Linux-based (Raspian, a Debian derivative) and can simultaneously support the MQTT server and the MQTT clients interacting with the sensors and actuators. Looking ahead to Project #3 (the revised Multiple choice app) I believe that the same RPi can be used as a web server running Flask (https://flask-restful.readthedocs.io/en/latest/ )

There are Paho MQTT client libraries in several programming languages, including Python and Java. The Paho MQTT client library for Android is wrapped into an Android Service PahoMQTT making Paho suitable for the Python-based MQTT clients running on the RPi and the Java-based Android app.

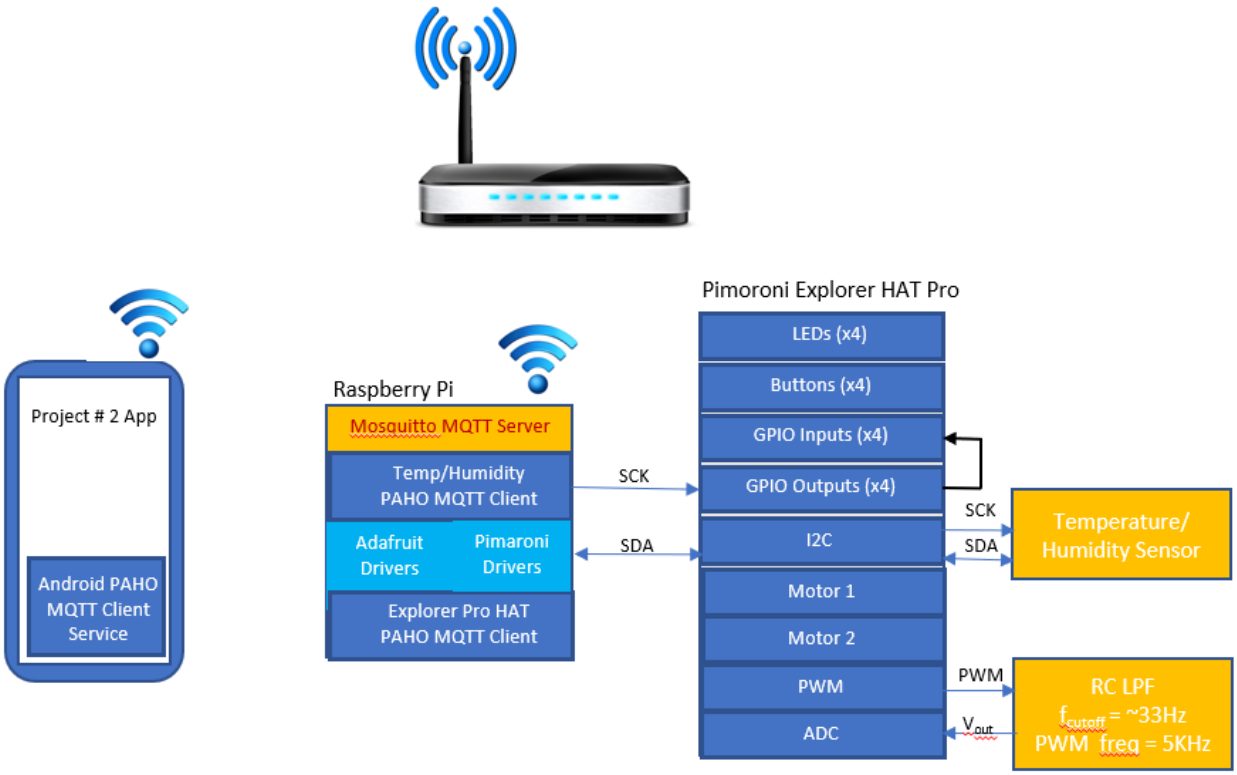
There are several tools to help you bring-up, and debug, your MQTT client apps. I used MQTT.fx (http://www.mqttfx.jensd.de/) under both Windows and on the RPi and the IoT MQTT Panel app (https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod&hl=en&gl=US) to prototype my Android app.

After researching alternative for sensors and actuators I have chosen the Pimaroni Explorer HAT Pro and an I2C-based temperature/humidity sensor.

The Pimoroni Explorer HAT Pro (https://shop.pimoroni.com/products/explorer-hat) is a full-size RPi HAT that provides 5V-safe,up to 500ma GPIO output and inputs, a 4 channel ADC (**A**nalog to **D**igital **C**onverter), 4 colored LEDs, 4 capacitive buttons, and 2 H-Bridge motor drivers (up to 200ma/channel). Pins for PWM, I2C, 3.3v, and SPI are brought out to a separate header. The peripherals on the board are accessible through a Python library and there is good documentation including a number of example programs.

There are several low-cost I2C temperature/humidity sensors that can be used on the project. I prototyped my design with the Adafruit AHT-20 breakout board (https://www.adafruit.com/product/4566) and the Digilent PmodHygro (https://store.digilentinc.com/pmod-hygro-digital-humidity-and-temperature-sensor/), but you can select a different sensor.

The figure on the next page is a block diagram of the IoT project architecture:




## Sensors:

We will implement the following sensors for this project:

- Buttons (Explorer Pro) – The Android app should show the status of the 4 buttons. My thought is that the app will include 4 widgets indicating whether each button is "pressed" or " not pressed."
- GPIO inputs (Explorer Pro) – The Android app should include 4 widgets indicating whether each of the 4 GPIOs is "on" or "off". The GPIO outputs on the Explorer Pro will be wrapped around to the GPIO inputs with jumper wires to provide the input.
- Temperature/Humidity (Temp/Humidity sensor) – The Android app should indicate the current temperature and humidity returned by the sensor. Temperature and Humidity values will be periodically measured and published to the Android app. The sample rate can be set in the Android app although my expectation is that the MQTT client responsible for the temperature/humidity readings can use a default sample rate. Temperature and Humidity can be displayed using one of the "progress bar" widgets.
- Analog voltage (Explorer Pro) – The Android app should display the analog voltage set by a PWM-based DAC (**D**igital to **A**nalog **C**onverter). The voltage could be displayed in a simple Textbox or the application could use one of the "progress bar" widgets.

## Actuators:

- LEDs (Explorer Pro) – The Android app should allow the user to turn the 4 colored LEDs "on" and "off" individually. The Android app could use toggle switch widgets.

- GPIO outputs (Explorer Pro) – The Android app should enable the user to turn each of the 4 GPIO inputs on and off. The Android app could use toggle switch widgets.
- PWM DAC (Explorer Pro + external RC circuit) – The Android app should use one of the "slider" widgets to control the DAC output voltage.

  Neither the Explorer Pro or the RPi include an integrated DAC, but it is easy to prototype one. The Explorer Pro brings one of the PWM-capable outputs to a header. This PWM output could be connected to a single-pole passive low pass RC filter. My prototype varies the duty cycle of a 5KHz signal between 0 and 100 percent, resulting in an output from ~0v to ~3.3v. My filter uses a 1KΩ resistor driving into a 5µF capacitor from two 10µF electrolytic capacitors connected in series. This is not ideal, but these are the components I had available and are common enough that they are typically included in Experimenter (or "Electronic Accessory") kits. The filter has a cutoff frequency of ~34 Hz. There is a small amount of ripple and voltage variation in the DAC output, but this DAC is adequate for the project. There are much better filters and/or DAC solutions – feel free to innovate.

- Motor Controllers (Explorer Hat) – Unused in the project but could be used towards extra credit on the project.

## Task list

Here is a rough list of tasks for the project:

1. Procure the hardware and software needed for the project. I have provided a BOM in the next section.
2. Bring up the RPi. This will include installing a Raspian image on a microSD card (8GB minimum but I suggest using 16GB or 32GB card. They are typically on sale at Best Buy, Office Depot, etc. and are often cheaper than an 8GB card). With the RPi 3, 3B, or 4 you can use either WiFi or a hardwired ethernet connection to your home router. I use SSH connections for both command line (MobaXterm and/or PuTTy) and the RPi Desktop (VNC Connect and VNC Viewer) for my development work on the RPI. There are excellent tutorials on how to configure a "headless" RPi system.
3. Install the Mosquitto server, the MQTT client library for Python, the Python libraries for the Explorer Pro and your temperature/humidity sensor, and your preferred Python development environment. I am using Microsoft VSCode with several Python extensions and find it easy to use and more than adequate for my purposes. I have created Python Virtual environments for each phase of the work I have done so far. You will become proficient in using `PIP`, `sudo` and `apt` in the process.
4. Install the Explorer Pro and the temperature/humidity sensor hardware. Run the test programs that are included in the libraries to confirm that the devices are operating correctly.
5. Design the list of MQTT "topics" and the format for commands and responses. You will develop two MQTT clients to run on the RPi – one for the Temperature/Humidity sensor and the other for the Explorer Pro. While they could be combined into a single client, I don't believe that this is the best design choice. The Publish/Subscribe methodology used by MQTT supports multiple clients and it cleaner to group related functionality and separate it from non-related functionality. While it would be a good idea to provide "Secure links" and user/password

authentication in the system it is not necessary for this project and adds additional complexity without a large gain in our non-global client/server environment.

6. Write the two MQTT clients that Publish and Subscribe to the relevant topics. The development of each of these clients can be done independently.
7. Verify (and debug if necessary) the MQTT clients using MQTT.fx or the command line MQTT Publish/Subscribe utilities. I would do this first on the RPi and then remotely with your PC connected to the RPI on your network.
8. Use an app like IoT MQTT Panel (there are several in the Google Play Store) to prototype your Android application. The requirement for the project is to develop an Android app using the Android SDK but prototyping with an Android IoT MQTT client API will give you a head start since you can essentially debug and test the MQTT communication with your clients.
9. Develop an Android app that monitors and displays the sensors and controls the actuators. Integrate the Android app with the MQTT server and clients running on the RPi. If you have bought up your system using an MQTT client app your debug should be limited to problems in your application.
10. Prepare and submit your deliverables. Deliverables should include all the source code you developed, a Theory of Operation/Design report and a demo video. Since this is a team-based project we expect you to do version control with GitHub and GitHub classroom.

## Bill of Materials

Here is a preliminary list of materials you need to implement this project:

- Raspberry Pi 3 or 3B or 4 – Available from many sources either as a standalone product or bundled into "starter kit" that includes a power supply, case, and, in some kits, a USB media card "reader" that you can use to create the microSD card that holds the Operating System, tools and utilities to run the RPi. The "native" OS for the RPi is Raspian which is a Debian Linux release.
- Pimoroni Explorer Pro Hat – Available from several sources including Amazon, Adafruit, Pimoroni, PiShop, etc.
- Temperature/Humidity sensor. There are several low-cost sensors available from several vendors including Amazon and Adafruit. You may find it easiest to make use of an I2C-based sensor. Be sure to research the library support for the sensor.
- Prototyping kit – At a minimum you will need a prototyping board (the one on the Explorer Pro Hat is very small), jumper wires, and an assortment of capacitors and resistors. There are several reasonably priced Electronic Accessory Kits. I purchased a kit from Amazon (https://www.amazon.com/gp/product/B07H7VZCX5/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1) for $11.99 but Adafruit, Sparkfun, Pimoroni, and others have equivalent kits.
- (optional) – A USB to Serial debug/console cable. Although the process for creating a headless RPi system is straightforward and well documented, I have found that it is useful to have a Serial debug cable in my electronic tool box. These cables provide a serial interface to the Raspian shell and console. For example, let us say that you followed all of the instructions but your RPi Wifi link did not come up. You can use a serial debug cable to the RPi console to run `raspi-config` (the configuration utility included with Raspian to enable and configure WiFi, SSH, VNC, username/password, etc. These cables are not overly expensive. The Adafruit cable I use costs

under $10 (https://www.adafruit.com/product/954). It is also available from Amazon at a higher cost (https://www.amazon.com/gp/product/B00DJUHGHI/ref=ppx_od_dt_b_asin_title_s00?ie=UTF8&psc=1).

## Useful Links

- *Digital to Analogue Conversion with Raspberry Pi*
  https://core-electronics.com.au/tutorials/digital-to-analogue-conversion-with-raspberry-pi.html
- *How to Set Up a Headless Raspberry Pi, Without Ever Attaching a Monitor*
  https://www.tomshardware.com/reviews/raspberry-pi-headless-setup-how-to,6028.html
- *Setting up a Raspberry Pi Headless*
  *https://www.raspberrypi.org/documentation/configuration/wireless/headless.md*
- *MQTT with a Raspberry Pi and an Arduino*
  https://www.youtube.com/watch?v=p3vJxGKWDIg
- *Playing with MQTT by Android*
  Part 1: https://www.youtube.com/watch?v=BAkGm02WBc0
  Part 2: https://www.youtube.com/watch?v=6AE4D8INs_U&t=3s
- *Paho Android Service – MQTT Client Library Encyclopedia*
  https://www.hivemq.com/blog/mqtt-client-library-enyclopedia-paho-android-service/
- *Paho Python – MQTT Client Library Encyclopedia*
  https://www.hivemq.com/blog/mqtt-client-library-paho-python/
- *Hands-on MQTT Programming with Python* by Gaston Hiller
  https://subscription.packtpub.com/book/application_development/9781789138542
- *MQTT Essentials - A Lightweight IoT Protocol* by Gaston Hiller
  https://subscription.packtpub.com/book/application-development/9781787287815