**Department of CSE**

**Course Code**

**CSE406**

**Course Title**

**Internet of Things**

**Lab 03**

**Section : 01 Semester : Summer2025**

****

**Submitted By**

| **Name** | **ID** |
| --- | --- |
| **Seendid Saleh Kabir** | **2022-1-60-122** |



**Submitted To**

**Dr. Raihan Ul Islam**

**Associate Professor**

**Department of Computer Science and Engineering**

**East West University**

**Lab HTTP**

**ESP8266 ↔ Flask REST – Connectivity Test Report**

**Goal**

Verify that a single NodeMCU (ESP8266) can periodically call a REST endpoint hosted on a PC over the local network and receive JSON responses.

**Setup**

* **Client:** NodeMCU 1.0 (ESP-12E) running Arduino sketch (HTTPClient + WiFiMulti).
  + SSID: **“Seendid Wifi 2G”** (2.4 GHz).
  + Request interval: **10 s** (delay(10000)).
* **Server:** Python **Flask** app listening on **0.0.0.0:5000** with route **/rest**.
  + GET → {"message":"GET request received"}
  + POST → echoes JSON
* **Network:** Same LAN/subnet.
  + **Server IP:** 192.168.0.103
  + **ESP IP:** 192.168.0.108

**Evidence & Observations**

* **ESP Serial Monitor (sample):**
* WiFi connected
* IP address:
* 192.168.0.108
* [HTTP] begin...
* [HTTP] GET... code: 200
* {"message":"GET request received"}

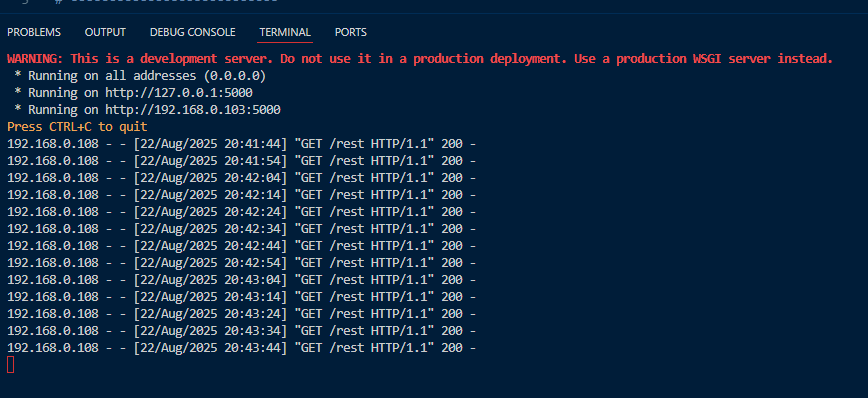
Repeats every ~10 s.

* **Flask Console (sample):**
* 192.168.0.108 - - [22/Aug/2025 20:43:44] "GET /rest HTTP/1.1" 200 -
* 192.168.0.108 - - [22/Aug/2025 20:43:54] "GET /rest HTTP/1.1" 200 -
* 192.168.0.108 - - [22/Aug/2025 20:44:04] "GET /rest HTTP/1.1" 200 -
* **Timing:** Timestamps show requests at ~10-second intervals (e.g., :44, :54, :04, :14).  
   Effective rate ≈ **6 requests/min**.
* **Status:** All responses **HTTP 200** with expected JSON payload. Connectivity is stable.

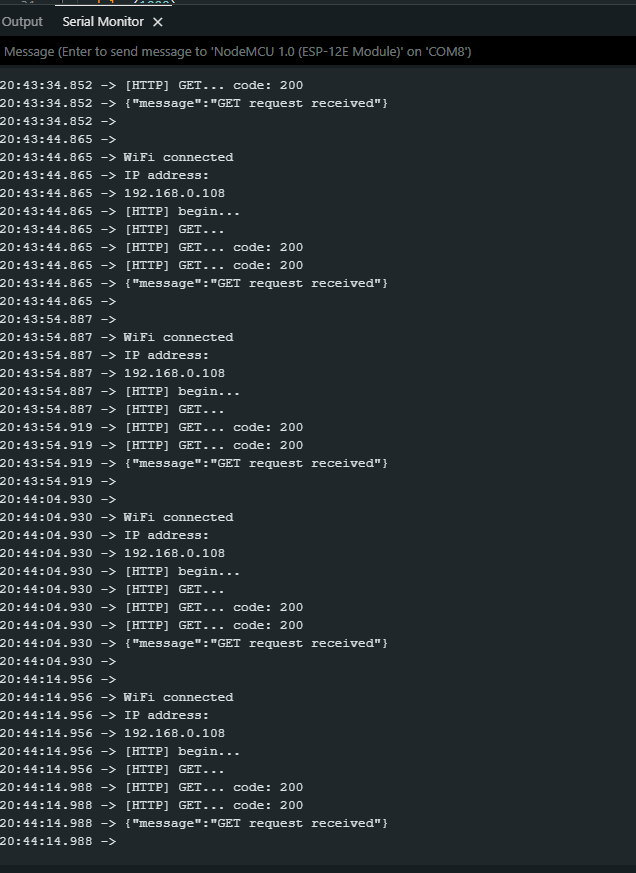
**Conclusion**

The system works as intended with **one** NodeMCU acting as an HTTP client and a PC-hosted Flask server responding on /rest. The earlier connection issue was due to attempting a **5 GHz** SSID; switching to **2.4 GHz** resolved it.

VScode Output:



Arduino Output:



**Lab COAP  
CoAP Light Control — Final Verification Report (ESP8266 + aiocoap)**

**1) Objective**

Validate end-to-end CoAP control of the NodeMCU (ESP8266) “light” endpoint for both commands:

* "1" → Turn ON
* "0" → Turn OFF

**2) System Setup**

* Device: NodeMCU (ESP8266)
* Firmware (server): coap-simple with endpoint /light (code you shared)
  + Parses payload as text ("1"/"0")
  + Drives active-low on-board LED
  + Replies with current state ("1" when ON, "0" when OFF) and 2.04 Changed
* Wi-Fi: 2.4 GHz (Seendid Wifi 2G); ESP got IP 192.168.0.108
* Client: Python aiocoap PUT to coap://192.168.0.108/light
  + Two equivalent scripts were used; one also sets Content-Format: application/octet-stream.

**3) Test Procedures & Results**

A. Instruction ON (payload = b'1')

Client action

Message(code=PUT, uri="coap://192.168.0.108/light", payload=b"1")

ESP serial (observed – your screenshots):

[Light] Request received.

Payload received: 1

Instruction: Turn ON

Client console (observed):

ACK 2.04 Changed ...

Response Code: 2.04 Changed

Response Payload: 1

Result: PASS — LED turns ON (LOW), server echoes "1".

**B. Instruction OFF (payload = b'0')**

Client action

Message(code=PUT, uri="coap://192.168.0.108/light", payload=b"0")

ESP serial (earlier run):

[Light] Request received.

Payload received: 0

Instruction: Turn OFF

Client console:

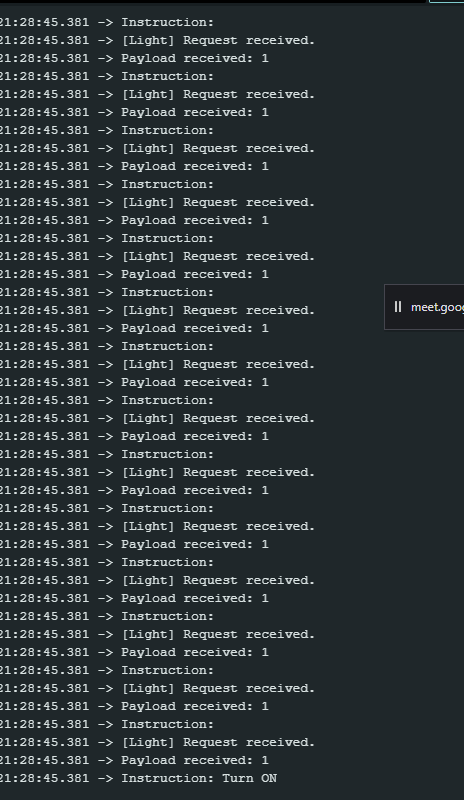
Response Code: 2.04 Changed

Response Payload: 0

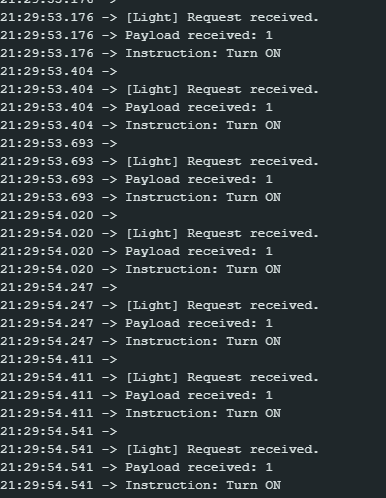
Result: PASS — LED turns OFF (HIGH), server echoes "0".

**4) Observations**

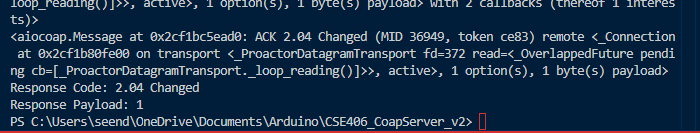
* The ESP prints the expected Instruction: Turn ON/OFF lines and mirrors the state in its CoAP response.
* Python side consistently receives 2.04 Changed with the correct payload ("1"/"0").

Arduino Output:(Instruction turned off)  


Instruction turned on:

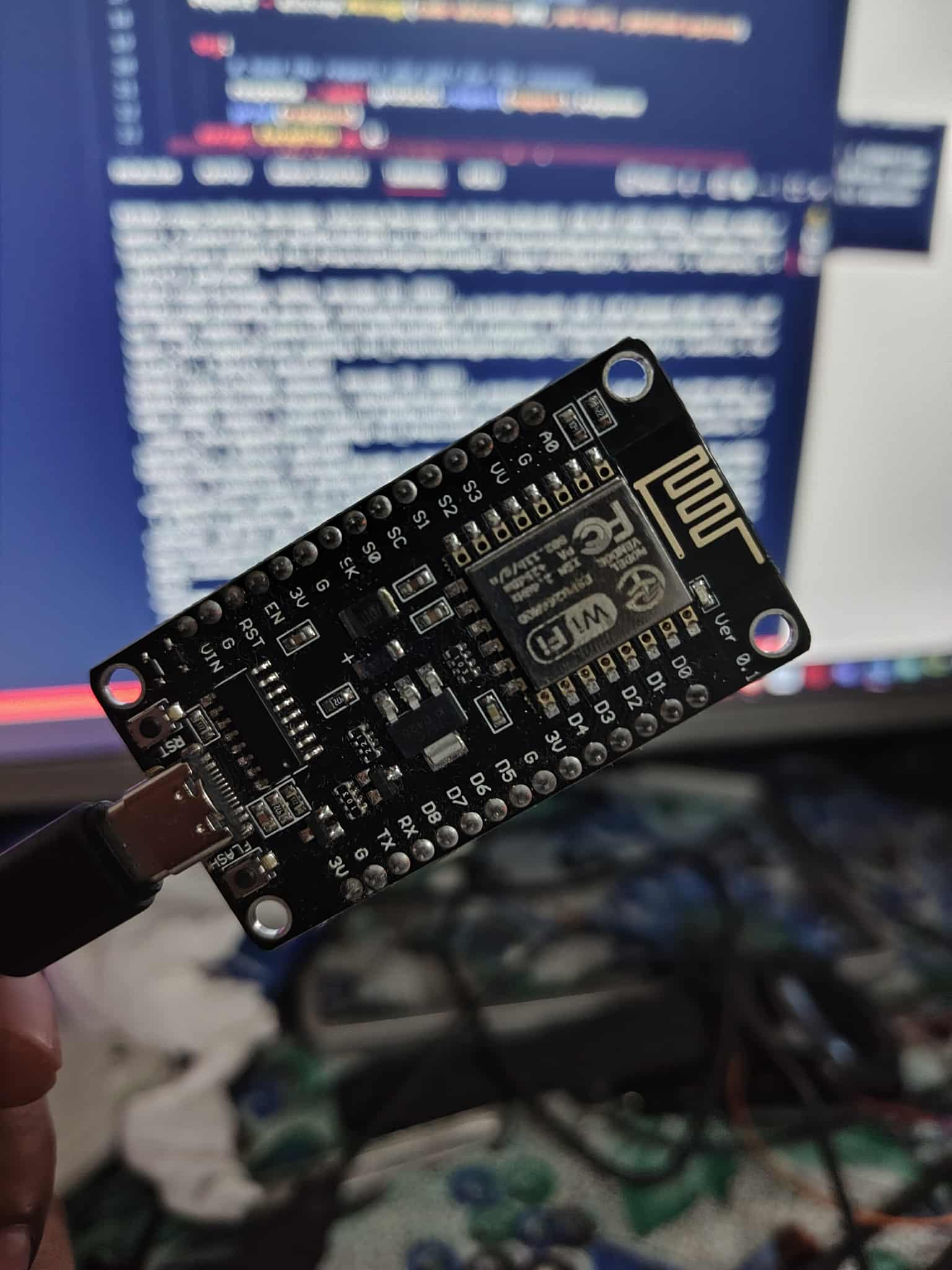


Vs output:



Situation:

payload = "0".encode('utf-8')



Situation:

payload = "1".encode('utf-8')

