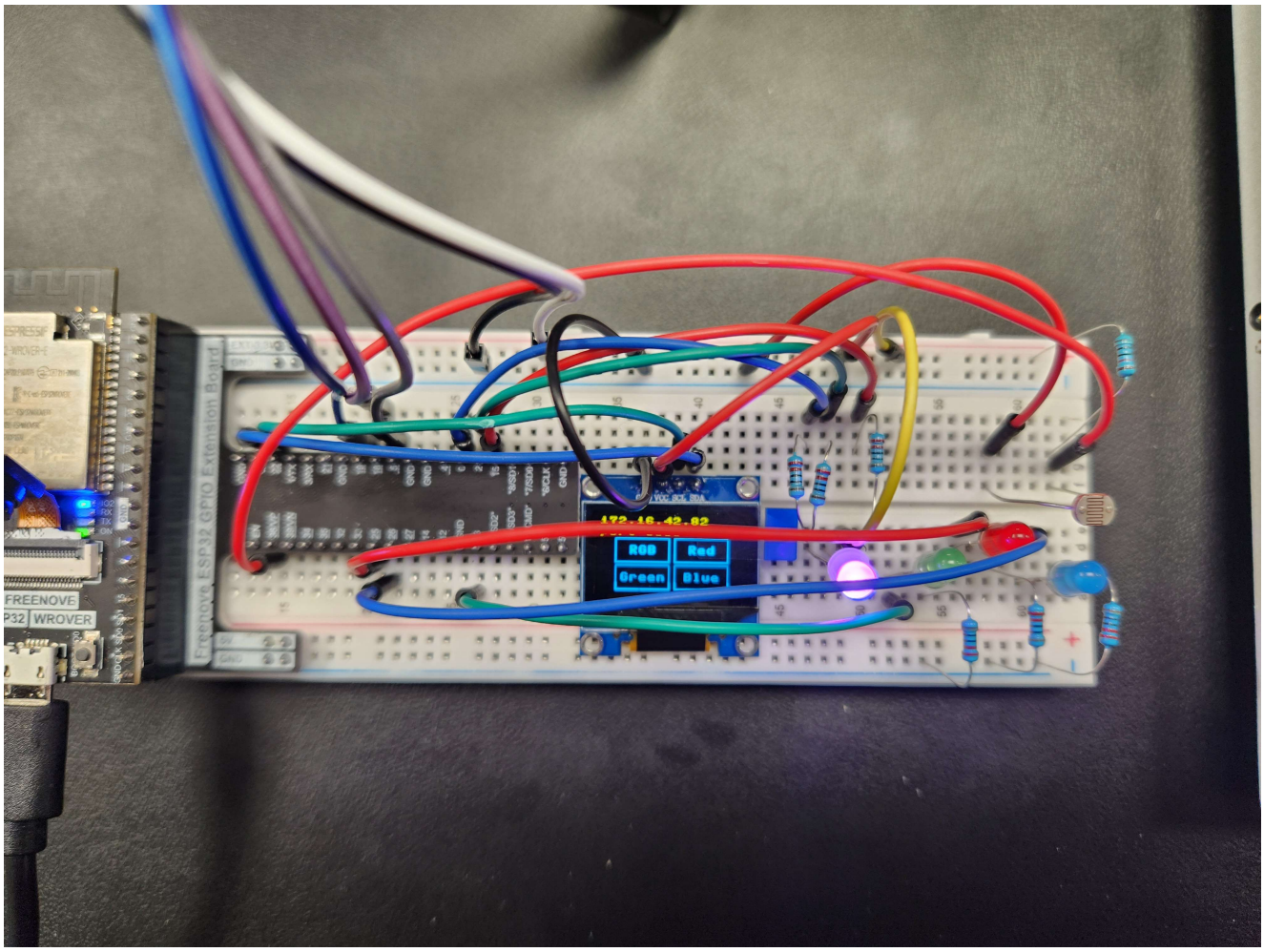
Internet Of Things Final Project

Tyler Guldberg

Steps

- Setup the following circuit:

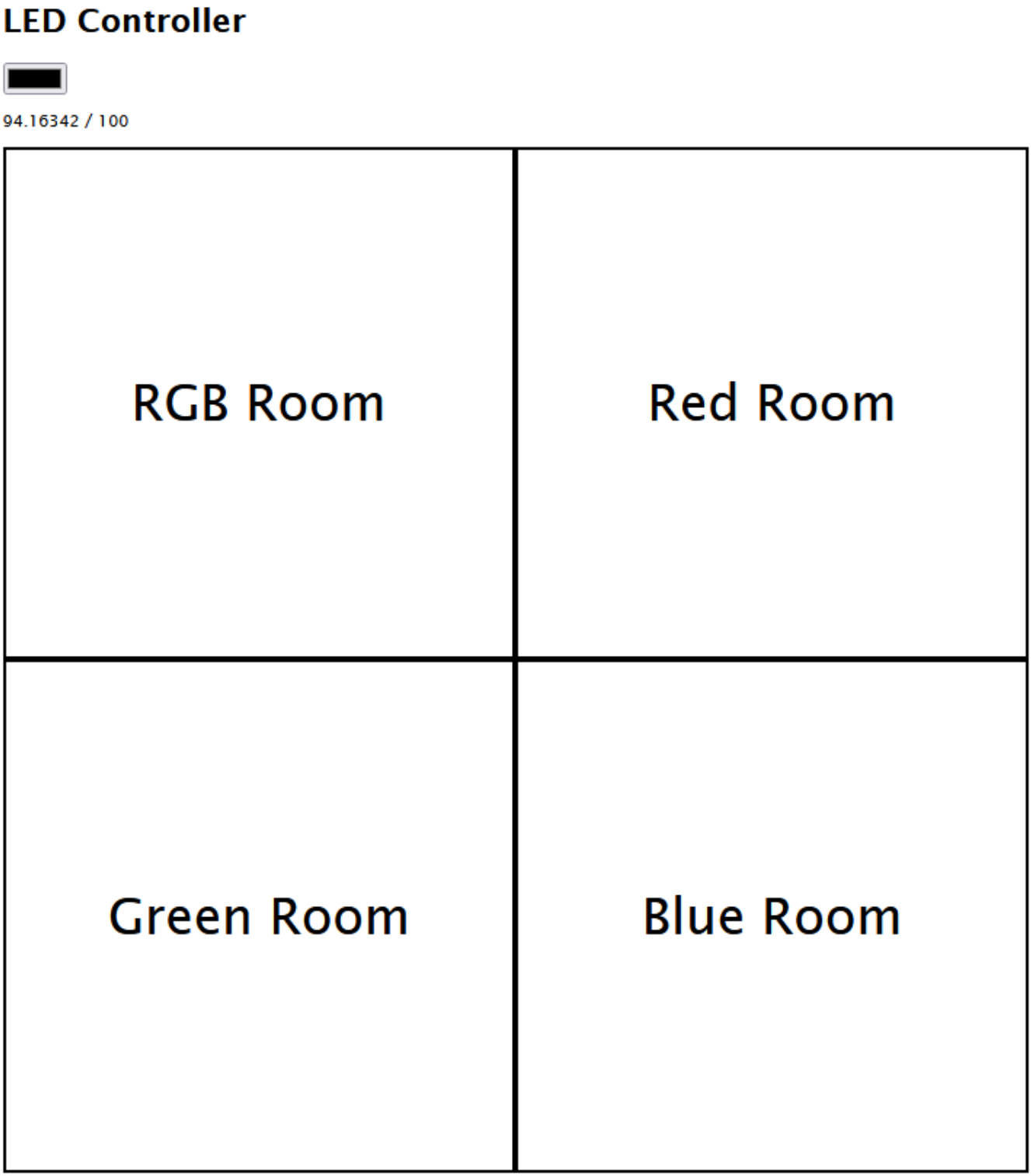


There are five major sections of this circuit

- RGB LED
- 3 Individual LEDs
- Rotary Encoder
- OLED Display
- Photoresistor

These components on their own would make a neat product, however this class is called **Internet** of Things, so an internet component will only heighten this products use!

Through the power of Websockets, we can have a webpage display *and control* the data and states of the breadboard.



A really simple webpage like above can control the lights in their entirety. Not only that, but the webpage works seamlessly with the breadboard components in tandem! Control from both the webpage or the breadboard to your hearts content.

- The bulk of this work is through websockets:

```
# a portion of main.py

@app.route('/')
async def index(request):
    ipaddr = wifi.get_ip_addr()
    with open('index.html', 'r') as f:
        text = f.read().replace('ADDRESS', f'{ipaddr}:{port}')
        return text, {'Content-Type': 'text/html'}

@app.route('/slider')
@with_websocket
async def slider(request, ws):
    global values

    def swap_state(pin):
        """Changes the rotary encoder's interrupt function based on a boolean
        representing which 'state' it is in. The boolean is swapped each time it is ran"""

        global state_bool
        global selected_line
        global box_select

        if selected_line % 4 == 3:
            box_select = 0
            display.led_rooms(led_values)
            display.buffer.rect(16, 18, 44, 18, 1, False)
            display.show()
            state_bool = not state_bool
            rotary.callback = box_scroll
            switch.callback = select
        elif state_bool:
            display.rgb_display.show_selected(selected_line % 4)
            rotary.callback = change_color
        elif not state_bool:
            display.rgb_display.refresh()
            display.rgb_display.line_select(selected_line)
            rotary.callback = select_color

        state_bool = not state_bool

    def change_color(value):
        """Modifies the value selected LED"""

        global selected_line
        selected_line %= 4

        values[selected_line] += value * 75

        if values[selected_line] <= 0:
            values[selected_line] = 0
        elif values[selected_line] >= 1023:
            values[selected_line] = 1023
```

```
        display.rgb_display.update_values(values)
        display.rgb_display.value_loop()
        display.rgb_display.show_selected(selected_line)
        RGB_LED.set_color(values)
        led_values[0] = duty_to_rgb(values)
        asyncio.run(ws.send(json.dumps(led_values)))

    def select(pin):
        global box_select
        global selected_line
        global send_data

        box_select %= 4

        if box_select == 0:
            selected_line = 0
            display.rgb_selection(LABELS, values)
            switch.callback = swap_state
            rotary.callback = select_color
        else:
            led_values[box_select] = not led_values[box_select]
            leds[box_select - 1].value(led_values[box_select])

            display.clear()
            display.led_rooms(led_values)
            display.buffer.rect(x_y_pair[box_select][0], x_y_pair[box_select][1], 44, 18, not led_value
            display.show()
            asyncio.run(ws.send(json.dumps(led_values)))

    switch = DebouncedSwitch(5, select)

    def duty_to_rgb(values):
        converted = [0, 0, 0]

        for i in range(len(values)):
            converted[i] = round((values[i]/1023)*255)

        return converted

    while True:
        value = await ws.receive()
        value = json.loads(value)
        try:
            led_values[value['led']] = not led_values[value['led']]
            leds[value['led'] - 1].value(led_values[value['led']])
        except:
            values = hex_to_rgb(value['rgb'])
            RGB_LED.set_color(values)
            display.rgb_display.update_values(values)
        update_display()

@app.route('/ldr')
@with_websocket
async def ldr(request, ws):
    LDR = ADC(Pin(36))
    LDR.atten(ADC.ATTN_11DB)
```

```
    LDR.width(ADC.WIDTH_10BIT)
    value = LDR.read()
    asyncio.run(ws.send(str(100 - (100 * value / 1028))))
    time.sleep(2)
    ldr(request, ws)
```

Websockets allow two way communication between the board and the webpage. This enables both 'clients' to work in tandem.

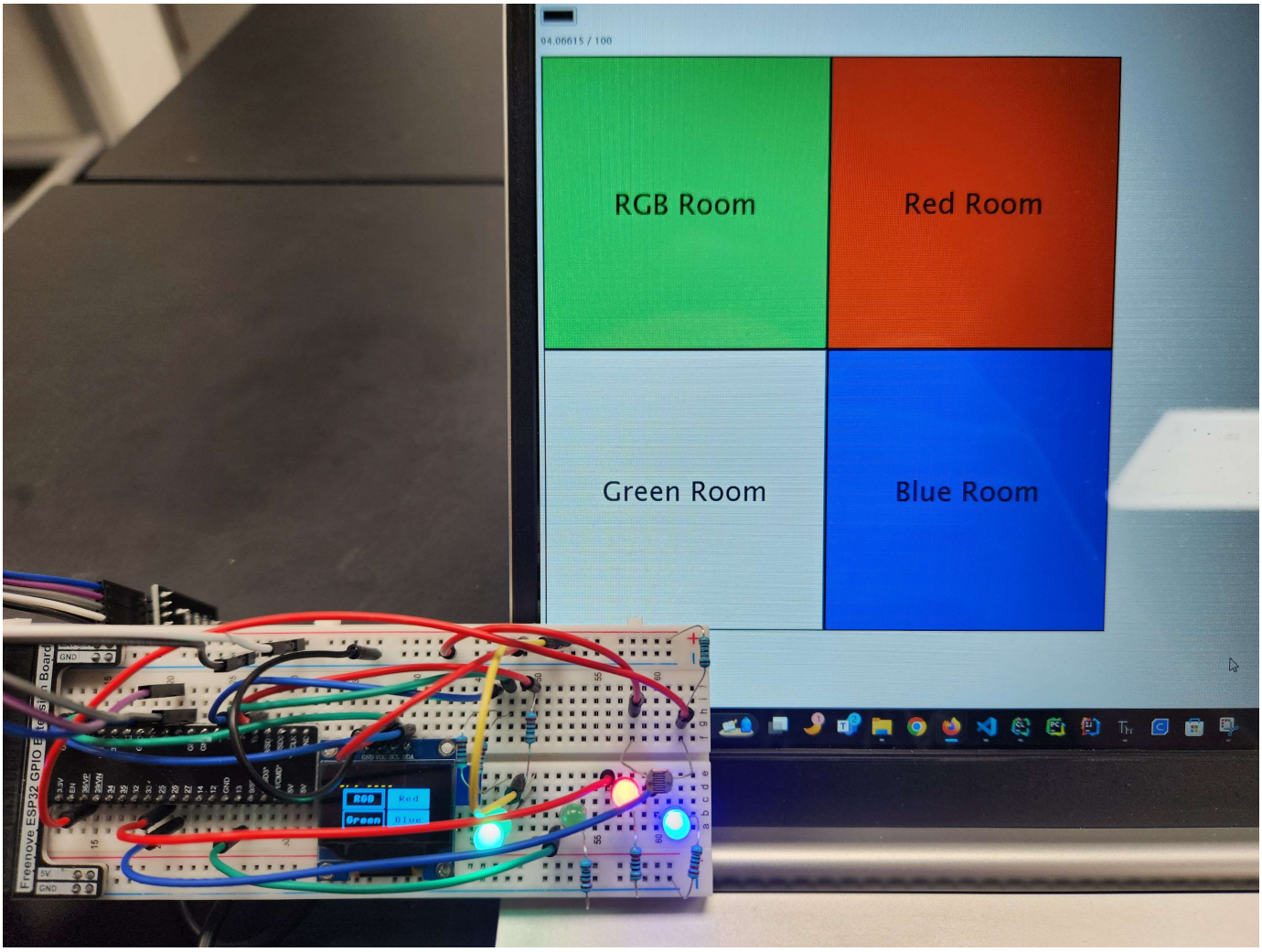
```
<script>
    const ip = 'ADDRESS';
    window.addEventListener("load", function () {
        let ws = new WebSocket(`ws://${ip}/slider`);
        let photoWs = new WebSocket(`ws://${ip}/ldr`)

        rooms = [document.getElementById('RedRoom'), document.getElementById('GreenRoom'), document.get
        roomColors = ['#ff0000', '#00ff00', '#0000ff']
        roomBools = [false, false, false]

        ws.onmessage = (event) => {
            data = JSON.parse(event.data); // String containing states of LEDs and RGB LED color in he
            document.getElementById("RGBroom").style.background = 'rgb(' + String(data[0]) + ')';
            document.getElementById("colorPicker").value = 'rgb(120, 76, 200)';

            for(let i = 0; i < data.length; i++) {
                if(i > 0) {
                    if(data[i]) {
                        rooms[i - 1].style.background = roomColors[i - 1];
                    }
                    else {
                        rooms[i - 1].style.background = '#ffffff';
                    }
                    roomBools[i] = data[i];
                    roomUpdate(rooms[i - 1], roomBools[i], roomColors[i - 1]);
                }
            }
        }

        photoWs.onmessage = (event) => {
            data = JSON.parse(event.data);
            document.getElementById("LDRStatusLabel").innerText = data + ' / 100';
        }

        ws.onclose = (event) => {
            console.log("websocket closed");
        }
        ws.onerror = (event) => {
            console.log("websocket error: ", event);
        }

        function roomUpdate(room, roomBool, roomColor) {
            if(roomBool) {
```

```
                room.style.background = roomColor;
            }
            else {
                room.style.background = '#ffffff';
            }
        };

        rooms[0].addEventListener("click", function () {
            data = JSON.stringify({ led: 1 });
            roomBools[1] = !roomBools[1];
            roomUpdate(rooms[0], roomBools[1], roomColors[0]);
            ws.send(data); // Send back id of the LED that was clicked
            console.log("red clicked");
        });

        rooms[1].addEventListener("click", function () {
            data = JSON.stringify({ led: 2 });
            roomBools[2] = !roomBools[2];
            roomUpdate(rooms[1], roomBools[2], roomColors[1]);
            ws.send(data);
            console.log("green clicked");
        });

        blueRoom = document.getElementById("BlueRoom");
        rooms[2].addEventListener("click", function () {
            data = JSON.stringify({ led: 3 });
            roomBools[3] = !roomBools[3];
            roomUpdate(rooms[2], roomBools[3], roomColors[2]);
            ws.send(data);
            console.log("blue clicked");
        });

        colorPicker = document.getElementById("colorPicker");
        colorPicker.addEventListener("change", function () {
            document.getElementById("RGBroom").style.background = colorPicker.value;
            console.log(colorPicker.value);
            data = JSON.stringify({ rgb: colorPicker.value });
            ws.send(data);
            console.log("rgb changed");
        });
    });

</script>
```

- This is all of the logic that sends and receives data to and from the board.

The bulk of this product is having two way communication using Websockets and demonstrating different applications of this concept (in this case with individual LEDs, a RGB LED that is synchronized, and full control

of them all from two locations at once!).



And this shows the synchronization of the board and webpage.

FinalProject.zip

All of the necessary files are in this zip file. Simply download it, upload it all to ESP32, and hit run!

Do you need to have control over your lights all of the time? Of course you do, and with this neat little product, you can have all the control one could ask for!

(In all seriousness, I do wish I had the time to refactor a great deal of this code. Namely, making a menu class to handle callback functions and menu navigation with the rotary encoder which would clean up `main.py` *a ton. Also breaking up each device 'section' into different Websockets to greatly simplify the data that gets sent to and from.)*