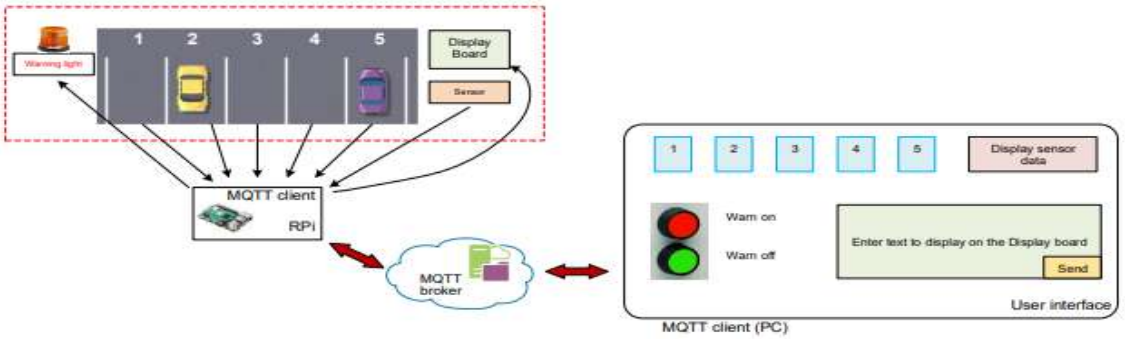
Project Description

Introduction:

- In this module, we will design a simple IoT-based smart parking system. In deliverable 1, we designed an edge node. In deliverable 2, we established a full duplex communication between the edge node and the lab PC using the MQTT protocol. Now, we will design a user interface(UI) that will be combined with the previous designs, and put the framework into a practical application, like smart parking.

Parking Lot Description:

- The parking lot consists of the following equipment:
    1. A warning light: PA will activate it when there is an emergency. When activated, the light will flash continuously.
    2. A sensor: reads some data, e.g. temp, motion, air, etc.
    3. A display board: displays any message to the public/visitor as sent by the PA.

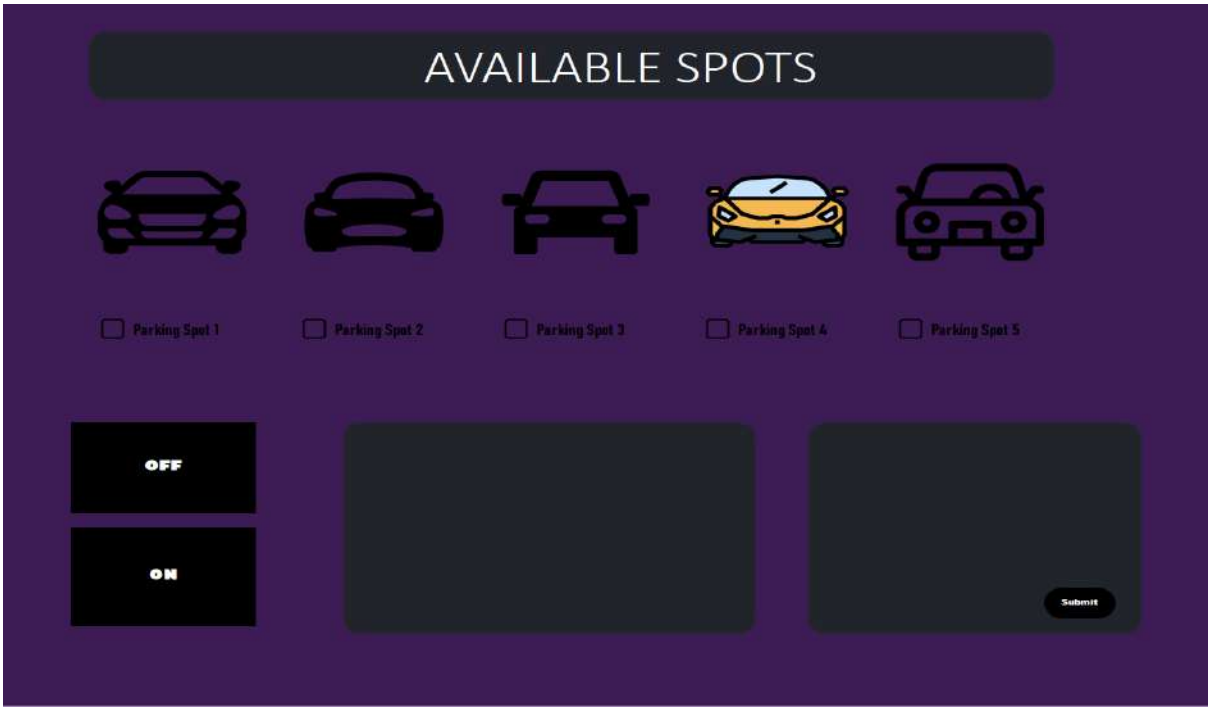


UI description:

- The design requirements for this deliverable are provided below:
    1. Parking spot indicator: It could be five lights (when a specific spot is empty, it should show green, otherwise red), or five checkboxes(when a specific spot is empty, it should be checked, otherwise unchecked), or some other way to indicate which one is occupied.
    2. Two buttons to control the warning light: When the “ON” button is pressed, the warning light on the ground must flash; when the “OFF” button is pressed, it is turned off.
    3. A test box or display box to show the outside sensor data(as you receive it continuously from the sensor)
    4. A test box or window where the PA can display a message on the display board. When the “Send” button is pressed, the message will be displayed on the board(for demo purposes, just print the message on RPI’s terminal).

Part 1) Basic UI

- In this Deliverable, For the 1st part to create the UI, I have used QT designer in order to design the UI. This UI consists of the terminal which prints the message to show the outside data. It consists of a text box that prints data to be displayed on the display board. An ON and OFF buttons help turn the warning lights on and off in the ground and also one more feature is of identifying which parking lot is full which is done with the help of a checkbox. The representation of the UI is shown below: -



- The basic UI generated with the help of QT design saved in a file with extension .ui which I had to convert to .py with a command in the terminal which is provided below:

```
PS C:\Users\Pal\Desktop\pal_del3>
C:\Users\Pal\AppData\Local\Packages\PythonSoftwa
reFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache
\local-packages\Python39\Scripts\pyuic5.exe
c:\Users\Pal\Desktop\pal_del3\pal_del3.ui -o
c:\Users\Pal\Desktop\pal_del3\pal_del3.py
```

- On performing this command I got the UI file but was not able to get the images in the python file. In order to get these images I had to convert my qrc file to a py file too. This was done by the command shown below:

```
PS C:\Users\Pal\Desktop\pal_del3>
C:\Users\Pal\AppData\Local\Packages\PythonSoftwareFoundation.Pytho
n.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\Scripts\pyrc
```

```
c5.exe C:\Users\Pal\Downloads\test.qrc\test.qrc -o
C:\Users\Pal\Downloads\test.qrc\test_rc.py
```

Part 2) UI code:

- This code was initially generated with the help of QT designer which was then converted into a python file. I have also added CSS in order to showcase that the buttons, textboxes, checkboxes, labels, and frames appear in a good way. An example of this code is shown below:
    1. The code for the main window is provided below with CSS: -

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1380, 957)
        font = QtGui.QFont()
        font.setFamily("Calibri")
        font.setBold(True)
        font.setItalic(False)
        font.setWeight(75)
        MainWindow.setFont(font)
        self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setStyleSheet("background-color:rgb(61,
28, 85)")
        self.centralwidget.setObjectName("centralwidget")
```

```
        self.Warning_On =
QtWidgets.QPushButton(self.centralwidget)
        self.Warning_On.setGeometry(QtCore.QRect(80, 700,
211, 131))
        self.Warning_On.setStyleSheet("QPushButton{\n"
"    border: none;\n"
"    background-color:rgb(0, 0, 0);\n"
"    color: rgb(255, 255, 255);\n"
"     font: 8pt \"MS Reference Specialty\";\n"
"     font: 14pt \"Gill Sans Ultra Bold\";\n"
"    border-left: 1px solid rgb(0, 0, 0);\n"
"    border-right:1px solid rgb(0, 0, 0);\n"
"    border-bottom: 5px solid rgb(0, 0, 0);\n"
"}\n"
```

2. Code for PushButton with CSS: -

```
"\n"
"QPushButton:hover{\n"
"    border: none;\n"
"    background-color:rgb(160, 160, 245);\n"
"    border-left: 1px solid rgb(160, 160, 245);\n"
"    border-right:1px solid rgb(160, 160, 245);\n"
"    border-bottom: 5px solid rgb(160, 160, 245);\n"
"  \n"
"}")
        self.Warning_On.setObjectName("Warning_On")
        self.Warning_Off =
QtWidgets.QPushButton(self.centralwidget)
```

```
        self.Warning_Off.setGeometry(QtCore.QRect(80, 560,
211, 121))
        self.Warning_Off.setStyleSheet("QPushButton{\n"
"    border: none;\n"
"    background-color:rgb(0, 0, 0);\n"
"    color: rgb(255, 255, 255);\n"
"     font: 8pt \"MS Reference Specialty\";\n"
"     font: 14pt \"Gill Sans Ultra Bold\";\n"
"    border-left: 1px solid rgb(0, 0, 0);\n"
"    border-right:1px solid rgb(0, 0, 0);\n"
"    border-bottom: 5px solid rgb(0, 0, 0);\n"
"}\n"
"\n"
```

3. Code for TextEdit which is the CSS: -

```
"QTextEdit:hover {\n"
"  border: 2px solid rgb(143, 142, 141);\n"
"}\n"
"\n"
"QTextEdit:focus{\n"
"  border: 2px rgb(143, 142, 141);\n"
"}")
        self.Send_Command.setObjectName("Send_Command")
        self.Submit =
QtWidgets.QPushButton(self.centralwidget)
        self.Submit.setGeometry(QtCore.QRect(1190, 780, 81,
41))
        self.Submit.setStyleSheet("QPushButton{\n"
"    border: none;\n"
```

```
"    background-color:rgb(0, 0, 0);\n"
"      border-radius: 20px;\n"
"    color: rgb(255, 255, 255);\n"
"     font: 87 8pt \"Arial Black\";\n"
"    border-left: 1px solid rgb(0, 0, 0);\n"
"    border-right:1px solid rgb(0, 0, 0);\n"
"    border-bottom: 5px solid rgb(0, 0, 0);\n"
"}\n"
```

4. Code for Frames and adding images: -

```
     self.Spot_1 = QtWidgets.QFrame(self.centralwidget)
        self.Spot_1.setGeometry(QtCore.QRect(110, 170, 171,
221))
        self.Spot_1.setStyleSheet("QFrame{\n"
"image: url(:/icon/sedan-car-front.png);}")

self.Spot_1.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.Spot_1.setFrameShadow(QtWidgets.QFrame.Raised)
        self.Spot_1.setObjectName("Spot_1")
        self.Spot_2 = QtWidgets.QFrame(self.centralwidget)
        self.Spot_2.setGeometry(QtCore.QRect(340, 170, 171,
221))
        self.Spot_2.setStyleSheet("\n"
"image: url(:/icon/car.png);")

self.Spot_2.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.Spot_2.setFrameShadow(QtWidgets.QFrame.Raised)
        self.Spot_2.setObjectName("Spot_2")
        self.Spot_3 = QtWidgets.QFrame(self.centralwidget)
```

```
        self.Spot_3.setGeometry(QtCore.QRect(570, 170, 171,
221))
        self.Spot_3.setStyleSheet("image: url(:/icon/car
(1).png);")

self.Spot_3.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.Spot_3.setFrameShadow(QtWidgets.QFrame.Raised)
        self.Spot_3.setObjectName("Spot_3")
        self.Spot_4 = QtWidgets.QFrame(self.centralwidget)
        self.Spot_4.setGeometry(QtCore.QRect(800, 170, 171,
221))
        self.Spot_4.setStyleSheet("image: url(:/icon/car
(2).png);")

self.Spot_4.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.Spot_4.setFrameShadow(QtWidgets.QFrame.Raised)
        self.Spot_4.setObjectName("Spot_4")
        self.Spot_5 = QtWidgets.QFrame(self.centralwidget)
        self.Spot_5.setGeometry(QtCore.QRect(1020, 170, 171,
221))
        self.Spot_5.setStyleSheet("image:
url(:/icon/drive.png);")

self.Spot_5.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.Spot_5.setFrameShadow(QtWidgets.QFrame.Raised)
        self.Spot_5.setObjectName("Spot_5")
        self.checkBox =
QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox.setGeometry(QtCore.QRect(110, 420, 171,
31))
        font = QtGui.QFont()
        font.setFamily("Bahnschrift Condensed")
```

```
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.checkBox.setFont(font)
        self.checkBox.setFocusPolicy(QtCore.Qt.StrongFocus)
        self.checkBox.setStyleSheet("QCheckBox::indicator{\n"
```

5. Code for CheckBox with CSS: -

```
"QCheckBox::hover{\n"
"color: rgb(255, 255, 0);\n"
"}\n"
"\n"
"QCheckBox::indicator:unchecked{\n"
"image: url(:/newPrefix/unchecked.png);\n"
"}\n"
"\n"
"QCheckBox::indicatior:unchecked,
QCheckBox::indicator:unchecked:pressed{\n"
"image: url(:/newPrefix/check.png);\n"
"}\n"
"\n"
"QCheckBox::indicator:checked{\n"
"image: url(:/newPrefix/check.png);\n"
"}")
        self.checkBox.setObjectName("checkBox")
        self.checkBox_6 =
QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox_6.setGeometry(QtCore.QRect(1020, 420,
171, 31))
```

```
        font = QtGui.QFont()
        font.setFamily("Bahnschrift Condensed")
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.checkBox_6.setFont(font)
        self.checkBox_6.setFocusPolicy(QtCore.Qt.StrongFocus)
```

Part 3) Communication part: -

- For the communication part, This deliverable requires us to communicate both ways. This is because it needs to send the command to turn on the warning lights to the ground. It also requires displaying some readings to the terminal of the user and it also requires updating if a car is present in the particular parking spot or not. For this, I had to create two different topics for subscription and publish them in both publisher and subscriber files. For both of these, I have used the topics pub_topic and sub_topic. The screenshots of the codes are shown below:
- For Publisher: -
    1. Two different topics: -

```
pub_topic = "test/publisher"
sub_topic = "test/subscriber"
```

    2. MainWindow calls the UI and also consists of the different buttons function as well as the main branch to turn the warning lights on and off which is done by self.PushButton.clicked.connect(): -

```
class MainWindow(QtWidgets.QMainWindow,
pal_del3.Ui_MainWIndow)

   def __init__(self):

       super(MainWindow, self).__init__()

       self.setupUi(self)


self.Warning_On.clicked.connect(self.Warning_ON)


self.Warning_Off.clicked.connect(self.Warning_OFF)


self.Submit.clicked.connect(self.Submit_message)

       self.Terminal.setText(self.subscribe(client))
```

3. Function to turn ON and turn OFF warning lights (this uses pub_topic = “test/publisher” as a topic to publish to the subscriber): -

```
def Warning_ON(self):

           msg="RL_ON"

           client.publish(pub_topic, msg, qos=0,
retain=False)

           print(msg)


   def Warning_OFF(self):

           msg="GL_ON"

           client.publish(pub_topic, msg, qos=0,
retain=False)

           print(msg)
```

4. Function to submit message: -

```
   def Submit_message(self):

          msg = self.Send_Command.toPlainText();

          client.publish(pub_topic, msg, qos=0,
retain=False)

          print(msg)
```

5. Function to connect to the MQTT client: -

```
def on_connect(client, userdata, flags, rc):

      if rc == 0:

             print("Connected to MQTT Broker!")

             client.subscribe(sub_topic)

      else:

             print("Failed to connect, return code
%d\n", rc)


   def connect_mqtt() -> mqtt_client:

      return client
```

6. Function to subscribe, which means to get the data from the subscriber in order to perform changes to identify the parking spot numbers. This is done by checking the checkbox if a car is parked on the spot or not. It is also used to display values in the terminal, values such as humidity and temperature. This part also tells us that whenever key1 is pressed from the keypad the subscribe function gets the message and it splits it into temperature, spot and humidity. If the key pressed is 1, 2, 3, 4, 5 then the parking spot1, parking spot2, parking spot3, parking spot4, parking spot5 is filled with a checkbox getting checked. Similarly

pressing key 6,7,8,9,0 clears parking spot1, parking spot2, parking spot3, parking spot4, parking spot5 in the same manner. If we press C it clears all the parking spots.

```
def subscribe(self,client: mqtt_client):
    def on_message(client, userdata, msg):
     b = (msg.payload.decode('utf-8'))
     spot = b.split(",")[0]
     temp = b.split(",")[2]
     hum = b.split(",")[1]
     times_1 = 0


     if spot == "1:
             self.checkBox.setChecked(True)
     elif spot == "2":
             self.checkBox_9.setChecked(True)
     elif spot == "3":
             self.checkBox_8.setChecked(True)
     elif spot == "4":
             self.checkBox_7.setChecked(True)
     elif spot == "5":
             self.checkBox_6.setChecked(True)
     elif spot == "6":
           self.checkBox.setChecked(False)
     elif spot ==  "7":
             self.checkBox_9.setChecked(False)
     elif spot == "8":
             self.checkBox_8.setChecked(False)
     elif spot == "9":
```

```
                self.checkBox_7.setChecked(False)
        elif spot == "0":
                self.checkBox_6.setChecked(False)
        elif spot == "C":
                self.checkBox_6.setChecked(False)
                self.checkBox_7.setChecked(False)
                self.checkBox_8.setChecked(False)
                self.checkBox_9.setChecked(False)
                self.checkBox.setChecked(False)

        mainWindow.Terminal.setText("Temperature:
"+temp+" Humidity: "+hum)
        print(b)

      client.loop_start()

      client.subscribe(sub_topic, qos=2)
      client.on_message = on_message
```

7. This publisher also consists of the main function which calls everything and consists of a broker, client_id, etc.: -

```
if __name__ == "__main__":
mqttBroker ="broker.hivemq.com"
client_id = "fake_temp2"
client = mqtt_client.Client(client_id)  # Client Name
client.connect(mqttBroker)
client.loop_start()
```

```
app = QtWidgets.QApplication(sys.argv)

mainWindow = MainWindow()

mainWindow.show()

sys.exit(app.exec_())
```

- For Subscriber: -
    1. For the first part the subscriber consists of the setup() part which is used to set up the GPIO pins, as it is used in compliance with the raspberry pi. The code is shown below: -

```
# set up the GPIO pins

def setup():

   GPIO.setmode(GPIO.BCM)

   GPIO.setwarnings(False)

   GPIO.setup(23, GPIO.OUT) #GREEN LED

   GPIO.setup(18, GPIO.OUT) #RED LED

   #GPIO.setup(19, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Set BtnPin's mode is input, and pull up to high
level(3.3V)

   #GPIO.add_event_detect(19, GPIO.BOTH,
callback=detect, bouncetime=200)
```

    2. Now, the difference between deliverable 2 subscribe and deliverable 3 subscribe is that I added a keypad in order to take in the values of the parking spot numbers. This keypad is used as a 2-d array. The code for the keypad is shown below: -

```
# Define the pins used for the keypad

ROW = [12, 16, 20, 21] #
```

```
COL = [6, 13, 19, 26] #


# Define the keypad mapping
KEYPAD = [
    ["1", "2", "3", "A"],
    ["4", "5", "6", "B"],
    ["7", "8", "9", "C"],
    ["*", "0", "#", "D"]
]


# Initialize the GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
for pin in ROW:
    GPIO.setup(pin, GPIO.OUT)
for pin in COL:
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)


# Function to read the keypad
def get_key():
    for row_pin in ROW:
        GPIO.output(row_pin, 0)
        for col_pin in COL:
            if GPIO.input(col_pin) == 0:
                key =
KEYPAD[ROW.index(row_pin)][COL.index(col_pin)]
                while GPIO.input(col_pin) == 0:
                    pass
```

```
                GPIO.output(row_pin, 1)
                return key
        GPIO.output(row_pin, 1)
    return None

def readRow(line, characters):
    GPIO.output(line, GPIO.LOW)
    if(GPIO.input(COL_1) == GPIO.LOW):
        return (characters[0])
    if(GPIO.input(COL_2) == GPIO.LOW):
        return (characters[1])
    if(GPIO.input(COL_3) == GPIO.LOW):
        return (characters[2])
    if(GPIO.input(COL_4) == GPIO.LOW):
        return (characters[3])
    GPIO.output(line, GPIO.HIGH)
```

3. The code for reading temperature and humidity is the same as before and shown below: -

```
# function to read the temperature and humidity from
DHT11

MAX_UNCHANGE_COUNT = 100

STATE_INIT_PULL_DOWN = 1
```

```
STATE_INIT_PULL_UP = 2

STATE_DATA_FIRST_PULL_DOWN = 3

STATE_DATA_PULL_UP = 4

STATE_DATA_PULL_DOWN = 5


def read_dht11_dat():
    GPIO.setup(25, GPIO.OUT)
    GPIO.output(25, GPIO.HIGH)
    time.sleep(0.05)
    GPIO.output(25, GPIO.LOW)
    time.sleep(0.02)
    GPIO.setup(25, GPIO.IN, GPIO.PUD_UP)


    unchanged_count = 0
    last = -1
    data = []
    while True:
        current = GPIO.input(25)
        data.append(current)
        if last != current:
            unchanged_count = 0
            last = current
        else:
            unchanged_count += 1
            if unchanged_count > MAX_UNCHANGE_COUNT:
                break


    state = STATE_INIT_PULL_DOWN
```

```
lengths = []
current_length = 0


for current in data:
    current_length += 1


    if state == STATE_INIT_PULL_DOWN:
        if current == GPIO.LOW:
            state = STATE_INIT_PULL_UP
        else:
            continue
    if state == STATE_INIT_PULL_UP:
        if current == GPIO.HIGH:
            state = STATE_DATA_FIRST_PULL_DOWN
        else:
            continue
    if state == STATE_DATA_FIRST_PULL_DOWN:
        if current == GPIO.LOW:
            state = STATE_DATA_PULL_UP
        else:
            continue
    if state == STATE_DATA_PULL_UP:
        if current == GPIO.HIGH:
            current_length = 0
            state = STATE_DATA_PULL_DOWN
        else:
            continue
```

```
        if state == STATE_DATA_PULL_DOWN:
            if current == GPIO.LOW:
                lengths.append(current_length)
                state = STATE_DATA_PULL_UP
            else:
                continue
    if len(lengths) != 40:
        #print ("Data not good, skip")
        return False

    shortest_pull_up = min(lengths)
    longest_pull_up = max(lengths)
    halfway = (longest_pull_up + shortest_pull_up) / 2
    bits = []
    the_bytes = []
    byte = 0

    for length in lengths:
        bit = 0
        if length > halfway:
            bit = 1
        bits.append(bit)
    #print ("bits: %s, length: %d" % (bits, len(bits)))
    for i in range(0, len(bits)):
        byte = byte << 1
        if (bits[i]):
            byte = byte | 1
        else:
```

```
            byte = byte | 0
        if ((i + 1) % 8 == 0):
            the_bytes.append(byte)
            byte = 0
    #print (the_bytes)
    checksum = (the_bytes[0] + the_bytes[1] +
the_bytes[2] + the_bytes[3]) & 0xFF
    if the_bytes[4] != checksum:
        #print ("Data not good, skip")
        return False

    return the_bytes[0], the_bytes[2]
```

4. Also, there is a change in the detect() function from the previous deliverable. As we added a keypad we also need to detect the pressed key. Therefore the detect() function is updated in such a manner that it returns the key and only if a key is pressed the temperature and humidity is read. This function returns a value in a string as it needs to be sent to the publisher to update the parking spot and print the temperature and humidity in the terminal(textbox) of the UI. The code is provided below: -

```
def detect():
    key = get_key()
    pressed_key = ""
    if key in ["1","2","3","4","5", "6", "7", "8", "9",
"0", "C"]:
        pressed_key = key
        result = read_dht11_dat()
        if result:
```

```
            humidity, temperature = result
            return
f"{pressed_key},{humidity},{temperature}"
```

5. Now, the function remains the same as used before for turning on, and off LEDs. The code is shown below: -

```
def destroy():
    GPIO.output(23, GPIO.LOW)       # Green led off
    GPIO.output(18, GPIO.LOW)       # Red led off
    #GPIO.cleanup()
def green():
    GPIO.output(23, GPIO.HIGH)      #
    GPIO.output(18, GPIO.LOW)       #
def Red():
    GPIO.output(23, GPIO.LOW)      #
    GPIO.output(18, GPIO.HIGH)
```

6. Similar to publishers, there are two different topics used to send the data and receive the data. pub_top is used to subscribe to subscribers and sub_topic is used to publish to the publisher from the subscriber.

```
pub_topic = "test/publisher"
sub_topic = "test/subscriber"
mqttBroker ="broker.hivemq.com"
```

7. functions to connect to the client as well as subscribe remained the same as the last deliverable to get the commands from the publisher. The code is provided below: -

```
def on_connect (client, userdata, flags,  rc):
   print("connected" + str(rc))


def subscribe(client):
   def on_message(client, userdata, message):
       if (message.payload.decode('utf-8') ==
"GL_ON"):
           print("Turning green led on...")
           setup()
           green()
       elif (message.payload.decode('utf-8') ==
"RL_ON"):
           print("Turning Red led on...")
           setup()
           Red()
       elif (message.payload.decode('utf-8') ==
"LD_OFF"):
           print("Turning off led...")
           setup()
           destroy()
       elif (message.payload.decode('utf-8') ==
"OFF"):
```

```
            print("Turning OFF humiture sensor...")
            setup()
            destroy()
        else:
            print("MESSAGE FROM TEXT BOX:
"+message.payload.decode())
```

8. The code which provides that client subscribes to pub_topic and a loop which detects and publishes the received values of keys, temperature, and humidity publishes it to the publisher is shown below: -

```
    client.subscribe(pub_topic,  qos=0)
    client.on_message = on_message

client = mqtt.Client("Asuj_Patel")
client.connect(mqttBroker)



client.on_connect = on_connect
subscribe(client)




while True:
    mssg = detect()
    print(mssg)
```

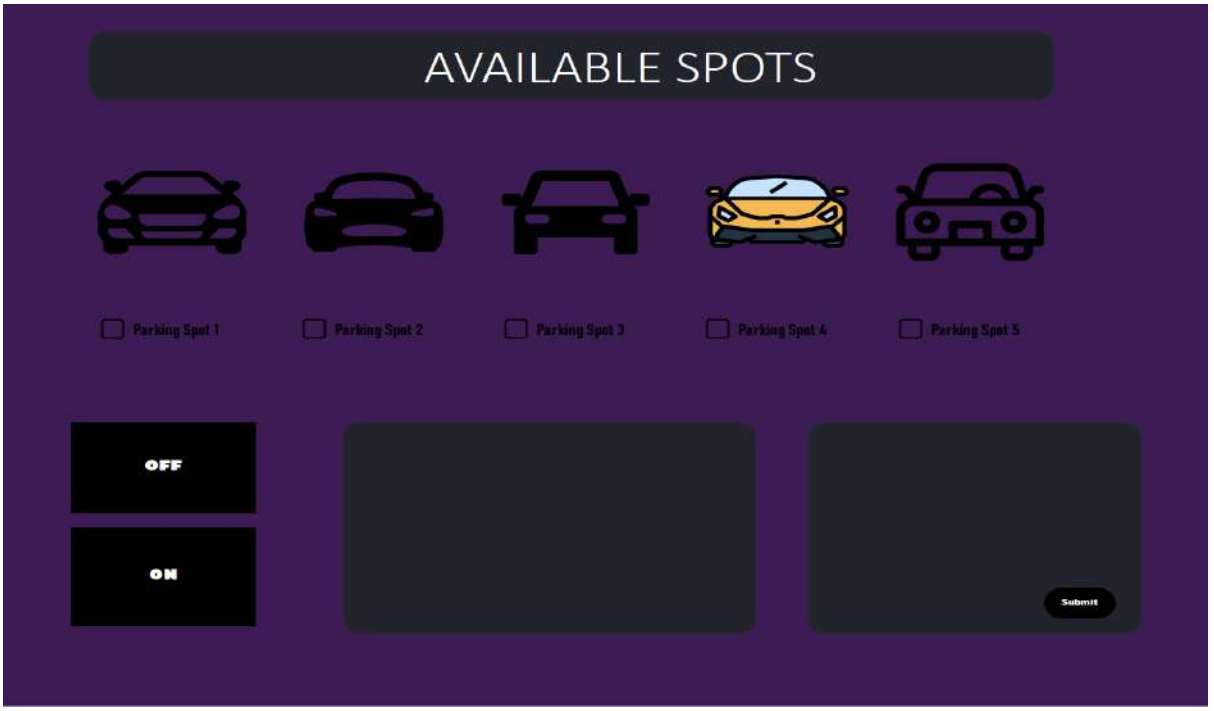
```
    client.publish(sub_topic, mssg)
    time.sleep(2)
    client.loop()


client.loop_forever()
```

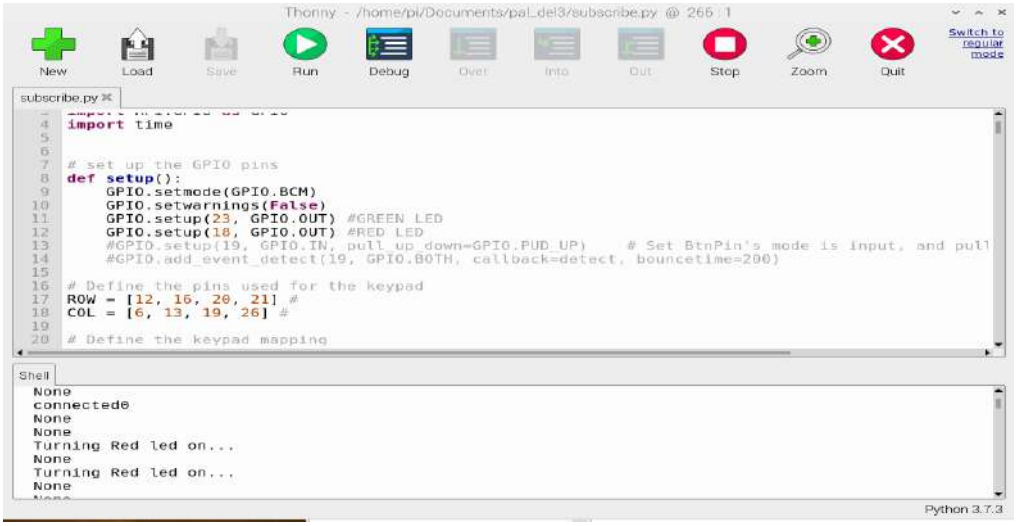
- Overall the whole code for the subscriber, publisher, and the code UI is provided above. Combining all these provides me with a real-time working parking lot system. Dual communication is the best thing to learn from this deliverable for me. It was easily implemented just by changing the topic. Learning how to make a UI and merging the code with the publisher and subscriber was also a great way to understand IoT in real time.

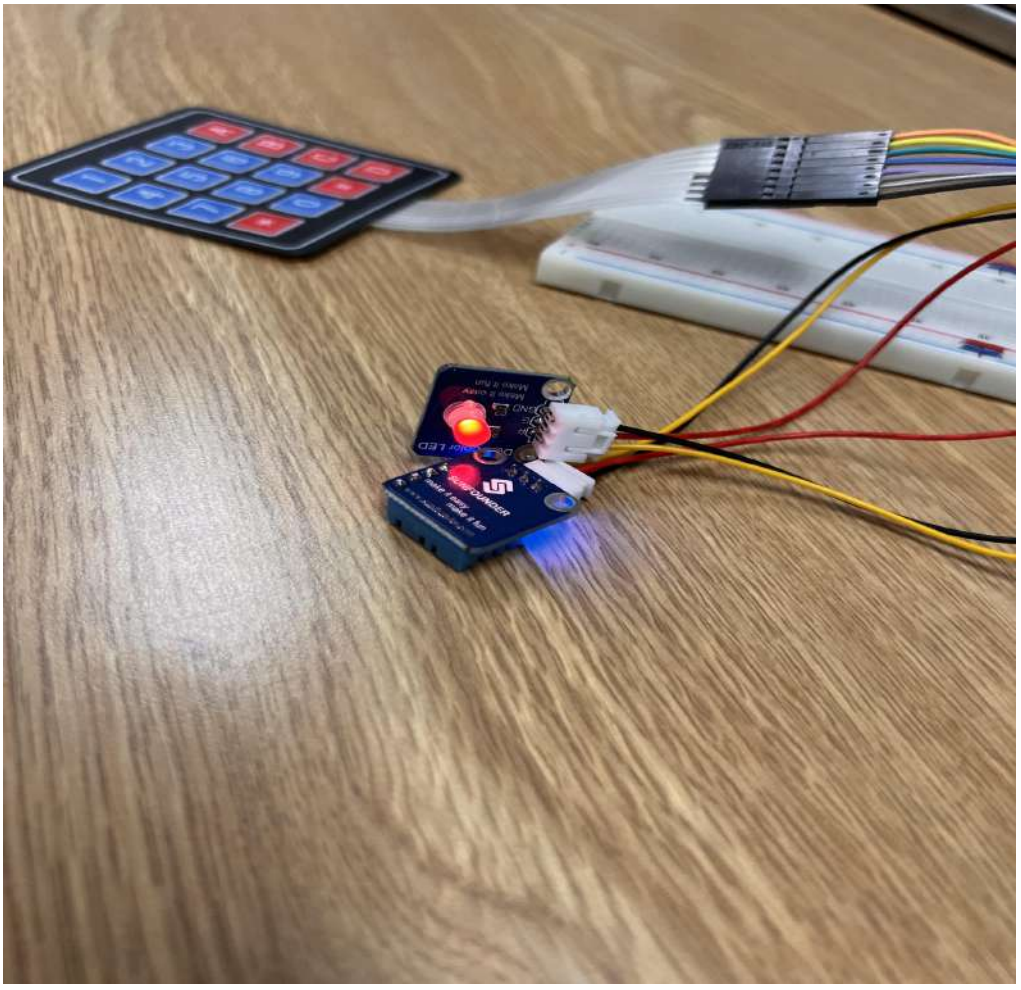
Part 4) Output Pictures: -

- Running the publish file we get this: -

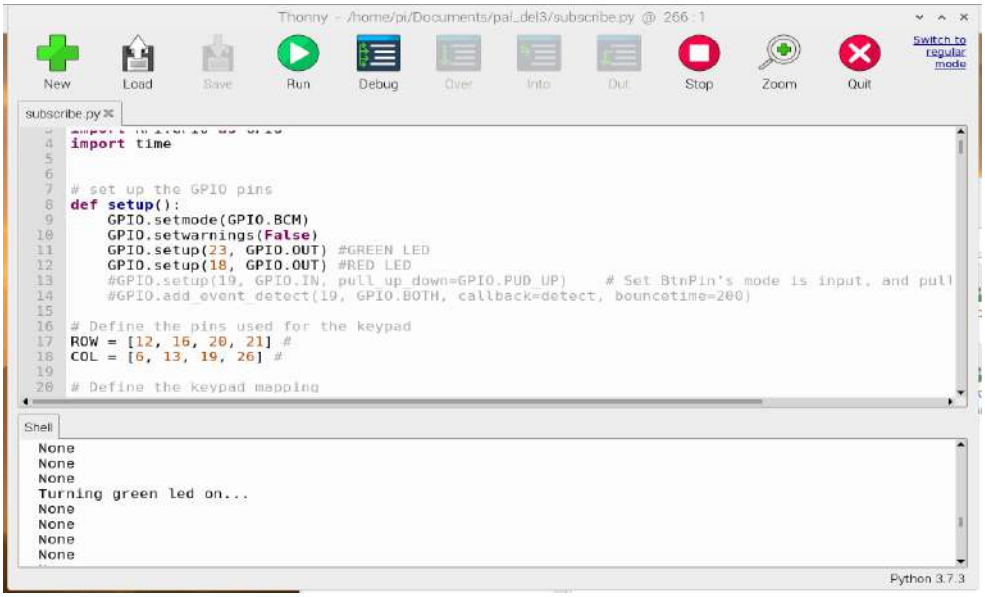


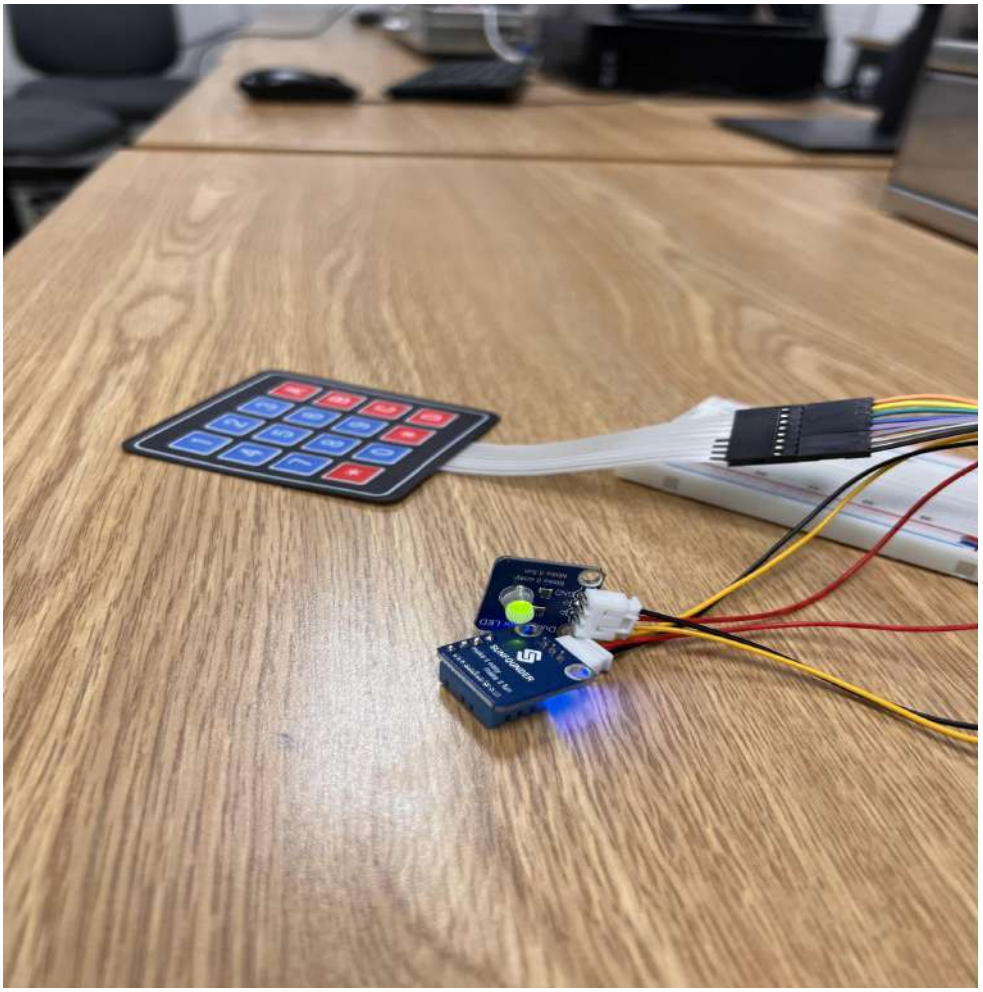
- On clicking “ON” we get this as the output on the circuit and terminal: -



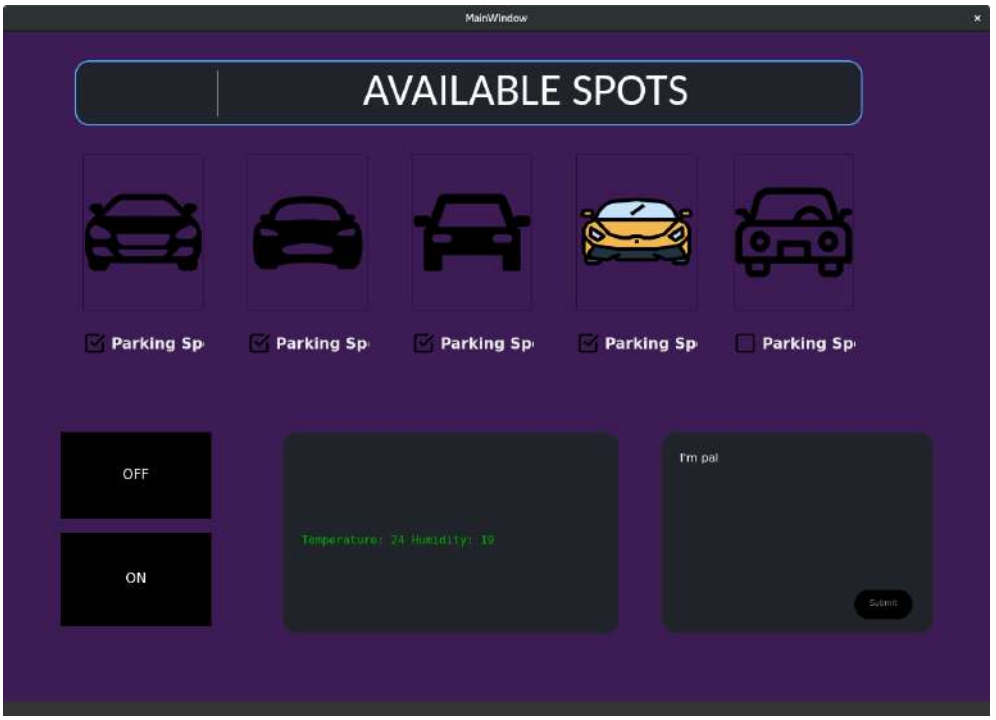


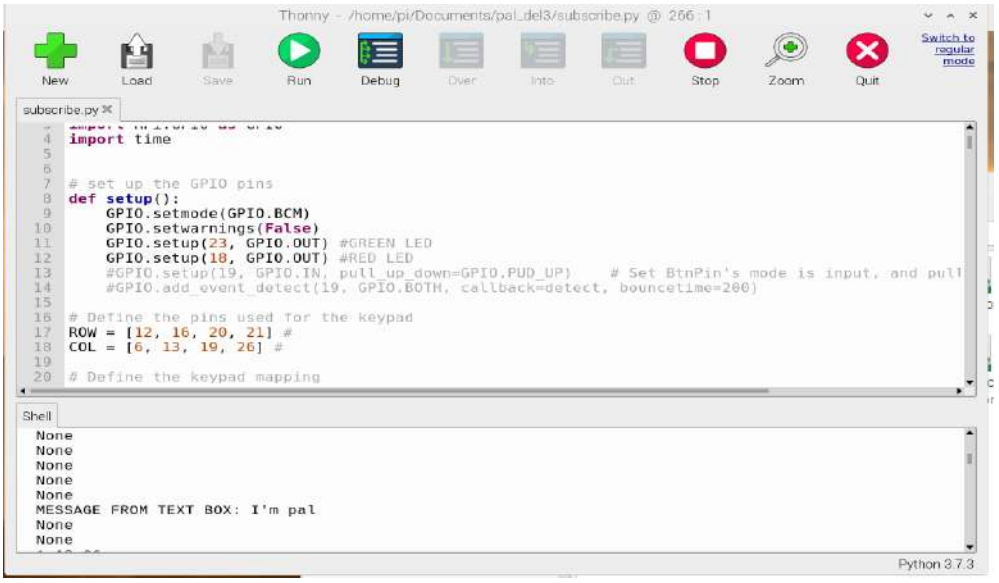
- On clicking “OFF” on the UI it turns on the Green LED on the circuit : -



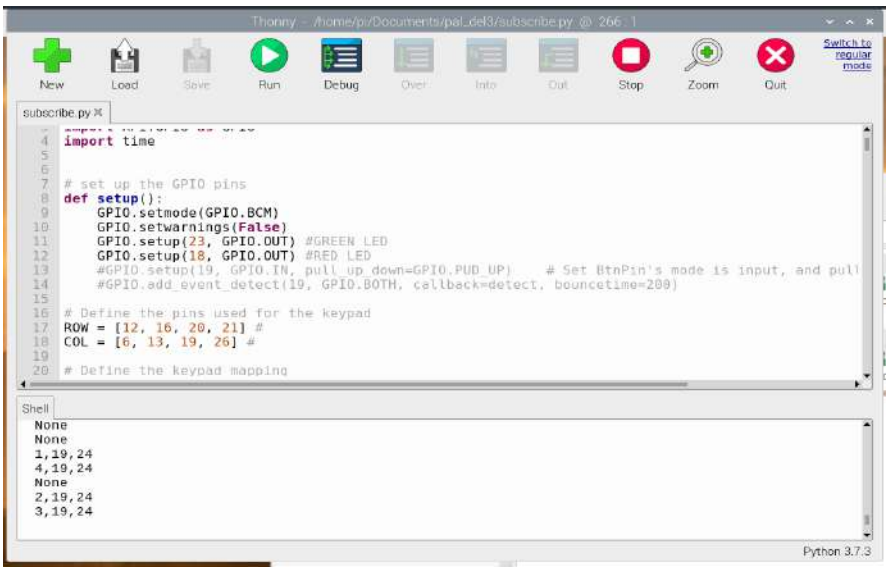


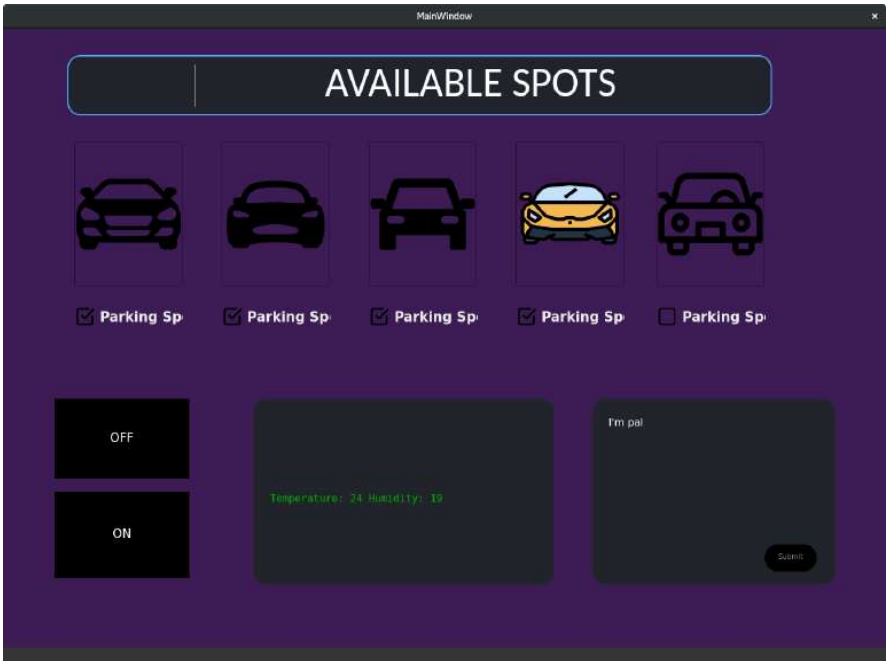
- On sending the message we get this on the RPi's terminal. The message should be sent from the UI's text box, The screenshot has been provided below: -



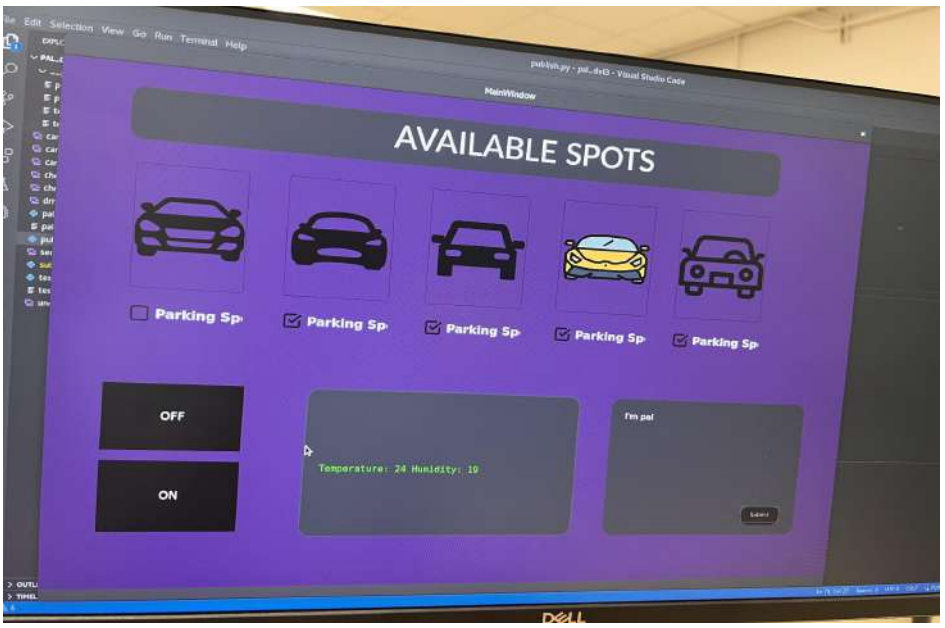


- On pressing keys 1, 4, 2, and 3 we get this on the terminal and the UI: -

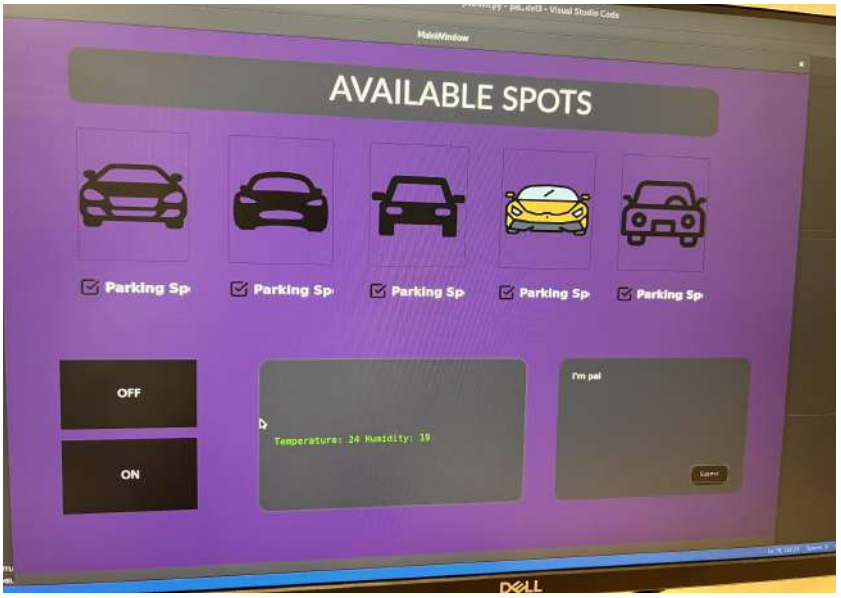




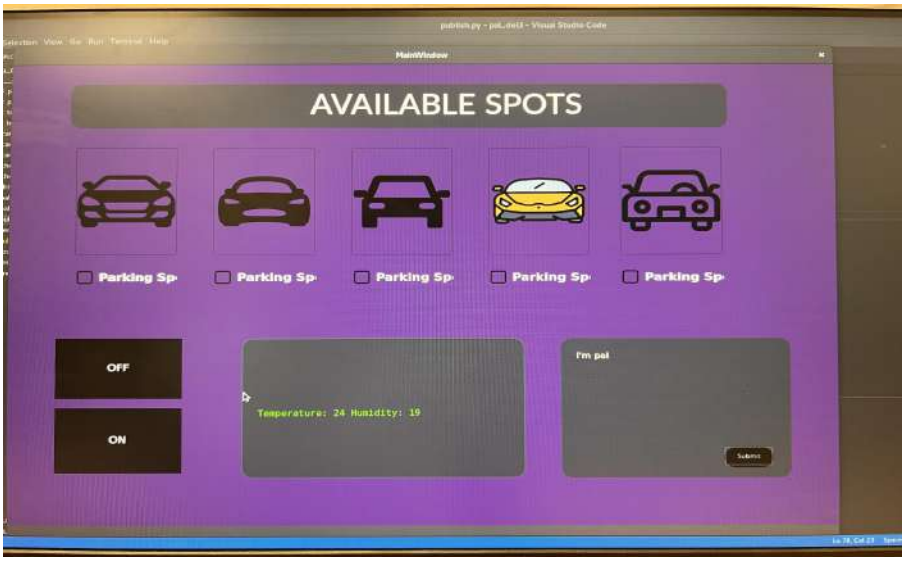
- On pressing Key6 which is used to turn off the parking spot 1 we get: -



- When Key “C” is pressed which is used to clear all the parking spot we get: -
    1. Before: -

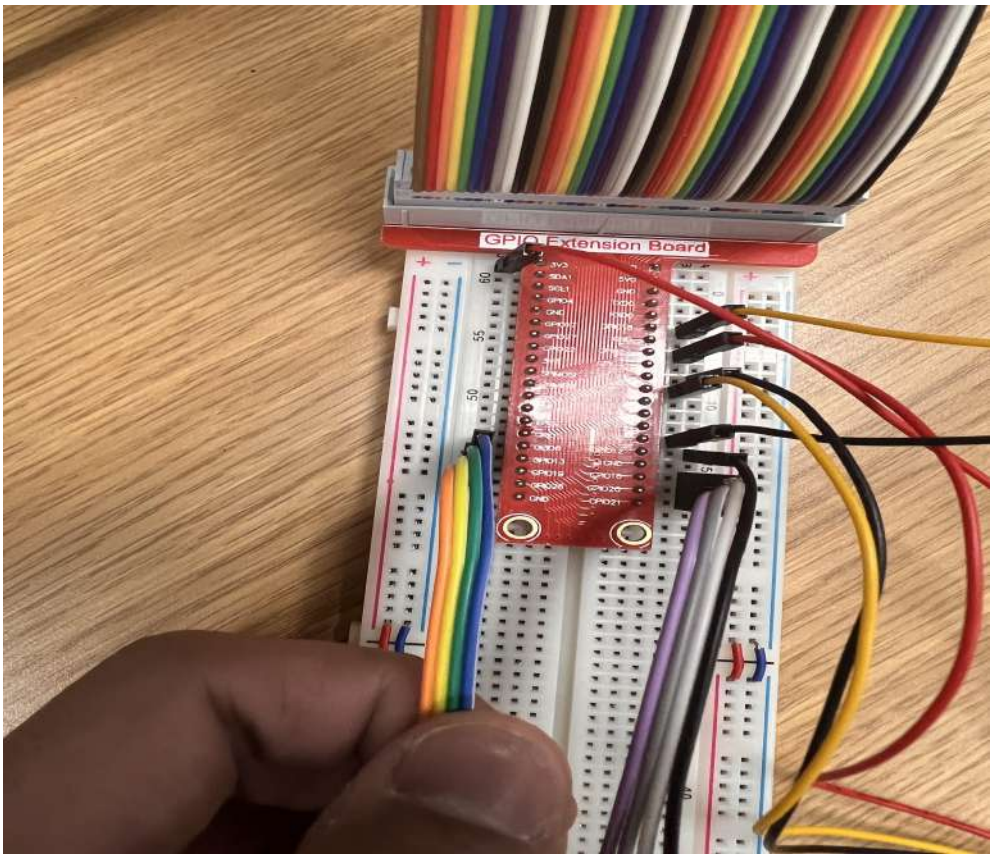


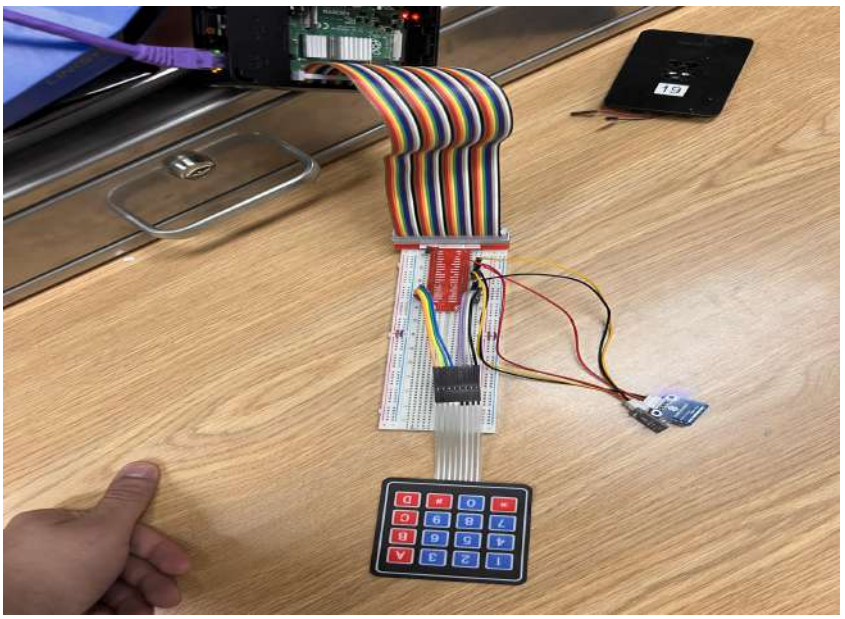
2. After: -

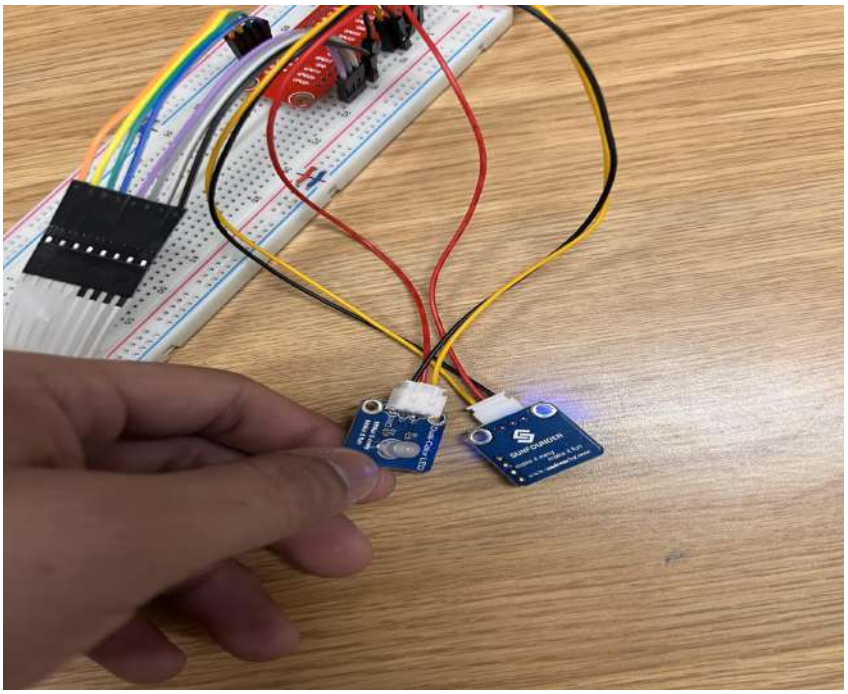


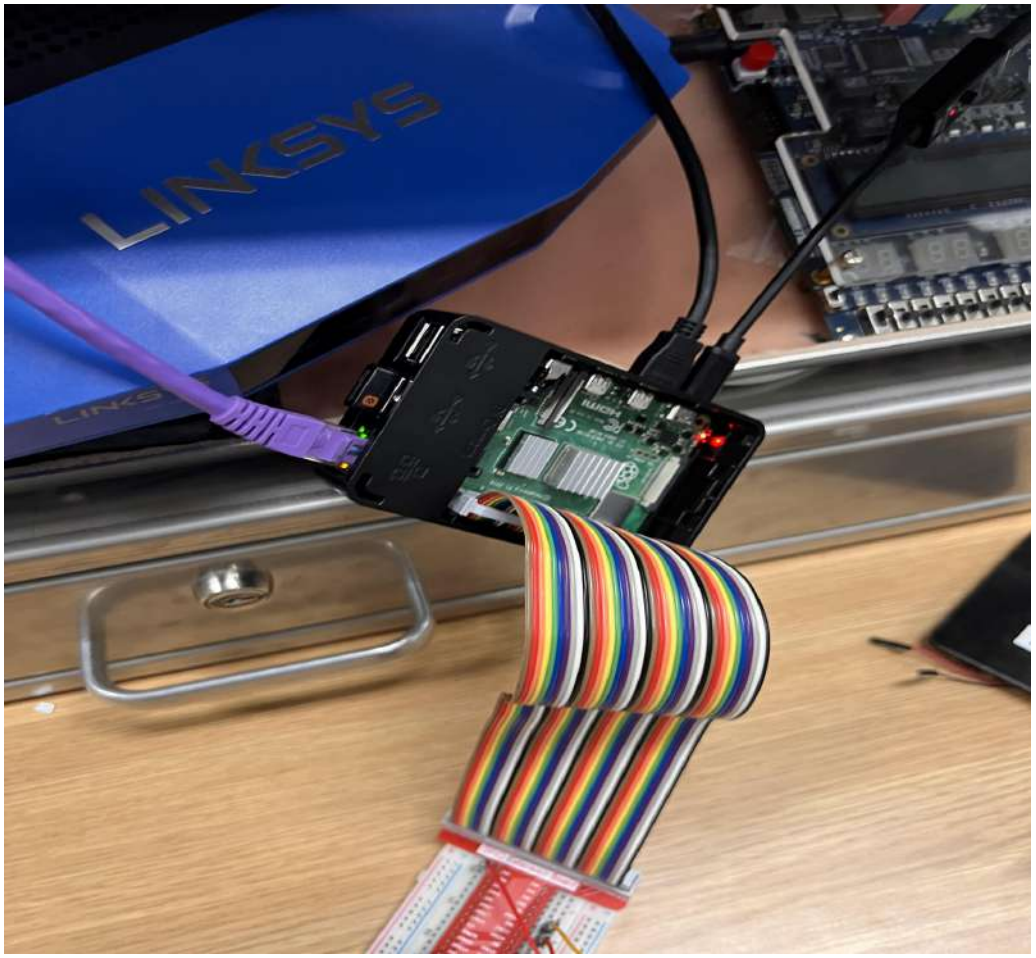
Part5) Circuit Design:

-



- 

- 

- 

- **Packages used: -**
  - ➔ pip3 install Rpi.GPIO
  - ➔ DHT11.py module from the sun founder
  - ➔ json
  - ➔ pyQt5
  - ➔ sys