EXPERIMENT 2

AN INTRODUCTION TO THE INTERNET OF THINGS (IOT)
EXAMPLE II – PART 1

DOCUMENT INFORMATION	
COURSE TITLE	ELECTRONICS DESIGN II
COURSE CODE	ELET2415
DATE	
STUDENT ID	

DOCUMENT LAST REVISED
WINTER 2024

Blank Page

ABOUT THIS EXPERIMENT:

Experiment 2 – An Introduction to the Internet of Things Systems – Example II – Part 1. This laboratory activity serves as another example on the fundamentals of IoT Architecture and Development. Again, students are tasked with engaging in a series of hands-on activities. It is important to acknowledge that some information in this document has been sourced from various references, including but not limited to, ***Yana Creations*** (https://www.yanacreations.com/), ***Microchip Technology*** (https://www.microchip.com/) and ***Arduino*** (https://www.arduino.cc/).

EXPERIMENT OBJECTIVES:

- ☐ Continue working with the Arduino Integrated Development Environment (IDE).
- ☐ Design, write and debug simple application programs in the Arduino IDE.
- ☐ Garner an understanding the Internet of Things (IoT) System Architecture.
- ☐ Garner a working knowledge of Technologies Utilized in Backend IoT Development.
- ☐ Garner a working knowledge of Technologies Utilized in Frontend IoT Development.

PRE-LABORATORY REQUIREMENTS:

Before attempting the laboratory activities, students are required to:

- Complete Experiment 1.
- Complete the MongoDB introductory course.
- Read through this experiment manual.

EXTREMELY IMPORTANT:

- Fork the "hydromonitor" IoT template repository from https://github.com/iotHub1/hydromonitor.git to your own Github account.

- Use the command line terminal (CMD terminal on windows) to clone the hydromonitor IoT template repository from your own Github and carry out the instructions outlined in the `README.md` file to setup and run your application.

EQUIPMENT AND COMPONENTS:

Table 1 list the equipment and components required to carry out the activities.

Table 1 – Equipment and Components Required for Experiment 1	
Equipment	• Digital Multimeter, Breadboard, Set Jumper Wires, 5V Power Supply • A Windows computer loaded with the latest version of all the required software. See the course webpage.
Component	• Arduino Nano and ESP32 Development Boards

INTRODUCTION TO ARDUINO ANALOG OERATIONS, FASTLED LIBRARY, AND THE DHT22:

Many Arduino boards, if not all, are equipped with a multichannel analog to digital converter (ADC). In particular, the Arduino Nano has a 10-bit ADC which maps input voltages, between 0 and 5V, on each of the analog input pins integer values between 0 and 1023. The ESP32 has a built in 12-bit ADC which maps input values between 0 and 4095.

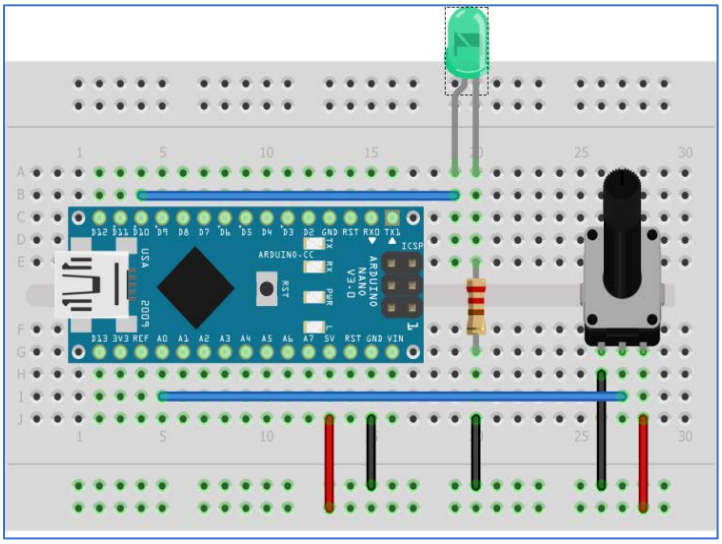
In this activity students will be exposed to the following analogy functions.

- `analogRead()`. Used to read the analog values to analog pins.
- `analogWrite()`. Used to write analog values to analog pins.

Additional information on these concepts is available for reference at https://www.arduino.cc/reference/en/ and https://www.arduino.cc/reference/en/https://docs.arduino.cc/learn/microcontrollers/analog-input .

Tutorial 0.1 – Working with Analog I/O:

In this initial activity, students are required to employ a potentiometer to regulate the brightness of an LED. The goal is to observe changes in the LED's brightness as the potentiometer is adjusted. Construct the circuit outlined in Figure 1, compile and upload the provided sketch to the Arduino Nano. Save the sketch as **`analog.ino`**.



analog.ino

```
#define analogPin A0
#define LED 10

uint16_t value = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(LED,OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
  value = analogRead(analogPin);
  value /= 4; // Maps 0-1023 ADC values to 0-255
  Serial.println(value);
  analogWrite(LED, value);
}
```

Figure 1 – Circuit and Sketch

Tutorial 0.2 - FastLED Library

The FastLED Arduino library comes with a number of examples that students can utilize to help in building a comprehensive understanding of how to use this library to interface with any addressable LED strip or array. The official documentation, which contains example sketches for this library is available for reference at https://fastled.io/docs/index.html#autotoc_md0 . In this activity students will utilize the following function of the library to control the addressable LED(s).

Design and construct a circuit employing the Arduino Nano board and the 7-bit addressable LED module. Upload the **`[ File > Examples > FastLED > Blink ]`** sketch that came with the library to the board. Observe the LED module. Feel free to modify this sketch to help build your understanding.

Subsequently, modify the previous sketch with the code given below.

analog.ino

```
void loop() {
    // put your main code here, to run repeatedly:
    // This example shows the different ways of using the CRGB function
    uint8_t brightness = 255; // brightness values ranges from 0 to 255

    for(int x=0; x<7; x++){
      leds[x] = CRGB( 0, 0, 255);          // R, G, B range for each value is 0 to 255
      FastLED.setBrightness( brightness ); // Ranges from 0 to 255
      FastLED.show();                      // Send changes to LED array
      delay(50);
     }

     delay(1000);
    for(int x=0; x<7; x++){
      leds[x] = CRGB( 255, 255, 255);
      FastLED.setBrightness( brightness );
      FastLED.show();
      delay(50);
     }

     delay(1000););

     for(int x=0; x<7; x++){
      leds[x] = CRGB::Black;
      FastLED.setBrightness( brightness );
      FastLED.show();
      delay(50);
        }

    delay(1000);

    for(int x=0; x<7; x++){
      leds[x] = CRGB::Green;
      FastLED.setBrightness( 255 );
      FastLED.show();
      delay(50);
     }

    delay(1000);

}
```

Tutorial 0.3 - DHT22 Temperature and Humidity Module

Students are required to complete a section of the tutorial at https://learn.adafruit.com/dht, excluding the sections regarding python programming. This activity should be done utilizing the Arduino Nano and aims to provide students with an understanding in interfacing with this sensor.

HYDROPONICS MONITORING SYSTEM:

Like Experiment 1, in this laboratory experiment, students are tasked with creating an IoT system based on the architecture provided in the system architecture shown in figure 3. For this example, students will implement some basic interface of a hydroponics farm.

The aim is to develop and implement a control and monitoring IoT system for a hydroponics farm. The hydroponics system is a controlled environment designed for optimal plant growth. It incorporates technology, featuring a temperature and humidity sensor to monitor the atmospheric conditions within the cultivation space. These sensors assist with temperature regulation, maintaining an optimal range for plant development. Also managing humidity levels to create an ideal growing atmosphere.

Additionally, the system integrates LED lighting designed to mimic the sun's spectrum, providing frequencies of light essential for photosynthesis. These LED lights are programmable, allowing for the customization of light cycles tailored to the specific needs of the cultivated plants. By simulating natural sunlight, the hydroponics system can optimize the plant's ability to absorb nutrients; promoting robust growth, accelerated maturation, and enhanced overall yield. This setup increase efficiency and minimizes resource usage, making it an environmentally friendly and sustainable approach to agriculture.

Figure 2 – Hydroponics Farm

IoT System Architecture and Implementation:

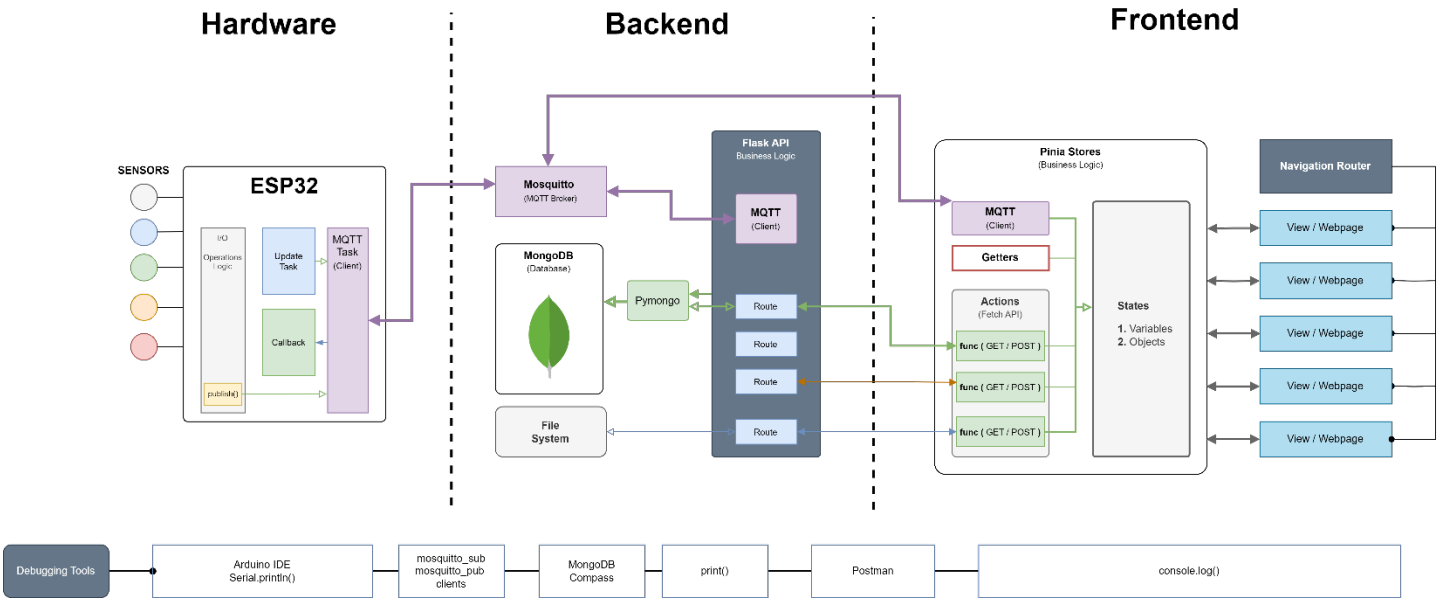


Figure 3 – System Architecture

The hydroponics farm's control system is an IoT system based on the architecture illustrated in the provided system diagram in figure 3. Specifications for each component will be outlined for the tasks required for the whole system development.

The **Hardware** component has three tasks: (1) The remote control of the colour spectrum (frequency of light), the number of LEDs activated and the brightness of the LEDs within the hydroponics system. (2) To monitor the farm environment. The hardware employs a DHT22 temperature and humidity sensor. Heat index must be calculated based on the sensor's data. The collected sensor data must be published to a topic subscribed to by both the backend and frontend segments of the system. (3) Process and execute actions in response to messages published to the topics to which it is subscribed. This comprehensive functionality ensures seamless control, monitoring, and responsiveness within the hydroponic cultivation environment.

The **Backend** component has two tasks: (1) Store data, published by the hardware, in the database following the schema specified in the hardware specifications. (2) Allow accessibility of the stored data to the frontend through API routes.

The **Frontend** should deliver three distinct user interfaces: (1) Controls for managing the system's lighting. (2) For real-time visualization of data generated by the environmental monitoring sensors. (3) For visualizing and performing statistical analysis on the data. This multifaceted approach ensures a comprehensive and user-friendly experience, empowering users to efficiently interact with and derive insights from the hydroponics system.

Activity 1 – Implementing the Hardware Component (ESP32, LED Module, DHT22)

The hardware component specifications are as follows, note that the first two specifications have already been completed. **Note:** This activity must be completed using the ESP32 board.

- ✓ Implement the Network Time Protocol (NTP) to maintain system time with 1 second accuracy.
- ✓ Implement the MQTT protocol for bi-directional communications between Hardware, Backend and Frontend sections.
- ☐ Able to control a 7-bit addressable LED using **`FastLED`** library.
- ☐ Interface with the **`DHT22`** temperature and humidity senor.
- ☐ Able to calculate Heat Index from **`DHT22`** sensor data.
- ☐ Process and action messages published by the frontend with the following or other similar schemas.
  `'{"type": "controls", "brightness": 255, "leds": 7, "color": { "r": 255, "g": 255, "b": 255, "a": 1 } }'`
- ☐ Publish updates from the sensors every second, to the backend and frontend following the schema.
  `'{"id": "student_id", "timestamp": 1702212234, "temperature": 30, "humidity":90, "heatindex": 30}'`

Learning objectives:

☐ Interfacing with addressable LED modules using FastLED library.
☐ Interfacing with the DHT22 sensor module.

Extremely Important:

Before proceeding with the implementation of the remaining hardware specifications, it is critical to adjust the **`hardware/hardware.ino`** code to incorporate specific configuration details. Specifically: (1) Updating the Wi-Fi credentials to enable internet connectivity for the ESP32 on any accessible network. (2) Edit the MQTT client configuration settings to include the necessary server and port details, and the default topic for hardware publishing to include all relevant topics for hardware subscription. Some guidance can be obtained from the '***Wi-fi and MQTT configuration on Arduino***' section in the Appendix.

A. Using KiCAD design and then construct a circuit employing the ESP32 board, the DHT22 sensor and the 7-bit addressable LED module. The pin configuration for the ESP32 board can be found in the Appendix. Retrieve all necessary KiCAD components from an online source or create a new component if it is not already installed in your KiCAD library. [**Required Effort Level 3.0**]

B. Import required libraries for all sensors and devices you will be using and instantiate any sensor classes required.

```
hardware.ino
// LIBRARY IMPORTS
#include <rom/rtc.h>
#include <math.h>  // https://www.tutorialspoint.com/c_standard_library/math_h.htm
#include <ctype.h>

// ADD YOUR IMPORTS HERE
```

C. Insert the appropriate initializations and pin configurations for all devices. [**Required Effort Level 1.0**]

```
hardware.ino
void setup() {
  Serial.begin(115200);  // INIT SERIAL

  // INITIALIZE ALL SENSORS AND DEVICES

  /* Add all other necessary sensor Initializations and Configurations here */

  initialize();           // INIT WIFI, MQTT & NTP
 //  vButtonCheckFunction(); // UNCOMMENT IF USING BUTTONS THEN ADD LOGIC FOR INTERFACING WITH BUTTONS IN THE vButtonCheck FUNCTION
```

D. Complete all utility functions. [**Required Effort Level 1.0, 1.0, 4.0**]

```
hardware.ino
//***** Complete the util functions below ******
double convert_Celsius_to_fahrenheit(double c){
    // CONVERTS INPUT FROM °C TO °F. RETURN RESULTS
}

double convert_fahrenheit_to_Celsius(double f){
    // CONVERTS INPUT FROM °F TO °C. RETURN RESULT
}

double calcHeatIndex(double Temp, double Humid){
    // CALCULATE AND RETURN HEAT INDEX USING EQUATION FOUND AT
    // https://byjus.com/heat-index-formula/#:~:text=The%20heat%20index%20formula%20is,an%20implied%20humidity%20of%2020%25
}
```

E. Implement the vUpdate function according to the hardware specifications. This function plays a crucial role in transmitting all sensor data to both the backend and frontend every second. [**Required Effort Level 3.0**]

```
hardware.ino
void vUpdate( void * pvParameters ) {
}
```

F. The function defined in the code block below, defines a callback function that will be executed whenever a message is received on a topic subscribed to by the microcontroller. It is partially completed in the

hardware/hardware.ino file. Students are required to complete this function in order to process and action any message received from either the backend or frontend according to the schemas outlined in the hardware specifications. The MQTT messages received are in the format of JSON strings and must be converted to JSON objects before processing. [**Required Effort Level 3.0**]

hardware.ino

```
void callback(char* topic, byte* payload, unsigned int length) {
  // ############## MQTT CALLBACK  ####################################
  // RUNS WHENEVER A MESSAGE IS RECEIVED ON A TOPIC SUBSCRIBED TO

  Serial.printf("\nMessage received : ( topic: %s ) \n",topic );
  char *received = new char[length + 1] {0};

  for (int i = 0; i < length; i++) {
    received[i] = (char)payload[i];
  }

  // PRINT RECEIVED MESSAGE
  Serial.printf("Payload : %s \n",received);
}
```

Regarding the callback function, messages published by the frontend are of type "controls" as outlined in the schema given in the hardware specifications. You are required to extract the various control parameters, including the number of LEDs, RGB color values, and brightness from the JSON object.

Additionally, the code must iterate through the LED module (0 to 6), setting the colour and brightness for the number of LEDs specified in the **"leds"** field in the received MQTT message. The FastLED.show() function should be employed to update the LED module, and a small delay introduced between iterations using vTaskDelay() to ensure smooth execution.

Additionally, LEDs beyond the specified number in the received message should be turned off by setting them to black.

This would be a great time to commit and push your code to your GitHub account.

For guidance, revisit the 'Commit and push your code to GitHub' section available in the Appendix.

Activity 2 – Backend (Database, API)

A substantial portion of the backend specifications has been provided; the primary task is for students to engage in independent research. This involves researching technologies including (Pymongo driver, MongoDB CRUD & Aggregation operations, Flask API, MQTT, etc) to enhance their understanding of the IoT system's backend. It emphasizes the importance of personal initiative in acquiring a nuanced understanding and mastery of the backend technologies. Specifications for the backend are as follows:

- ☐ Insert sensor data published by hardware into mongoDB database (Database name: ELET2415, Collection name: climo) using appropriate CRUD function.
- ☐ Create a collection called ‘climo’ using MongoDB Compass.
- ☐ Create a unique index on the ‘timestamp’ field for the climo collection of your MongoDB database.

Important Note: Students understanding of these technologies will be tested during lab demonstrations weekly and will count towards student’s lab grade.

Activity 2.0 - Creating a new MongoDB Collection and Unique Index [**Required Effort Level 6.0**]

Create a local MongoDB user account for the Flask API, if you haven’t done so already. **Students should have already done this if they have completed Laboratory I**. Guidance on creating a local account is available in the Appendix.

Students are required to (1) create a collection named 'climo'. (2) Create a unique index on the timestamp field within the climo collection. For detailed instructions, please consult the 'Creating a MongoDB Database, Collection, and Unique Index' section in the Appendix. This collection will be utilized to store data from the hardware component.

Students are also required to configure the MQTT backend client. While configuring MQTT client students are required to implement a callback function called **“update”** which must inserts the sensor data published by the hardware component into the ‘climo’ collection of the ELET2415 database. See “Backend MQTT configuration” in the Appendix for guidance. The update function should use the ‘addUpdate’ function defined in functions.py to insert the data into the database.

Note:

Implementation for the rest of the Backend will be accomplished during Experiment 3.

Activity 3 – Frontend (Vuejs, Vuetify – HTML, CSS and JavaScript)

Specifications for Webpage 1: route name Control

- ☐ UI for controlling brightness.
- ☐ UI for controlling number of active LED(s) segments.
- ☐ UI for selecting LED(s) colour.
- ☐ UI for indicating the number of active LED segments.
- ☐ On change of any of the UI(s) above, publish a message to a topic subscribed to by the hardware using the following schema `'{"type": "controls", "brightness": 255, "leds": 7, "color": { "r": 255, "g": 255, "b": 255, "a": 1 } }`.

Frontend section learning objectives:

- ☐ Learn how to create and use Vuejs refs and reactive variables.
- ☐ Learn how to bind components to ref and reactive variables.
- ☐ Learn how to create and use Vuejs Watchers and Computed properties.

Create View/Webpage

Create a new View for the 'Control' navigation route. For guidance, revisit the 'Creating a View/Webpage for navigation routes' section available in the Appendix. Modify it to implement the frontend specifications for Webpage 1: route name Control.

Remove the `<div>` element from the template before continuing.

UX/UI – Modify the webpage to look like the image illustrated in the figure below. This will be implemented in three (3) steps.

1. Create the UI layout in template section using grid system and add the required CSS styling to the style section.
2. Add desired Vuetify components found at https://vuetifyjs.com/en/components/all/.
3. Add business logic to the webpage by adding javaScript code to the script section as well as the corresponding store found in frontend/src/store for updating all state variables and objects using the data received from both hardware (via MQTT) and backend (via API calls).

Important Note: These steps are general and should be applied in order to effect frontend specifications throughout the course.

Create UI Layout **[Required Effort Level 2.0]**

- Add a container. Set the **fluid** property, and align center.
- Add one (1) row inside the container. Give it a max-width of 1200px.
- Add two (2) columns inside the row.
- Set the align property for all columns to center and the justify property for all rows to center.

Add Vuetify Components **[Required Effort Level 2.5]**

Add four (4) sheets inside the first column. Each sheet must have the following properties set:

- color : surface
- elevation : 0
- max-width : 800
- width : 100%
- margin-bottom:1

For detail information on spacing (Margins and Paddings) and border radius, reference the vuetifiy documentation at https://vuetifyjs.com/en/styles/border-radius/ and https://vuetifyjs.com/en/styles/spacing/#how-it-works

Update the first sheet as follows:

- Add ‘rounded-t-lg’ to its class attribute.
- Add card to first sheet and set its class attribute to ‘text-secondary’. Set the following properties.
  - title : “LED Controls”
  - color: surface
  - subtitle: “Recent settings”
  - variant: tonal
  - flat

Update the second sheet as follows:

- Add card inside the sheet with a padding-top of 5. Set the following properties.
  - color: surface
  - variant: tonal

- Add a Slider component inside the card and set its class attribute to ‘pt-2 bg-surface’. Set following properties.
  - append-icon: ‘mdi:mdi-car-light-high’
  - density: compact
  - thumb-size:16
  - color: secondary
  - label: ‘Brightness’
  - direction: horizontal
  - min:0
  - max:250
  - step:10
  - show-ticks
  - thumb-label: always

Update the third sheet as follows:

- Justify center and align center.
- Add card inside and set its class attribute to ‘pt-5’. Set the following properties.
  - color: surface
  - variant: tonal

- Add a Slider component inside the card and set its class attribute to ‘pt-2 bg-surface’. Set following properties.
  - append-icon: ‘mdi:mdi-led-on’
  - density: compact
  - thumb-size:16
  - color: secondary
  - label: ‘LED Nodes’
  - direction: horizontal
  - min:1
  - max:7
  - step:1
  - show-ticks
  - thumb-label: always

Update the fourth and final sheet as follows:

- Add ‘pa-2’ to its class attribute. Justify center and align center set the border property.
- Add ProgressCircular component with the following properties.
  - rotate:0
  - size:200
  - width:15

- Add a ColorPicker component to the second column.

Before proceeding, it's essential for students to familiarize themselves with Vue.js ref and reactive variables, v-model binding, watchers, and computed properties. These will be extensively utilized in this and all future lab experiments. Refer to the official documentation at https://vuejs.org/guide/essentials/reactivity-fundamentals.html , https://vuejs.org/guide/essentials/watchers.html , https://vuejs.org/guide/components/v-model.html , and https://vuejs.org/guide/essentials/computed.html to develop a fundamental understanding of working with these variables and functions.

Implement Logic **[Required Effort Level 2.0, 2.0, 2.5]**

1. This webpage will publish messages to the hardware section. Any webpage that requires MQTT functionality must import and use the MqttStore located in **`frontend/src/store/mqttStore.js`**. For guidance, revisit the 'Frontend MQTT Configuration' section available in the Appendix. Ensure you request the correct server and port information from your lab demonstrator.

Note: In order to collect the output value from any Vuejs or Vuetify component used to build the UI, they must be bound to a ref or reactive variable using the v-model directive.

2. Create a reactive variable for holding all the values collected from the added Slider and ColorPicker components. Name the variable ‘led’ and initialize it with the following default values for its parameters as seen in the code block below. Ensure you create the timer and ID JavaScript variables as well.

Control.vue

```
const led         = reactive({"brightness":255,"leds":1,"color":{ r: 255, g: 0, b: 255, a: 1 }});
let  timer, ID    = 1000;
```

3. Bind the variable to all the necessary components in the **`<template>`** section as follows.

   Update the first Slider component in the UI with the following v-model directive, binding it to the appropriate parameter of the led reactive variable.

Control.vue

```
v-model="led.brightness"
```

Update the second Slider component in the UI with the following v-model directive, binding it to the appropriate parameter of the led reactive variable.

Control.vue
```
v-model="led.leds"
```

Update the ProgressCircular component in the UI with the following. This ensures the coloured portion of the component changes with the number of active LED(s).

Control.vue
```
:model-value="led.leds *15"
```

Add the span given in the code block below inside the progress circular.

Control.vue
```
<span class="text-onSurface font-weight-bold">{{ led.leds }} LED(s)</span>
```

Update the ColorPicker component in the UI with the following. This ensures the coloured portion of the component changes with the number of active LED(s).

Control.vue
```
v-model="led.color"
```

Within the Vue.js framework, utilizing the Composition API, the watch function can be employed to execute a callback each time a reactive state undergoes a change. In this task we will use a watch function to monitor the state of the led variable. Whenever this variable changes, we want to publish a message to the hardware section with all the new values contained within.

Add the watcher function defined in the code block below to the **<script>** section.

Control.vue
```
// WATCHERS
watch(led,(controls)=>{
    clearTimeout(ID);

    ID = setTimeout(()=>{
        const message = JSON.stringify({"type":"controls","brightness":controls.brightness,"leds":controls.leds,"color":
controls.color});
        Mqtt.publish("620012345_sub",message);  // Publish to a topic subscribed to by the hardware
    },1000)
})
```

Vue.js computed properties are used to perform dynamic computations and return a result based on the values of reactive dependencies.

We aim to utilize the presently chosen color from the ColorPicker component as the color for the ProgressCircular component. The currently selected color is stored in the 'led' variable. The 'color' property of

the ProgressCircular component requires a string containing the 'rgba()' function with the desired RGB values. To achieve this, we will employ a computed property to convert the data stored in the 'led' variable into a string containing the required color format used by the ProgressCircular component. Update the `<script>` section with the code given in the code block below.

Control.vue

```
// COMPUTED PROPERTIES
const indicatorColor = computed(()=>{
     return `rgba(${led.color.r},${led.color.g},${led.color.b},${led.color.a})`
})
```

Now that we have the computed property, that will return the currently selected colour in the required format, lets bind it to the ProgressCircular component.

Update the ProgressCircular component in the UI with the following.

Control.vue

```
:color="indicatorColor"
```

End of Experiment 2