IoT Project – Draft 2

All steps provided below are created by us using various tutorials found online mixed with our own personal experiences implementing our system

Our Core Platform – Home Assistant

We opted for the Home Assistant platform to use on our prototype as it is powerful, open-source, well maintained by a large community, and has support for almost every smart-home IoT device. This means it can act as a bridge between closed-ecosystem products, open-source products, and DIY projects. Our main reason for going with HA is that although its relatively easy to use, it includes a high degree of granularity which allows the user to really get into the fine-tuning of their setup. Therefore, it provides the ability to quickly create efficient, cohesive systems that can be monitored via a robust and dynamic UI, which is perfect for prototyping.
We tried using cloud-based systems such as Azure IoT and AWS IoT but found it difficult to setup and implement without spending hours researching bug issues and attempting fixes that never worked.
With this platform all we needed to do to gain remote access is enable port-forwarding on a router and use a secure proxy and domain.

Gateway Node Setup – Team

Setting up the gateway node requires installing the Home Assistant OS on a Raspberry Pi Model 3 or later, or on any of the other supported platforms. In our case we are using a Raspberry Pi Model 3B+.
First download Home Assistant or copy the download URL from the official website and use BalenaEtcher to download it to a microSD card. Then, insert the card into the Pi and wait for Home Assistant OS to boot.
https://www.home-assistant.io/installation/
https://www.balena.io/etcher/

Connect the Pi to your LAN via the Ethernet port, you will be able to access the dashboard from your browser using homeassistant.local on port 8123 (**homeassistant.local:8123**), no monitor, mouse or keyboard are required to set this up. If for whatever reason this doesn't work, try using the IP of the Pi running Home Assistant explicitly, e.g. **192.168.0.20:8123**. You can find this out easily on any MacOS, Windows or Linux machine by running this command in your respective command shell:

--Run

```
1.  arp -a
```

This will give list of all the IP and corresponding MAC addresses within your LAN, all you need to do is find the MAC of the Pi running Home Assistant, or simply run the above command before and after connecting the Pi to your LAN and taking the newest IP address. In most situations localhost will work perfectly fine.

When Home Assistant has fully started up, create an account and go through the short setup process.
You should then be presented with the Home Assistant dashboard. Navigate to **Configuration > Add-ons, Backups & Supervisor** and click the blue **Add-ons Store** button in the bottom right corner.
Add-ons required are:
Mosquitto broker

MQTT Broker Setup
The setup process for the MQTT Broker is very straightforward using the guide below:

https://github.com/home-assistant/addons/blob/master/mosquitto/DOCS.md

Room Prescence Setup – Douglas

Our Room Prescence nodes use ESP32 microcontroller board with a PIR sensor wired to them. The boards run a small snippet of MicroPython code which detects whether there is someone in the vicinity and sends the result via MQTT to the Mosquitto broker on our gateway Home Assistant node. This then updates the 'Rooms Occupied' card in the dashboard of Home Assistant to show which rooms are currently occupied.

Setting up the ESP32's requires flashing them with MicroPython software and loading them with code.
https://micropython.org/download/

To do this you first have to make sure you have the right VCP (Virtual COM Port) drivers from the FTDI website and have esptool.py installed on your machine using PIP or Brew.
https://ftdichip.com/drivers/vcp-drivers/
https://pypi.org/project/esptool/

```
1.  sudo pip3 install esptool
```

Or

```
1.  brew install esptool
```

Once the ESP32 is plugged into your machine using a power AND data USB cable (this is something we had a lot of issues with and cost us a lot of time), look for which COM port the ESP is on, and if you are on a UNIX or Linux based system, look for the name your system has identified the ESP as (usually under `/dev/tty` or `/dev/cu`).
I'm running MacOS so I used '`ls /dev | grep tty`' and '`ls /dev | grep cu`' in the bash terminal, although I later found a much easier way by running '`esptool.py flash_id`'.

Once the ESP32 is identified, run the following commands in your terminal. Also make sure you're in the directory where you downloaded the MicroPython wares to.
(My ESP identified as '`/dev/tty.usbserial-0001`', and the file name I'm using is 'esp32-ota-20220117-v1.18.bin').
Something else I did to make the process a bit easier; make a folder in my Documents directory and put the MicroPython bin file in there along with the Python code files, this means you don't have to switch directories during the process which can be confusing and error-inducing if you forget to do it (guilty).

--Run

```
1.  esptool.py --port /dev/tty.usbserial-0001 erase_flash
2.  esptool.py --chip esp32 --port /dev/tty.usbserial-0001 write_flash -z 0x1000 esp32-ota-20220117-
    v1.18.bin
```

If the terminal is failing to connect to the ESP device, it'll be in the wrong boot mode, so hold down the boot button on the board and run the commands again, keeping it pressed until the terminal shows it has started erasing or writing the flash.

Once completed, install 'ampy' on your machine. This is used to interface with MicroPython from the terminal to load, view and edit code on the ESP. While ampy has technically been deprecated for use with MicroPython in place of CircuitPython, it still works fine, and I had no issues using it.
https://learn.adafruit.com/micropython-basics-load-files-and-run-code/install-ampy

--Run

```
1.  sudo pip3 install adafruit-ampy
```

Once ampy is installed, you're ready to start loading code onto the ESP using the 'put' command. First navigate to the directory the code files you want to upload are located. Also, if using the files we have provided, make sure to edit the network settings in the boot.py and main.py files to match your specifications.

--Run

```
1.  ampy -p /dev/tty.usbserial-0001 put boot.py
```

Once you've loaded the boot.py files onto the ESP containing your network connection commands, you then need to access the MicroPython REPL to install the umqtt library. This must be done BEFORE the main.py file is uploaded, or it becomes a hassle to get the ESP to stop running the code in order to install the library. As I'm on MacOS, I used the pre-installed 'screen' emulator in the bash terminal to access the REPL. On Windows you can do the same by downloading 'puTTY'. Also, on MacOS and Linux you may have to specify the baud rate after the port name, I used 115200 baud as that's the max transfer rate officially supported by my ESP boards.
https://www.putty.org

--Run

```
1.  screen /dev/tty.usbserial-0001 115200
```

Once the REPL is active, you'll see the network connection code running from our boot.py file. Once this has finished running, you'll see the REPL command line, identified by the three arrows: '>>>'.

--Run

```
1.  import upip
2.  upip.install("umqtt.simple", "lib")
```

Once this has finished, exit the REPL using 'Control C + \' and then hit 'y' when asked if you'd like to exit the screen. I believe this is a screen only command so if using a different emulator refer to their documentation on how to exit. Now we're ready to upload our main.py file, remember again to be in the correct directory on your machine or you'll get some form of 'file/directory not found' errors.

--Run

```
1.  ampy -p /dev/tty.usbserial-0001 put main.py
```

Repeat for any other ESP's you wish to flash, and that should be them all set up. If you want to see if the code files have uploaded correctly, or if you're having issues and want to check the code syntax, then use the same 'ampy' commands as above but with 'get' in place of 'put', the terminal should print the code inside the relevant Python file stored on the ESP device.

M5Stick C Plus Wearable – Douglas

Unfortunately, getting this device set up to run the code I made for it did not work at all. For some reason it wouldn't fully communicate with any of the machines I tried flashing it with, be it MacOS, Windows or Ubuntu. I'm not sure why this didn't work as I have all the correct drivers, it shows up in my system reports as 'M5Stack' so it is being interfaced with, but no matter what I did I couldn't flash the board with any software. This was surprising to me as it's a popular device based on a well-established platform (ESP32-Pico), and while M5Stacks' documentation is flawed and missing certain important bits of information, most people don't have an issue using them. We even tried uploading some sample code to it via the Arduino IDE and it just wasn't having it, despite this being a very popular method of using the M5Stick. My best guesses are that either all of our systems are missing an inconspicuous dependency required to fully interface with it, or the board is faulty, which seems most likely at this point.

This seems a shame as the M5Stick C Plus is a well featured product with a built-in screen, LiPo battery, microphone, MPU6886 for spatial awareness, is available with a wrist strap, all while being at a very affordable price. We really thought this device would be one of our main nodes as it's the one the end-user would interact with the most, and its impressive capabilities meant we could have implemented some cool ideas on top of its main function, but after days of trying to get it to work with our various methods, we've declared it redundant.

Pi Camera Setup

Our Raspberry Pi camera node consists of a Raspberry Pi Model A+ and a Pi camera module. The camera stream is sent over the LAN via RTSP to the gateway Home Assistant node. If there is a Room Prescence topic published to the MQTT broker, Home Assistant then processes the video stream using a TensorFlow AI model to determine the occupier's current position e.g. standing, sitting, walking, fallen.