Sending data from ESP to email account

Aim:

ESP32 microcontroller to monitor real-time sensor values. When the sensor value exceeds a predefined threshold, the system will automatically send an email notification to the user, ensuring timely alerts for potential issues or conditions requiring attention.

Software / Hardware requirements:

Software Requirements:

1. **Thonny Python IDE**: A beginner-friendly IDE for Python, used for writing, testing, and debugging Python code on the ESP32.
2. **ESP32 Firmware**: Necessary for running MicroPython on the ESP32 board, allowing Python scripts to be executed.
3. **Required Python Libraries**:
   umail, network, time, machine, socket, etc.

Hardware requirements:

1. **ESP32 IoT Development Board**: The main microcontroller board used for the experiments, featuring dual-core processing, Wi-Fi, Bluetooth, and multiple GPIO pins.
2. **Soil Moisture Sensor**: A sensor used to detect light intensity, connected to the ADC pin of the ESP32 for reading analog values.
3. **Touch Sensor**: A small capacitive touch sensor that detects touch for further applications.
4. Other components include resistors, Breadboard, Jumper Wires, and Micro USB cables.

Part A

Programs:

1) Main.py

```
import umail
import network, time, machine
sender_email = “ ”
sender_name = “ ”
sender_app_password = “ ”
recipient_email = “ ”
email_subject = “ ”
print(“Hello ESP32")
# Send the email
smtp = umail.SMTP('smtp.gmail.com', 465, ssl=True) # Gmail's SSL port
smtp.login(sender_email, sender_app_password)
smtp.to(recipient_email)
smtp.write("From:" + sender_name + "<"+ sender_email+">\n")
smtp.write("Subject:" + email_subject + "\n")
smtp.write("Hello ESP32 " "\n")
smtp.send()
smtp.quit()
```

2) Boot.py

```
import network, time, machine
ssid = '“ ”
password = '“ ”
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(ssid, password)
while station.isconnected() == False:
  pass
print('Connection successful')
print(station.ifconfig())
```

3) Umail.py

```
import socket

DEFAULT_TIMEOUT = 10 # sec
LOCAL_DOMAIN = '127.0.0.1'
CMD_EHLO = 'EHLO'
CMD_STARTTLS = 'STARTTLS'
CMD_AUTH = 'AUTH'
CMD_MAIL = 'MAIL'
AUTH_PLAIN = 'PLAIN'
AUTH_LOGIN = 'LOGIN'

class SMTP:
    def cmd(self, cmd_str):
        sock = self._sock;
        sock.write('%s\r\n' % cmd_str)
        resp = []
        next = True
        while next:
            code = sock.read(3)
            next = sock.read(1) == b'-'
            resp.append(sock.readline().strip().decode())
        return int(code), resp

    def __init__(self, host, port, ssl=False, username=None, password=None):
        import ssl
        self.username = username
        addr = socket.getaddrinfo(host, port)[0][-1]
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(DEFAULT_TIMEOUT)
        sock.connect(addr)
        if ssl:
```

```
        sock = ssl.wrap_socket(sock)
    code = int(sock.read(3))
    sock.readline()
    assert code==220, 'cant connect to server %d, %s' % (code, resp)
    self._sock = sock

    code, resp = self.cmd(CMD_EHLO + ' ' + LOCAL_DOMAIN)
    assert code==250, '%d' % code
    if not ssl and CMD_STARTTLS in resp:
        code, resp = self.cmd(CMD_STARTTLS)
        assert code==220, 'start tls failed %d, %s' % (code, resp)
        self._sock = ssl.wrap_socket(sock)

    if username and password:
        self.login(username, password)

def login(self, username, password):
    self.username = username
    code, resp = self.cmd(CMD_EHLO + ' ' + LOCAL_DOMAIN)
    assert code==250, '%d, %s' % (code, resp)

    auths = None
    for feature in resp:
        if feature[:4].upper() == CMD_AUTH:
            auths = feature[4:].strip('=').upper().split()
    assert auths!=None, "no auth method"

    from ubinascii import b2a_base64 as b64
    if AUTH_PLAIN in auths:
        cren = b64("\0%s\0%s" % (username, password))[:-1].decode()
        code, resp = self.cmd('%s %s %s' % (CMD_AUTH, AUTH_PLAIN, cren))
    elif AUTH_LOGIN in auths:
```

```
            code, resp = self.cmd("%s %s %s" % (CMD_AUTH, AUTH_LOGIN, b64(username)
[:-1].decode()))
            assert code==334, 'wrong username %d, %s' % (code, resp)
            code, resp = self.cmd(b64(password)[:-1].decode())
        else:
            raise Exception("auth(%s) not supported " % ', '.join(auths))

        assert code==235 or code==503, 'auth error %d, %s' % (code, resp)
        return code, resp

    def to(self, addrs, mail_from=None):
        mail_from = self.username if mail_from==None else mail_from
        code, resp = self.cmd('MAIL FROM: <%s>' % mail_from)
        assert code==250, 'sender refused %d, %s' % (code, resp)

        if isinstance(addrs, str):
            addrs = [addrs]
        count = 0
        for addr in addrs:
            code, resp = self.cmd('RCPT TO: <%s>' % addr)
            if code!=250 and code!=251:
                print('%s refused, %s' % (addr, resp))
                count += 1
        assert count!=len(addrs), 'recipient refused, %d, %s' % (code, resp)

        code, resp = self.cmd('DATA')
        assert code==354, 'data refused, %d, %s' % (code, resp)
        return code, resp

    def write(self, content):
        self._sock.write(content)

    def send(self, content=''):
```

```
        if content:
            self.write(content)
        self._sock.write('\r\n.\r\n') # the five letter sequence marked for ending
        line = self._sock.readline()
        return (int(line[:3]), line[4:].strip().decode())

    def quit(self):
        self.cmd("QUIT")
        self._sock.close()
```

Circuit:

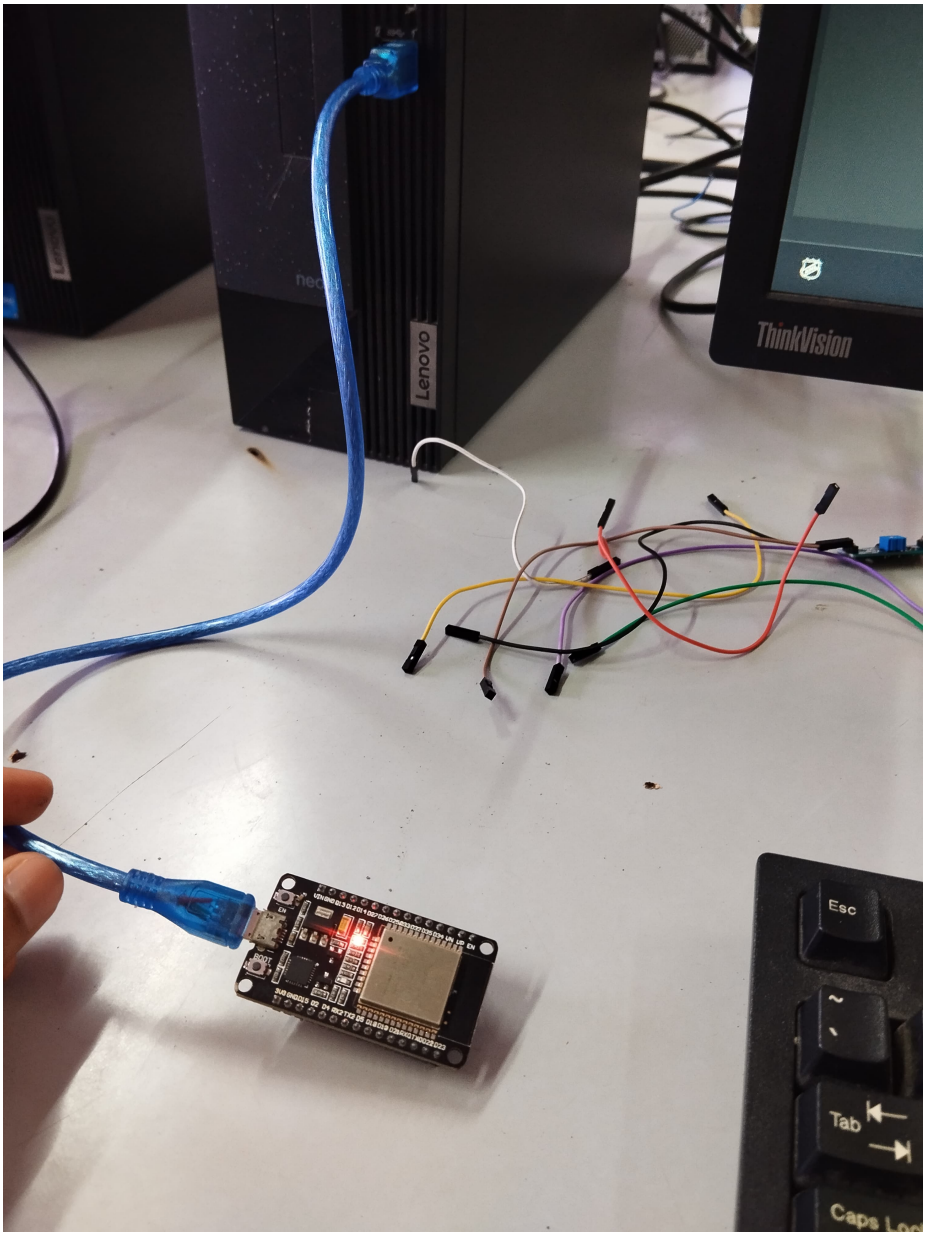

Fig. 1: ESP32 connections for mailing from dummy mail.

Output:

Fig. 2: Output of this code. Mail has been received to the receiving end.

Part B

Programs:

1) Main.py

```
import umail
import time
import machine

# Email configuration
sender_email = “ ”
sender_name = “ ”
sender_app_password = “ ”
recipient_email = “ ”
email_subject = 'Soil Moisture Level'

# Initialize capacitive touch sensor (adjust the pin as necessary)
touch_pin = machine.Pin(32)  # Replace with the actual pin used for the touch sensor
touch_sensor = machine.TouchPad(touch_pin)

# Initialize soil moisture sensor (analog pin)
moisture_sensor_pin = machine.Pin(34)  # Replace with the actual pin used for the moisture sensor
moisture_adc = machine.ADC(moisture_sensor_pin)

def send_email(moisture_value):
    try:
        smtp = umail.SMTP('smtp.gmail.com', 465, ssl=True)  # Gmail's SSL port
        smtp.login(sender_email, sender_app_password)
        smtp.to(recipient_email)
        smtp.write("From: " + sender_name + "<" + sender_email + ">\n")
        smtp.write("Subject: " + email_subject + "\n")
        smtp.write(f"Hello ESP32\nSoil Moisture Level: {moisture_value}\n")
        smtp.send()
        smtp.quit()
```

```
        print("Email sent!")
    except Exception as e:
        print("Failed to send email:", e)

# Main loop
try:
    while True:
        try:
            # Check if the touch sensor is triggered
            touch_value = touch_sensor.read()
            if touch_value < 500:  # Adjust threshold as needed
                print("Touch detected! Pausing for 30 seconds.")
                time.sleep(30)  # Pause for 30 seconds
            else:
                # Read soil moisture sensor value
                moisture_value = moisture_adc.read()  # Use ADC to read the analog value
                print("Soil Moisture Level:", moisture_value)

                # Send email with moisture value
                send_email(moisture_value)
                time.sleep(5)  # Wait for 5 seconds before the next reading

        except ValueError as e:
            print("Touch pad error:", e)
            time.sleep(1)  # Pause briefly before retrying

except KeyboardInterrupt:
    print("Program stopped.”)
```

2) Boot.py

```
import network, time, machine
ssid = “ ”
password = “ ”
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(ssid, password)
while station.isconnected() == False:
  pass
print('Connection successful')
print(station.ifconfig())
```

3) Umail.py

```
# uMail (MicroMail) for MicroPython
# Copyright (c) 2018 Shawwwn <shawwwn1@gmail.com>
# License: MIT
import socket

DEFAULT_TIMEOUT = 10 # sec
LOCAL_DOMAIN = '127.0.0.1'
CMD_EHLO = 'EHLO'
CMD_STARTTLS = 'STARTTLS'
CMD_AUTH = 'AUTH'
CMD_MAIL = 'MAIL'
AUTH_PLAIN = 'PLAIN'
AUTH_LOGIN = 'LOGIN'

class SMTP:
  def cmd(self, cmd_str):
    sock = self._sock;
    sock.write('%s\r\n' % cmd_str)
    resp = []
```

```
    next = True
    while next:
        code = sock.read(3)
        next = sock.read(1) == b'-'
        resp.append(sock.readline().strip().decode())
    return int(code), resp

def __init__(self, host, port, ssl=False, username=None, password=None):
    import ssl
    self.username = username
    addr = socket.getaddrinfo(host, port)[0][-1]
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(DEFAULT_TIMEOUT)
    sock.connect(addr)
    if ssl:
        sock = ssl.wrap_socket(sock)
    code = int(sock.read(3))
    sock.readline()
    assert code==220, 'cant connect to server %d, %s' % (code, resp)
    self._sock = sock

    code, resp = self.cmd(CMD_EHLO + ' ' + LOCAL_DOMAIN)
    assert code==250, '%d' % code
    if not ssl and CMD_STARTTLS in resp:
        code, resp = self.cmd(CMD_STARTTLS)
        assert code==220, 'start tls failed %d, %s' % (code, resp)
        self._sock = ssl.wrap_socket(sock)

    if username and password:
        self.login(username, password)

def login(self, username, password):
    self.username = username
```

```
        code, resp = self.cmd(CMD_EHLO + ' ' + LOCAL_DOMAIN)
        assert code==250, '%d, %s' % (code, resp)

        auths = None
        for feature in resp:
            if feature[:4].upper() == CMD_AUTH:
                auths = feature[4:].strip('=').upper().split()
        assert auths!=None, "no auth method"

        from ubinascii import b2a_base64 as b64
        if AUTH_PLAIN in auths:
            cren = b64("\0%s\0%s" % (username, password))[:-1].decode()
            code, resp = self.cmd('%s %s %s' % (CMD_AUTH, AUTH_PLAIN, cren))
        elif AUTH_LOGIN in auths:
            code, resp = self.cmd("%s %s %s" % (CMD_AUTH, AUTH_LOGIN, b64(username)
[:-1].decode()))
            assert code==334, 'wrong username %d, %s' % (code, resp)
            code, resp = self.cmd(b64(password)[:-1].decode())
        else:
            raise Exception("auth(%s) not supported " % ', '.join(auths))

        assert code==235 or code==503, 'auth error %d, %s' % (code, resp)
        return code, resp

    def to(self, addrs, mail_from=None):
        mail_from = self.username if mail_from==None else mail_from
        code, resp = self.cmd('MAIL FROM: <%s>' % mail_from)
        assert code==250, 'sender refused %d, %s' % (code, resp)

        if isinstance(addrs, str):
            addrs = [addrs]
        count = 0
        for addr in addrs:
```

```
        code, resp = self.cmd('RCPT TO: <%s>' % addr)
        if code!=250 and code!=251:
            print('%s refused, %s' % (addr, resp))
            count += 1
    assert count!=len(addrs), 'recipient refused, %d, %s' % (code, resp)

    code, resp = self.cmd('DATA')
    assert code==354, 'data refused, %d, %s' % (code, resp)
    return code, resp

def write(self, content):
    self._sock.write(content)

def send(self, content=''):
    if content:
        self.write(content)
    self._sock.write('\r\n.\r\n') # the five letter sequence marked for ending
    line = self._sock.readline()
    return (int(line[:3]), line[4:].strip().decode())

def quit(self):
    self.cmd("QUIT")
    self._sock.close()
```

Circuit:

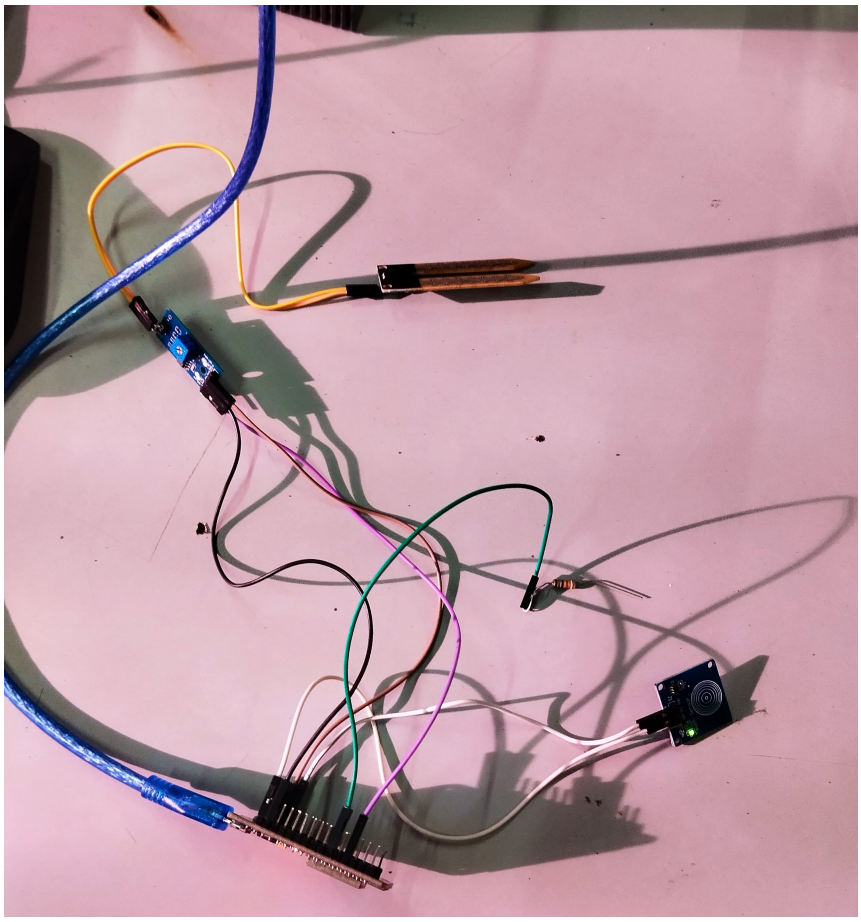

Fig. 3: ESP32 connections for mailing readings of soil moisture sensor wiht touch sensor connections.

Output:

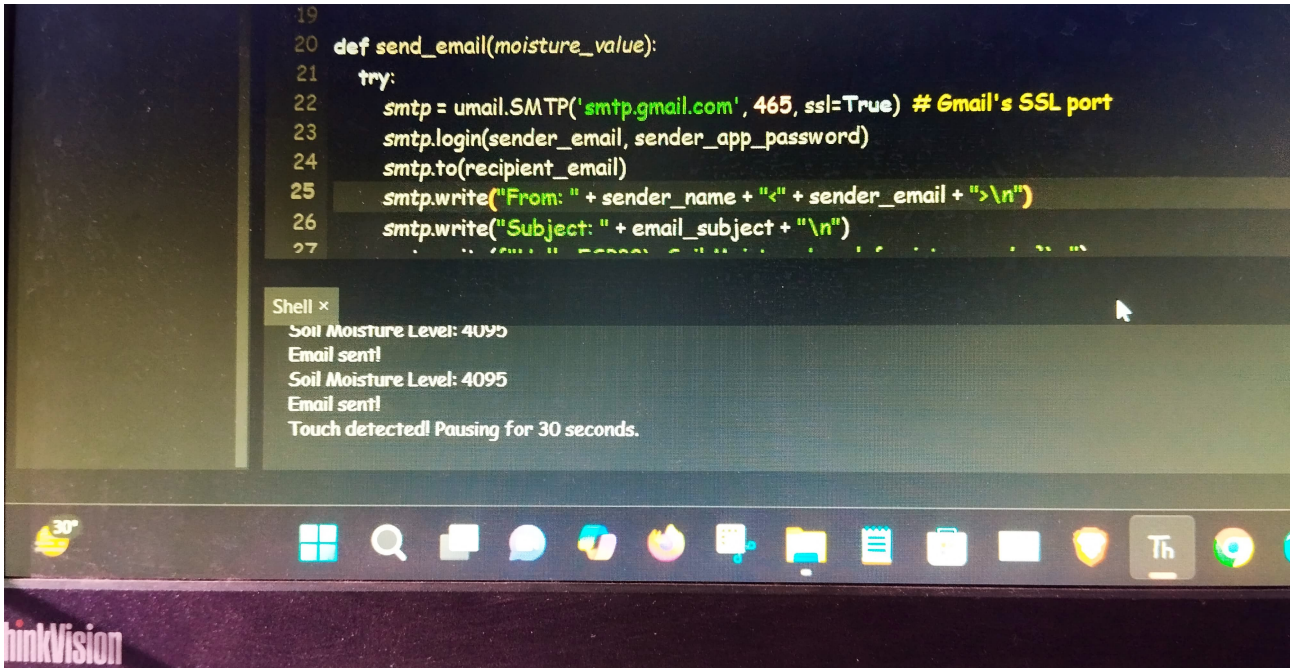

Fig. 4: Shell window telling that email has Been sent with soil moisture result. And if touch sensor detetcts a touch then it will stop mailing for 30 secs

Fig. 4: Received mail.

Conclusion:

The ESP32-based monitoring system effectively tracks real-time sensor values, sending timely email notifications when thresholds are exceeded. Utilizing Thonny Python IDE, MicroPython firmware, and essential libraries, this setup demonstrates a reliable and efficient IoT solution for automated alerts, ensuring prompt attention to potential issues in various applications.