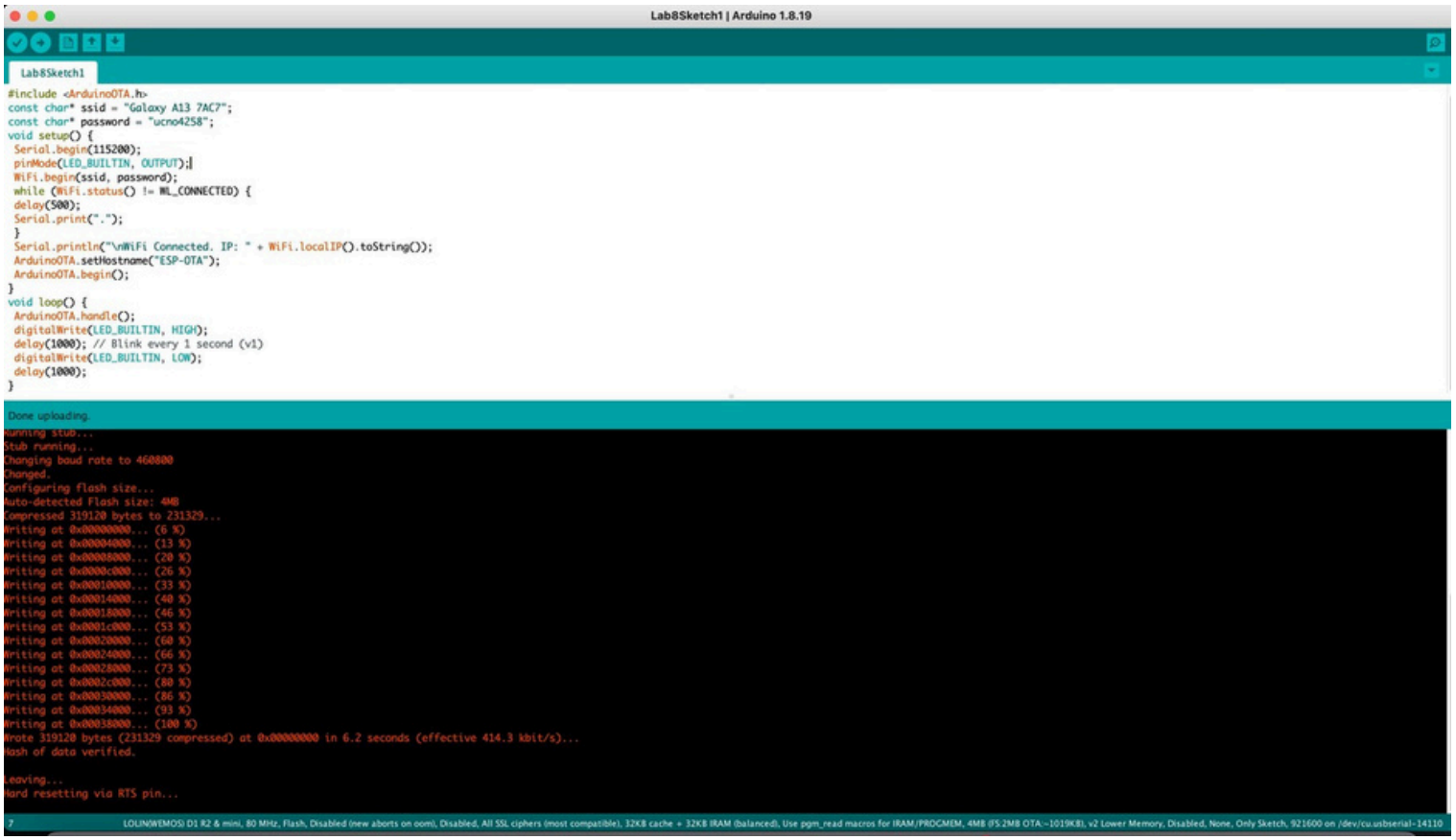
Lab Task 1: Manual OTA Updates with ArduinoOTA

In this task, we will implement an Over-the-Air (OTA) firmware update mechanism for the ESP8266 microcontroller using the Arduino OTA library.

```
#include <ESP8266WiFi.h>
#include <ArduinoOTA.h>
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
void setup() {
 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
 ArduinoOTA.setHostname("ESP-OTA");
 ArduinoOTA.begin();
}
void loop() {
 ArduinoOTA.handle();
 digitalWrite(LED_BUILTIN, HIGH);
 delay(1000); // Blink every 1 second (v1)
 digitalWrite(LED_BUILTIN, LOW);
 delay(1000);
}
```

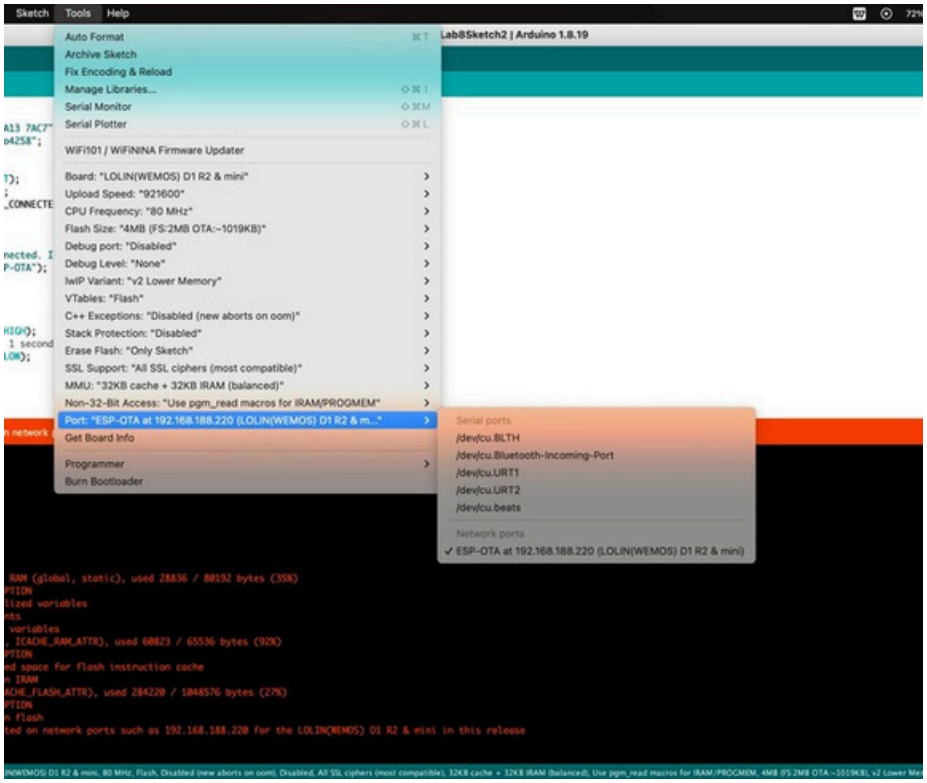
WiFi Connected. IP: 192.168.188.220

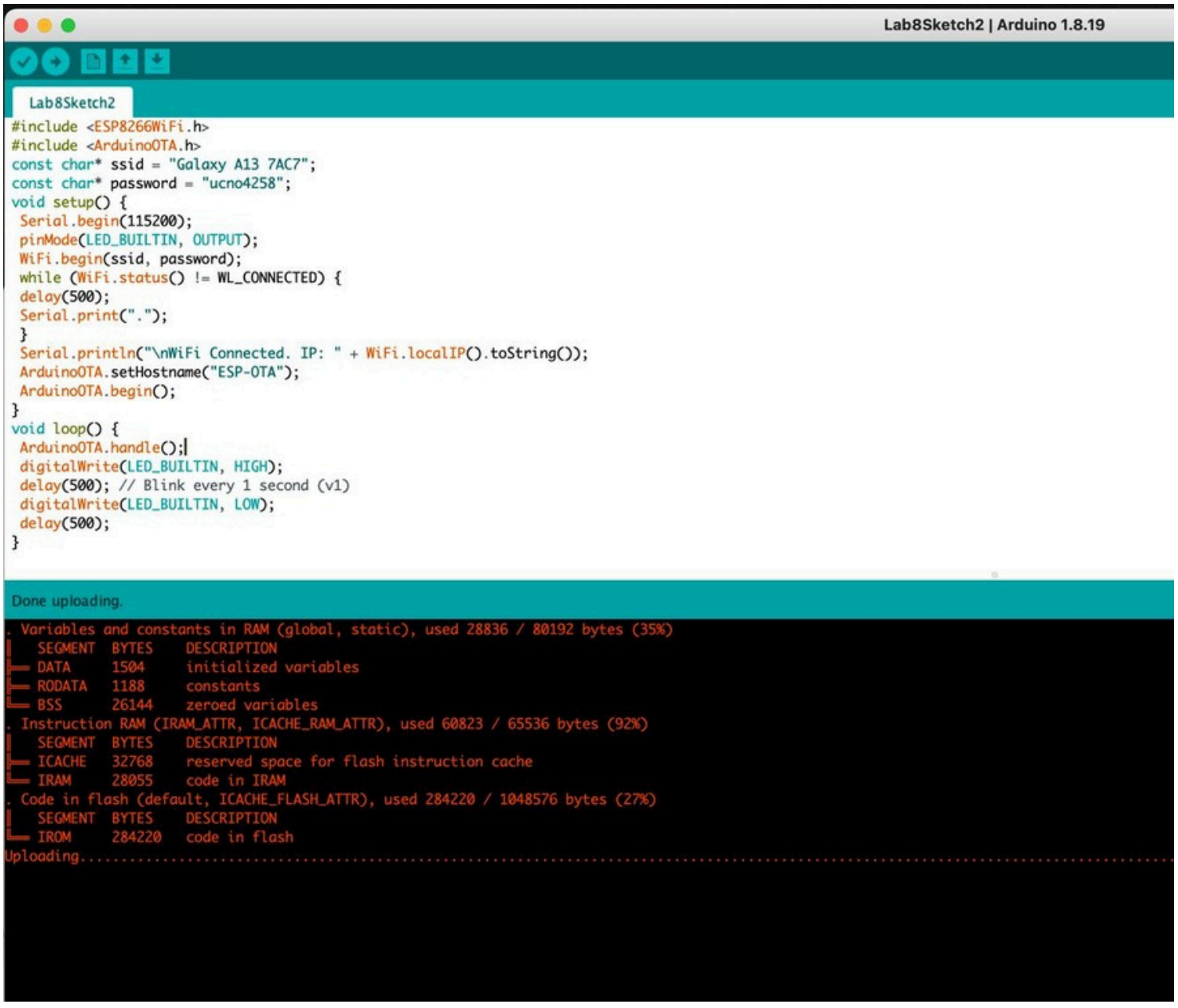
In the first step, we manually through wire updated the firmware of esp8266. Screenshots are attached. The esp's LED was blinking after each 1 second.



Now, we changed the delay to 500 meaning there will be a delay of 0.5 second between each blink. Exported the compiled binary. Powered the esp8266 through some power source. And by connecting through network port updated the firmware of esp8266 with the following code.

```
#include <ESP8266WiFi.h>
#include <ArduinoOTA.h>
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
void setup() {
 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
 ArduinoOTA.setHostname("ESP-OTA");
 ArduinoOTA.begin();
}
void loop() {
 ArduinoOTA.handle();
 digitalWrite(LED_BUILTIN, HIGH);
 delay(500); // Blink every 1 second (v1)
 digitalWrite(LED_BUILTIN, LOW);
 delay(500);
}
```


Sketch
Tools
Help
Auto Format
Archive Sketch
Fix Encoding & Reload
Manage Libraries...
Serial Monitor
Serial Plotter
WiFi101 / WiFiNINA Firmware Updater
Board: "LOLIN(WEMOS) D1 R2 & mini"
Upload Speed: "921600"
CPU Frequency: "80 MHz"
Flash Size: "4MB (FS:2MB OTA:~1019KB)"
Debug port: "Disabled"
Debug Level: "None"
lwIP Variant: "v2 Lower Memory"
VTables: "Flash"
C++ Exceptions: "Disabled (new aborts on oom)"
Stack Protection: "Disabled"
Erase Flash: "Only Sketch"
SSL Support: "All SSL ciphers (most compatible)"
MMU: "32KB cache + 32KB IRAM (balanced)"
Non-32-Bit Access: "Use pgm_read macros for IRAM/PROGMEM"
Port: "ESP-OTA at 192.168.188.220 (LOLIN(WEMOS) D1 R2 & m..."
Get Board Info
Programmer
Burn Bootloader
Serial ports
/dev/cu.BLTH
/dev/cu.Bluetooth-Incoming-Port
/dev/cu.URT1
/dev/cu.URT2
/dev/cu.beats
Network ports
ESP-OTA at 192.168.188.220 (LOLIN(WEMOS) D1 R2 & mini)


Lab8Sketch2 | Arduino 1.8.19
Lab8Sketch2
#include <ESP8266WiFi.h>
#include <ArduinoOTA.h>
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
void setup() {
Serial.begin(115200);
pinMode(LED_BUILTIN, OUTPUT);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
ArduinoOTA.setHostname("ESP-OTA");
ArduinoOTA.begin();
}
void loop() {
ArduinoOTA.handle();
digitalWrite(LED_BUILTIN, HIGH);
delay(500); // Blink every 1 second (v1)
digitalWrite(LED_BUILTIN, LOW);
delay(500);
}
Done uploading.
. Variables and constants in RAM (global, static), used 28836 / 80192 bytes (35%)
SEGMENT BYTES DESCRIPTION
DATA 1504 initialized variables
RODATA 1188 constants
BSS 26144 zeroed variables
. Instruction RAM (IRAM_ATTR, ICACHE_RAM_ATTR), used 60823 / 65536 bytes (92%)
SEGMENT BYTES DESCRIPTION
ICACHE 32768 reserved space for flash instruction cache
IRAM 28055 code in IRAM
. Code in flash (default, ICACHE_FLASH_ATTR), used 284220 / 1048576 bytes (27%)
SEGMENT BYTES DESCRIPTION
IROM 284220 code in flash
Uploading.....

Analysis of code:

The first two lines of the code include libraries: ESP8266WiFi.h lets the ESP8266 connect to WiFi, and ArduinoOTA.h allows us to send new code to the ESP using WiFi. Then we write our WiFi name (ssid) and password (password) so the ESP knows which network to join.

In the setup() part, we start the serial monitor using Serial.begin(115200);—this helps us see messages from the ESP on serial monitor. Then we set the built-in LED pin as an output using pinMode(LED_BUILTIN, OUTPUT); so we can blink it. After that, the ESP tries to connect to WiFi with WiFi.begin(ssid, password);. It keeps printing a dot (.) every half second while waiting to connect. When it connects, it shows the IP address (its network address) on the screen using Serial.println().

Next, we give the ESP a name using ArduinoOTA.setHostname("ESP-OTA");. This name helps us find the device easily when sending new code. Then we start the OTA service using ArduinoOTA.begin();.
In the loop() part, the code keeps calling ArduinoOTA.handle(); again and again. This checks if a new update is coming. If yes, it gets ready to receive the new code. After that, the code turns the LED on with digitalWrite(LED_BUILTIN, HIGH);, waits half a second in version 2 and wait for a second in version 1, then turns it off with digitalWrite(LED_BUILTIN, LOW); and waits again. This makes the LED blink every second, so we can see that the ESP is running

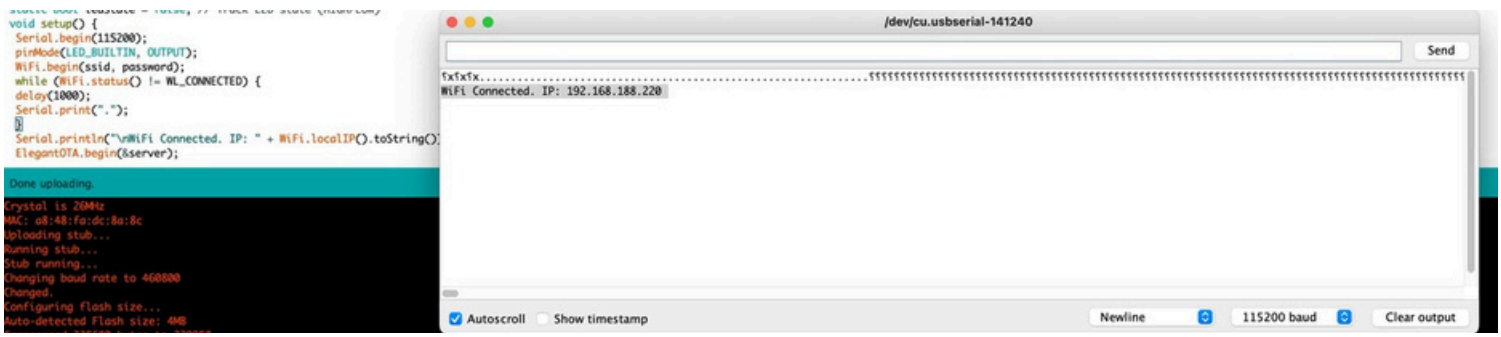
Short Description:

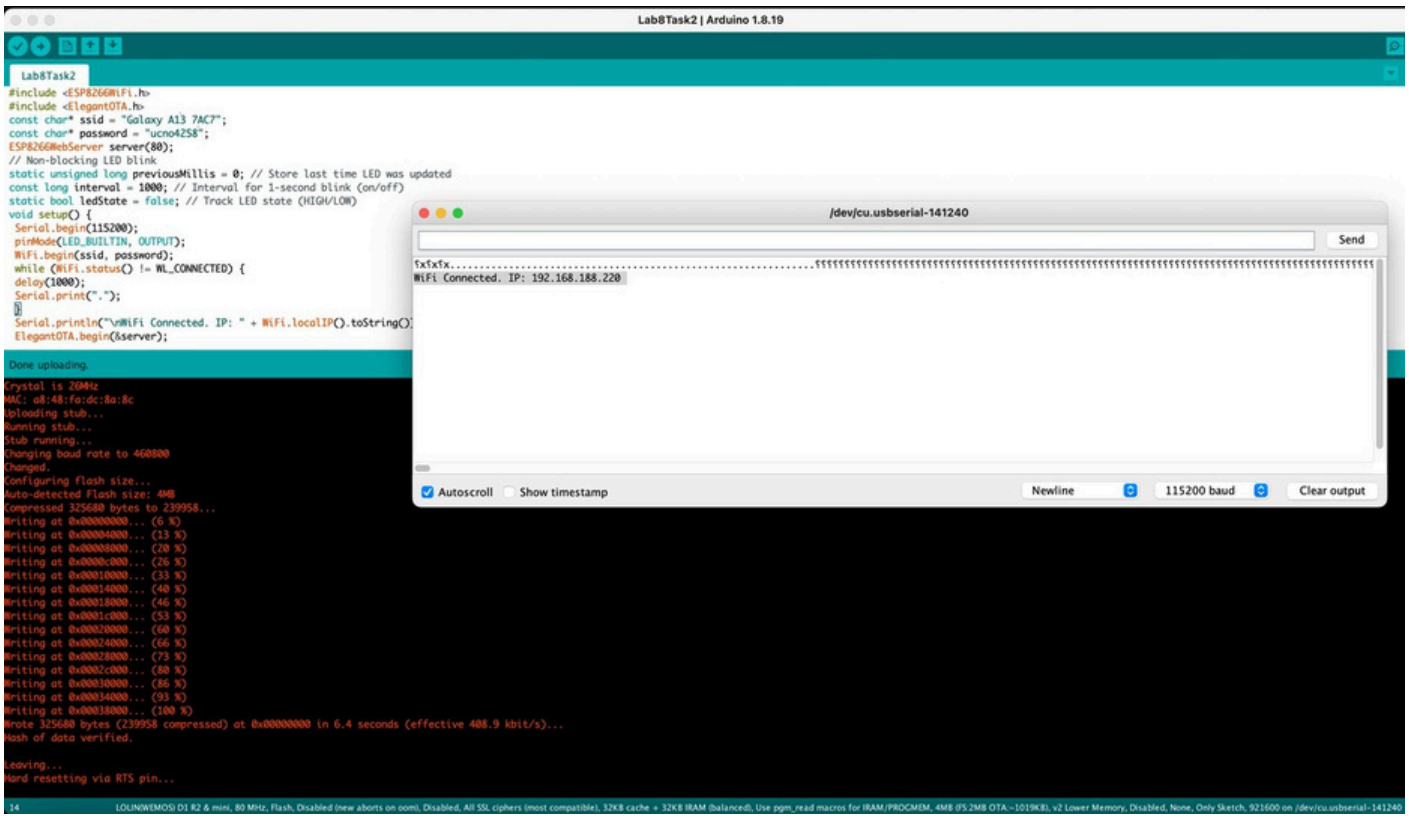
First, we connected the ESP8266 to my WiFi network by entering the SSID and password in the code. Then we used the ArduinoOTA library to enable Over-The-Air updates. We uploaded the code to the ESP8266 using USB the first time. After that, we started the OTA service and used the IP address shown in the Serial Monitor to send new code wirelessly from the Arduino IDE.
To verify the functionality, we made a small change in the code (like blinking the LED faster or slower) and uploaded it using OTA. The LED behavior changed, which confirmed that the new code was uploaded successfully.
One challenge we faced was the ESP8266 not showing up in the "Port" menu for OTA upload. We solved this by making sure both my computer and the ESP8266 were connected to the same WiFi network and by keeping the Serial Monitor closed while uploading via OTA.

Lab Task 2: Manual OTA Updates with ElegantOTA

Version 1:

```
#include <ESP8266WiFi.h>
#include <ElegantOTA.h>
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
ESP8266WebServer server(80);
// Non-blocking LED blink
static unsigned long previousMillis = 0; // Store last time LED was updated
const long interval = 1000; // Interval for 1-second blink (on/off)
static bool ledState = false; // Track LED state (HIGH/LOW)
void setup() {
Serial.begin(115200);
pinMode(LED_BUILTIN, OUTPUT);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(1000);
Serial.print(".");
}
Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
ElegantOTA.begin(&server);
server.begin();
}
void loop() {
server.handleClient();
ElegantOTA.loop();
unsigned long currentMillis = millis(); // Get current time
// Check if it's time to toggle the LED
if (currentMillis - previousMillis >= interval) {
ledState = !ledState; // Toggle state
digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW); // Update LED
previousMillis = currentMillis; // Save the current time
}
}
```





The LED was blinking with 1 second delay.

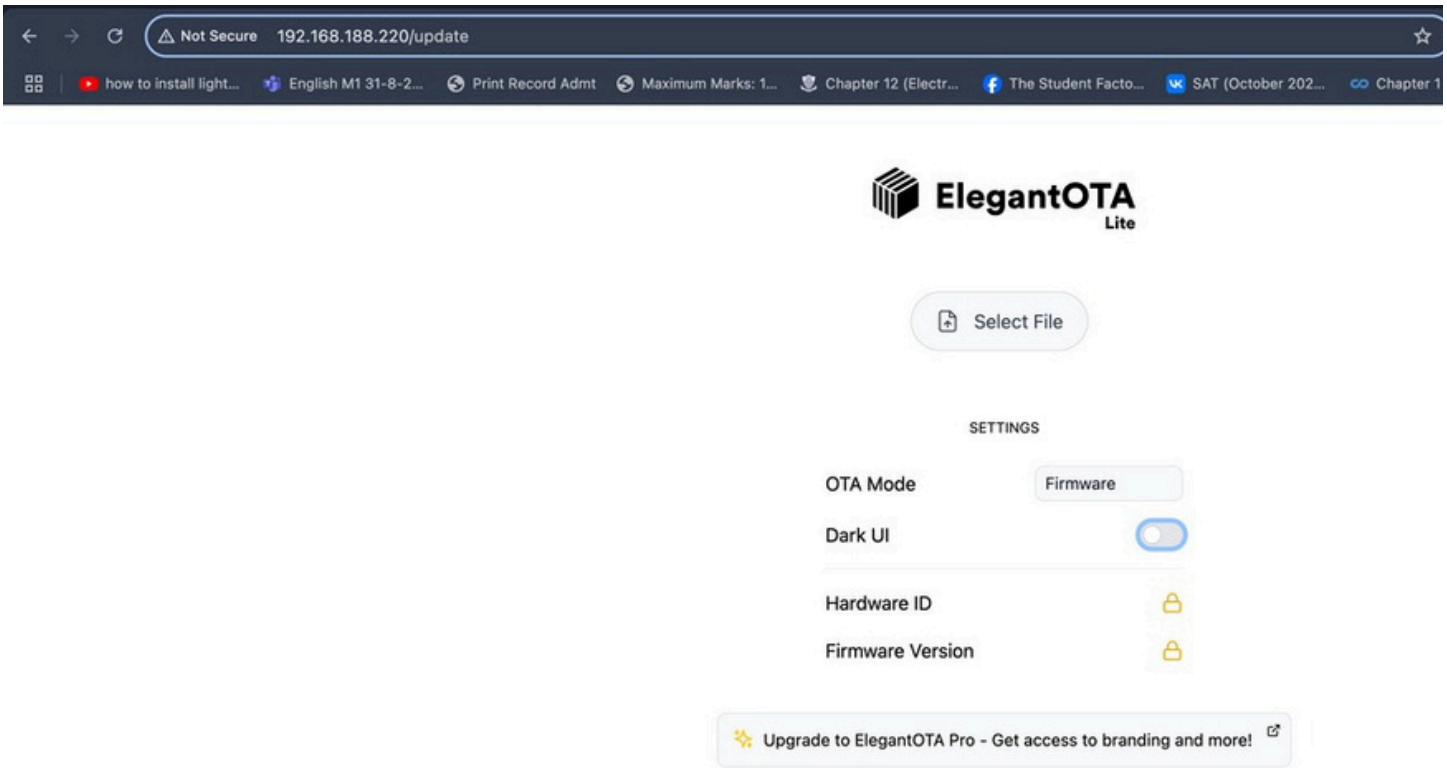
Version2:

```
#include <ESP8266WiFi.h>
#include <ElegantOTA.h>
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
ESP8266WebServer server(80);
// Non-blocking LED blink
static unsigned long previousMillis = 0; // Store last time LED was updated
const long interval = 500; // Interval for 1-second blink (on/off)
static bool ledState = false; // Track LED state (HIGH/LOW)
void setup() {
 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(1000);
 Serial.print(".");
 }
 Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
 ElegantOTA.begin(&server);
 server.begin();
}
void loop() {
 server.handleClient();
 ElegantOTA.loop();
 unsigned long currentMillis = millis(); // Get current time
 // Check if it's time to toggle the LED
 if (currentMillis - previousMillis >= interval) {
 ledState = !ledState; // Toggle state
 digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW); // Update LED
 previousMillis = currentMillis; // Save the current time
 }
}
```

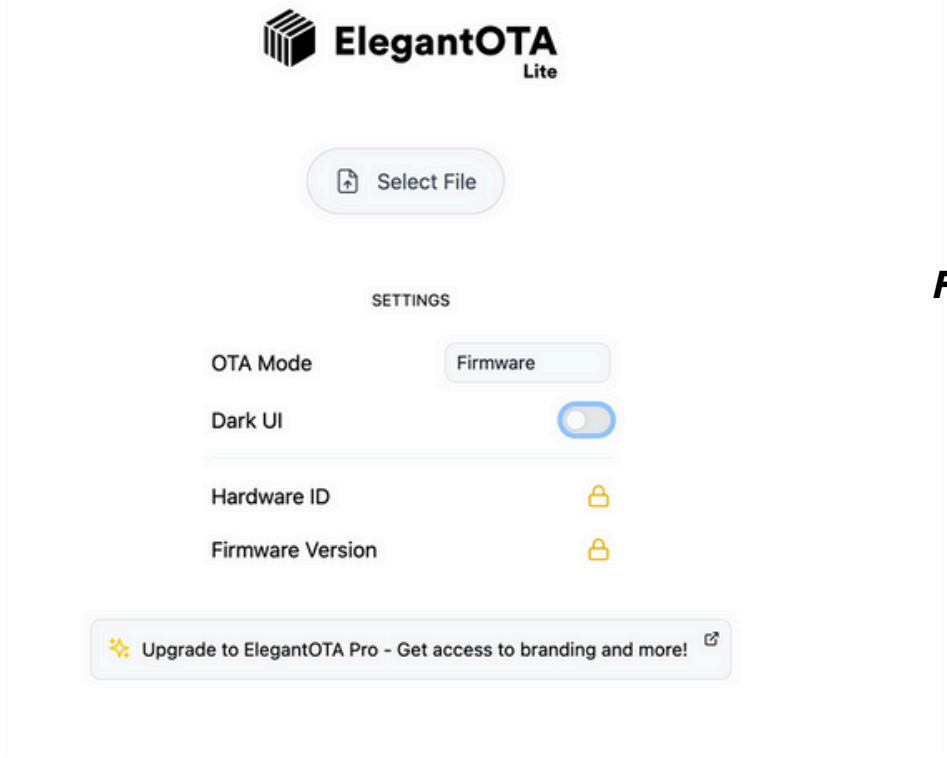
The LED was blinking with 1 second delay.

Version2:

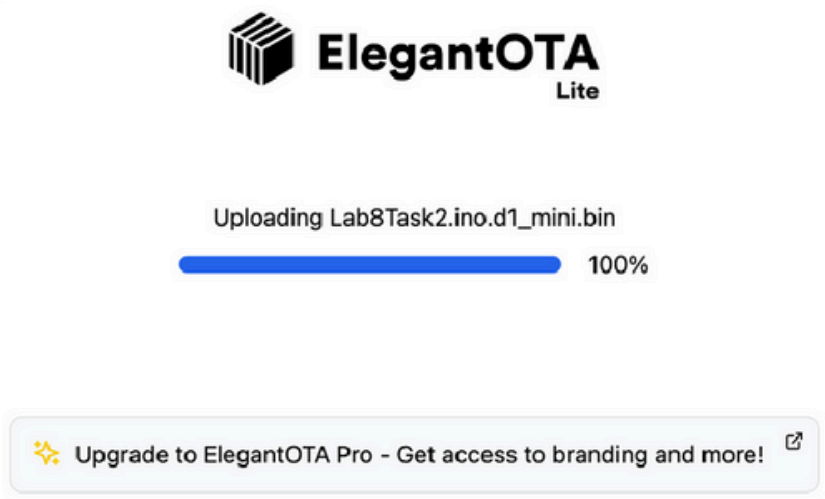
```
#include <ESP8266WiFi.h>
#include <ElegantOTA.h>
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
ESP8266WebServer server(80);
// Non-blocking LED blink
static unsigned long previousMillis = 0; // Store last time LED was updated
const long interval = 500; // Interval for 1-second blink (on/off)
static bool ledState = false; // Track LED state (HIGH/LOW)
void setup() {
 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(1000);
 Serial.print(".");
 }
 Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
 ElegantOTA.begin(&server);
 server.begin();
}
void loop() {
 server.handleClient();
 ElegantOTA.loop();
 unsigned long currentMillis = millis(); // Get current time
 // Check if it's time to toggle the LED
 if (currentMillis - previousMillis >= interval) {
 ledState = !ledState; // Toggle state
 digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW); // Update LED
 previousMillis = currentMillis; // Save the current time
 }
}
```



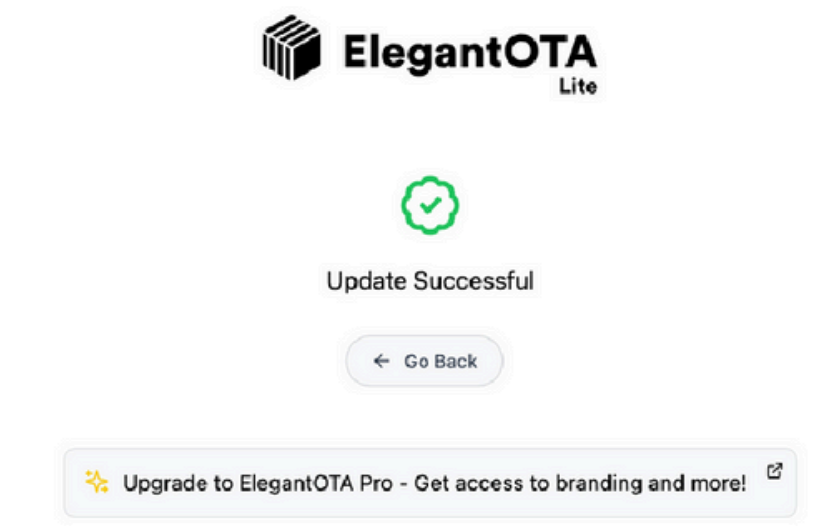
Using the ip address of esp8266 accessed the elegant OTA dashboard.



File Selection

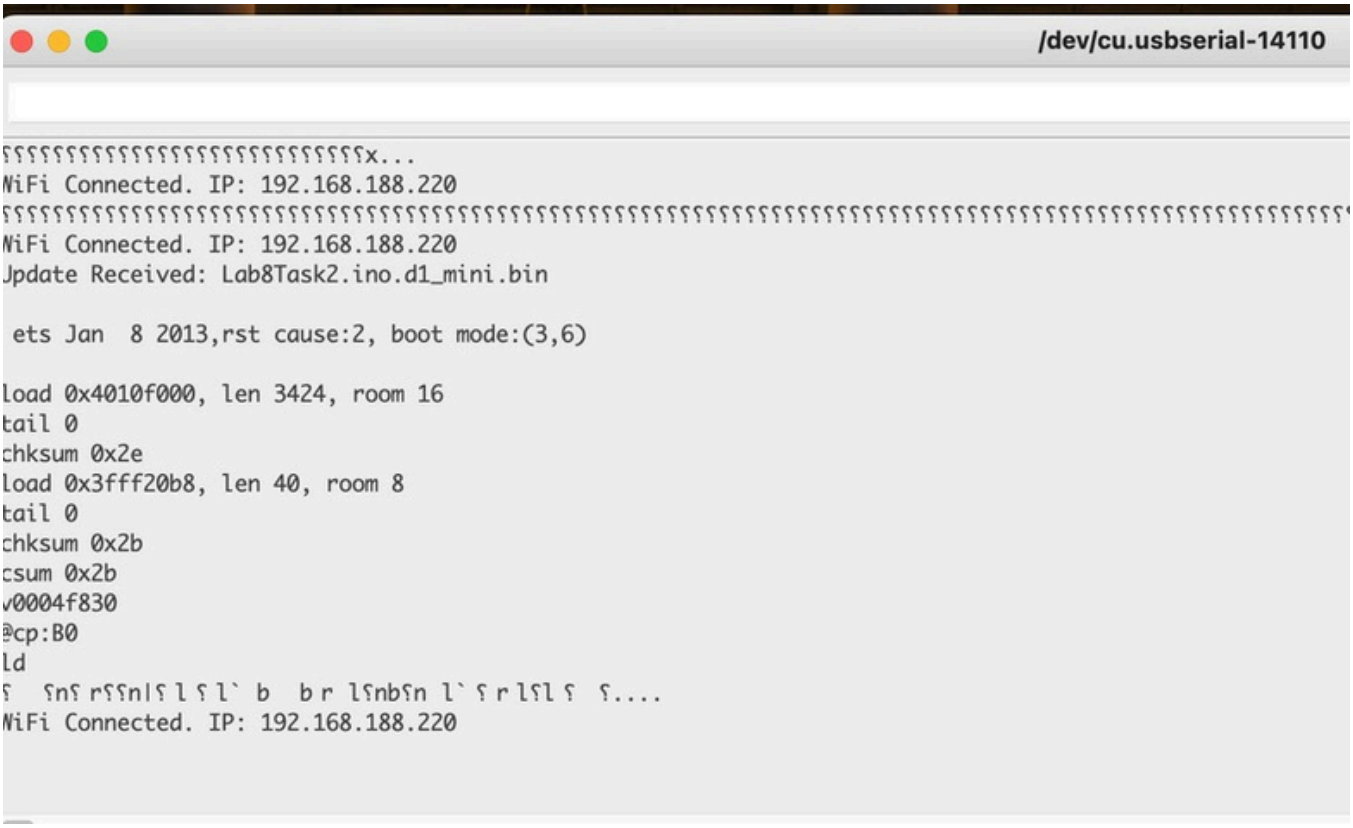


Choose the file



File uploaded successfully

LED is now blinking faster as the firmware has been updated.



Analysis of Code:

This code helps us update the ESP8266 over WiFi using a web page, with the help of the ElegantOTA library. First, it connects the ESP8266 to our WiFi network using the name (SSID) and password that we write at the top of the code. It keeps printing dots (.) on the Serial Monitor while it is trying to connect. When it finally connects, it shows the IP address of the device.

After the WiFi is ready, the code starts a small web server on the ESP8266. This server is needed for ElegantOTA to work. ElegantOTA makes it easy to update the device from a browser. We can just open the IP address in a browser, upload a new .bin firmware file, and the ESP8266 will install it.

The code also makes the onboard LED blink continuously without using delay(). It uses a timer to blink the LED every half a second. This helps show that the ESP is still running, even while the server is active. Overall, this code lets us upload new firmware wirelessly from a browser, and it also keeps the LED blinking as a sign that the device is working.

Short Description:

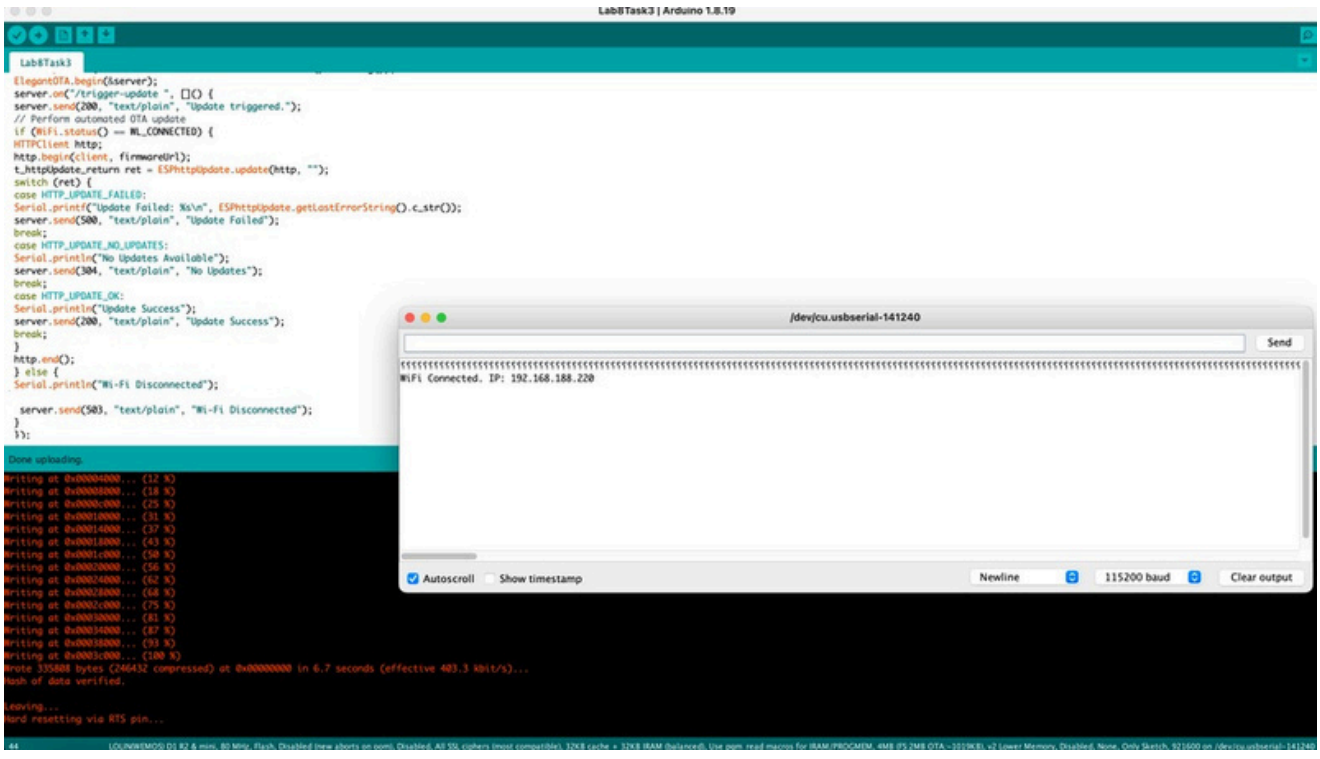
First, we wrote the code that connects the ESP8266 to WiFi and starts a web page using the ElegantOTA library. After uploading the code to the board using USB, we opened the Serial Monitor to see the IP address of the device. Then we typed that IP in my browser to open the update page. We uploaded a new .bin firmware file from there to update the device wirelessly.
To check if it worked, we changed in the code (like LED blink speed), made a new .bin file, and uploaded it again using the same web page. The LED started blinking differently, which proved the update was successful.

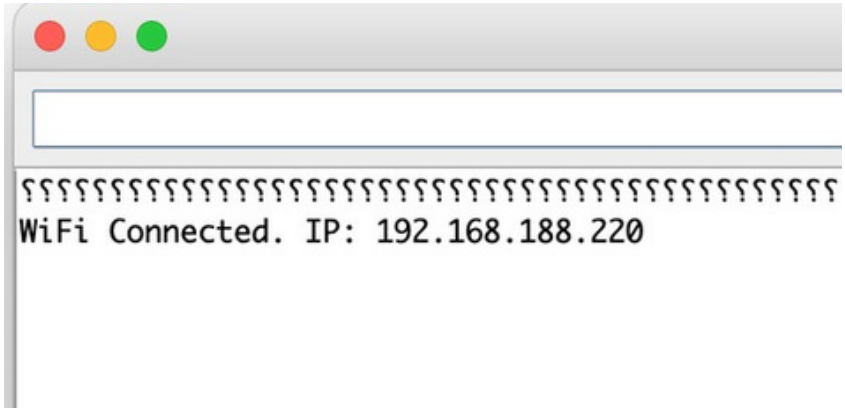
Lab Task 3: Automated OTA Updates with Node

Version 1:

```
#include <ESP8266WiFi.h>
#include <ElegantOTA.h>
#include <ESP8266HTTPClient.h> // Use <HTTPClient.h> for ESP32
#include <ESP8266httpUpdate.h> // Use <HTTPUpdate.h> for ESP32
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
const char* firmwareUrl = "http://192.168.188.103:8000//blink_v2.bin"; // firmware URL
ESP8266WebServer server(80);
WiFiClient client; // Create WiFiClient instance for HTTPClient
void setup() {

 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(1000);
 Serial.print(".");
 }
 Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
 ElegantOTA.begin(&server);
 server.on("/trigger-update", []() {
 server.send(200, "text/plain", "Update triggered.");
 // Perform automated OTA update
 if (WiFi.status() == WL_CONNECTED) {
 HTTPClient http;
 http.begin(client, firmwareUrl);
 t_httpUpdate_return ret = ESPhttpUpdate.update(http, "");
 switch (ret) {
 case HTTP_UPDATE_FAILED:
 Serial.printf("Update Failed: %s\n", ESPhttpUpdate.getLastErrorString().c_str());
 server.send(500, "text/plain", "Update Failed");
 break;
 case HTTP_UPDATE_NO_UPDATES:
```

```
Serial.println("No Updates Available");
server.send(304, "text/plain", "No Updates");
break;
case HTTP_UPDATE_OK:
Serial.println("Update Success");
server.send(200, "text/plain", "Update Success");
break;
}
http.end();
} else {
Serial.println("Wi-Fi Disconnected");
 server.send(503, "text/plain", "Wi-Fi Disconnected");
 }
});
server.begin();
}
// Non-blocking LED blink
static unsigned long previousMillis = 0; // Store last time LED was updated
const long interval = 1000; // Interval for 1-second blink (on/off)
static bool ledState = false; // Track LED state (HIGH/LOW)
void loop() {
server.handleClient();
ElegantOTA.loop();
unsigned long currentMillis = millis(); // Get current time
// Check if it's time to toggle the LED
if (currentMillis - previousMillis >= interval) {
ledState = !ledState; // Toggle state
digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW); // Update LED
previousMillis = currentMillis; // Save the current time
}
}
```

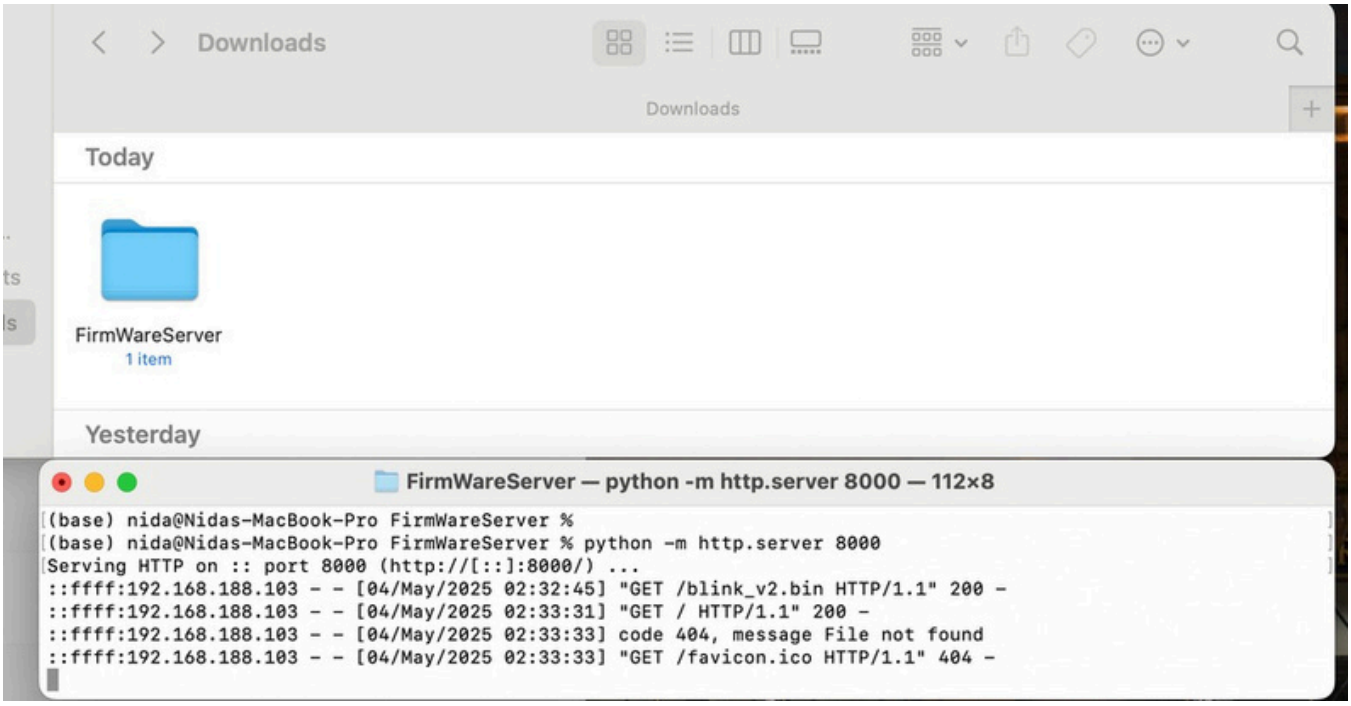


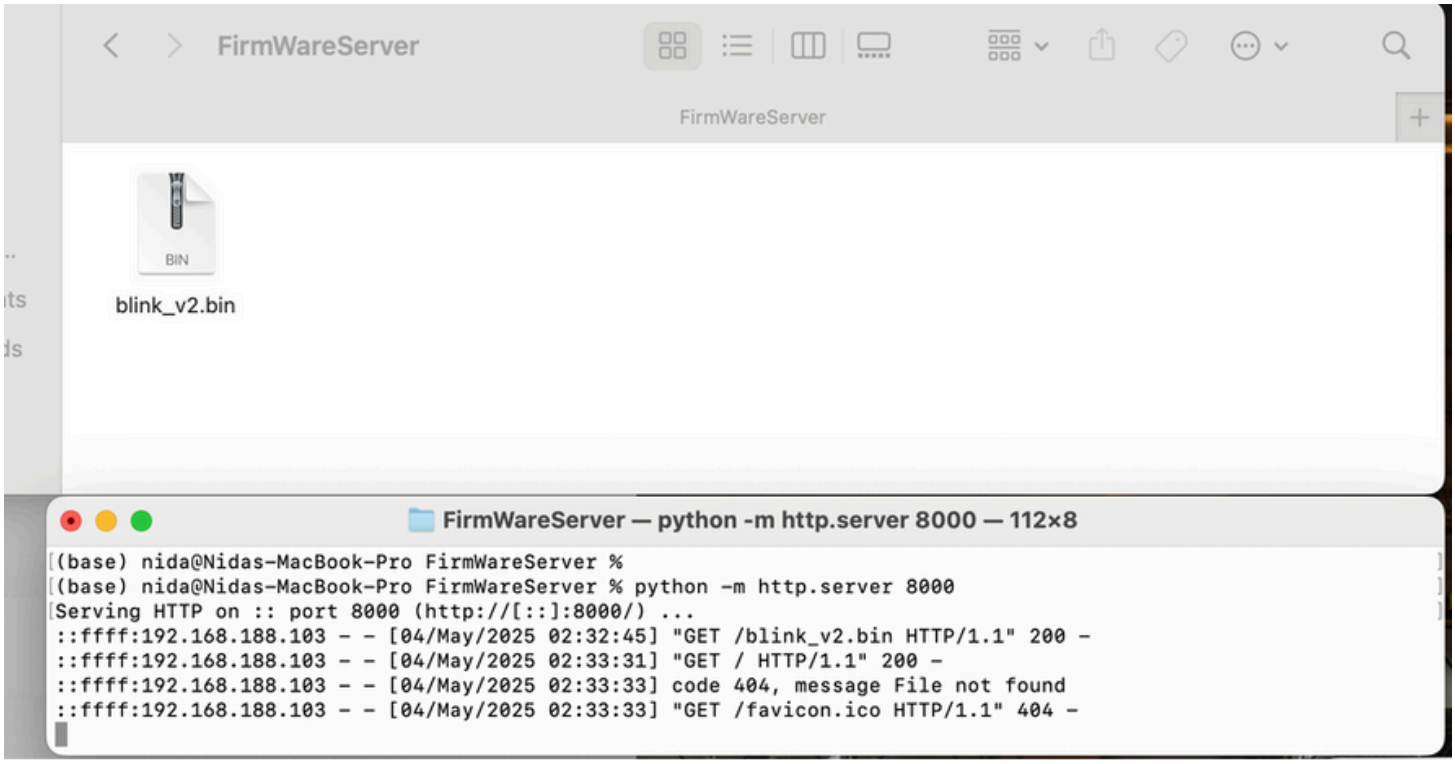


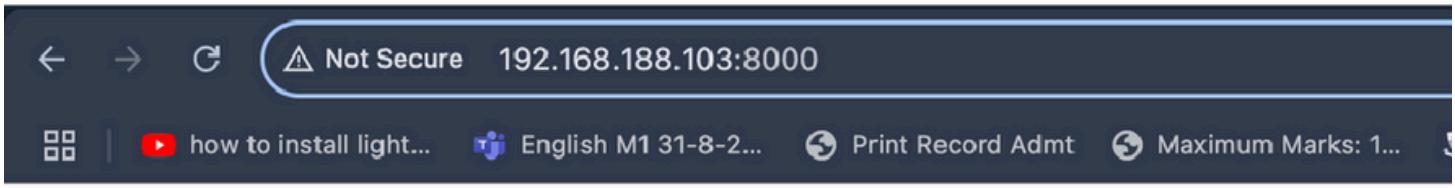
Version 2:

```
#include <ESP8266WiFi.h>
#include <ElegantOTA.h>
#include <ESP8266HTTPClient.h> // Use <HTTPClient.h> for ESP32
#include <ESP8266httpUpdate.h> // Use <HTTPUpdate.h> for ESP32
const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
const char* firmwareUrl = "http://192.168.188.103:8000//blink_v2.bin"; // firmware URL
ESP8266WebServer server(80);
WiFiClient client; // Create WiFiClient instance for HTTPClient
void setup() {
 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(1000);
 Serial.print(".");
 }
 Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());
 ElegantOTA.begin(&server);
 server.on("/trigger-update", []() {
 server.send(200, "text/plain", "Update triggered.");
 // Perform automated OTA update
 if (WiFi.status() == WL_CONNECTED) {
 HTTPClient http;
 http.begin(client, firmwareUrl);
 t_httpUpdate_return ret = ESPhttpUpdate.update(http, "");
 switch (ret) {
 case HTTP_UPDATE_FAILED:
 Serial.printf("Update Failed: %s\n", ESPhttpUpdate.getLastErrorString().c_str());
 server.send(500, "text/plain", "Update Failed");
 break;
 case HTTP_UPDATE_NO_UPDATES:
```

```
Serial.println("No Updates Available");
server.send(304, "text/plain", "No Updates");
break;
case HTTP_UPDATE_OK:
Serial.println("Update Success");
server.send(200, "text/plain", "Update Success");
break;
}
http.end();
} else {
Serial.println("Wi-Fi Disconnected");
 server.send(503, "text/plain", "Wi-Fi Disconnected");
 }
});
server.begin();
}
// Non-blocking LED blink
static unsigned long previousMillis = 0; // Store last time LED was updated
const long interval = 500; // Interval for 1-second blink (on/off)
static bool ledState = false; // Track LED state (HIGH/LOW)
void loop() {
server.handleClient();
ElegantOTA.loop();
unsigned long currentMillis = millis(); // Get current time
// Check if it's time to toggle the LED
if (currentMillis - previousMillis >= interval) {
ledState = !ledState; // Toggle state
digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW); // Update LED
previousMillis = currentMillis; // Save the current time
}
}
```

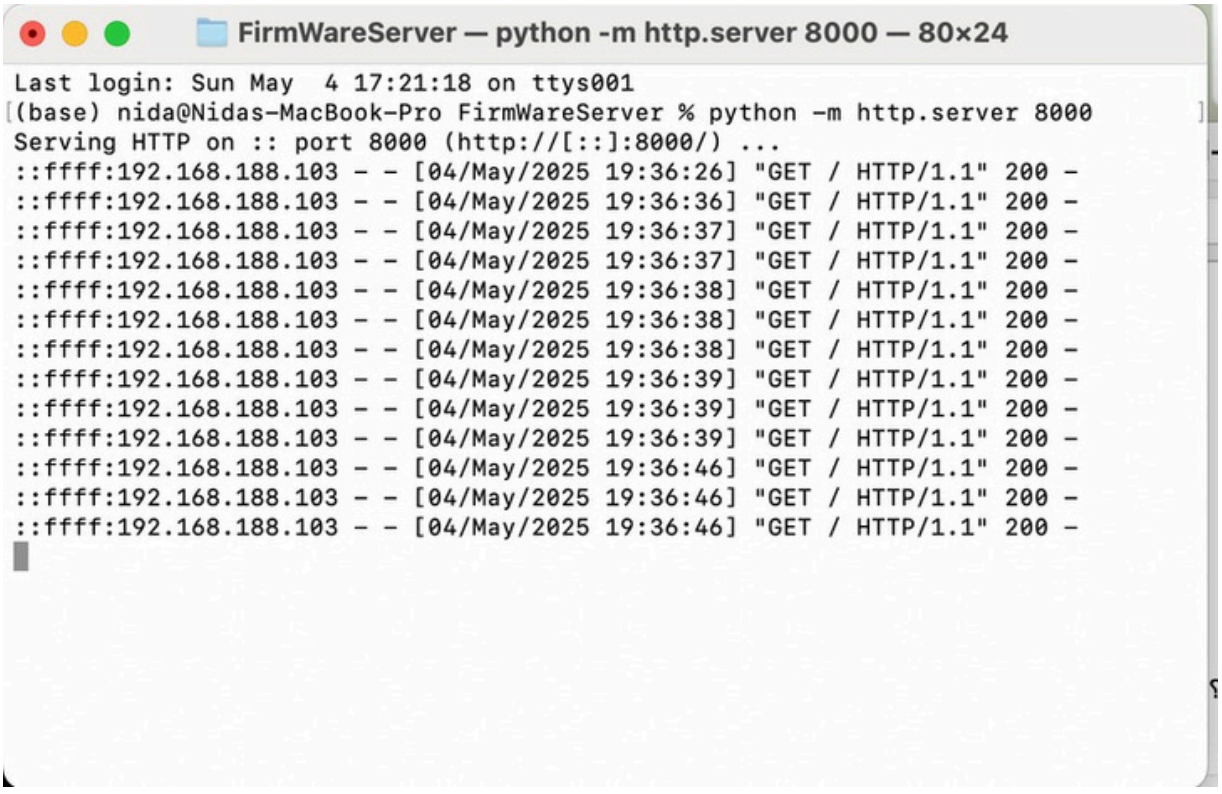






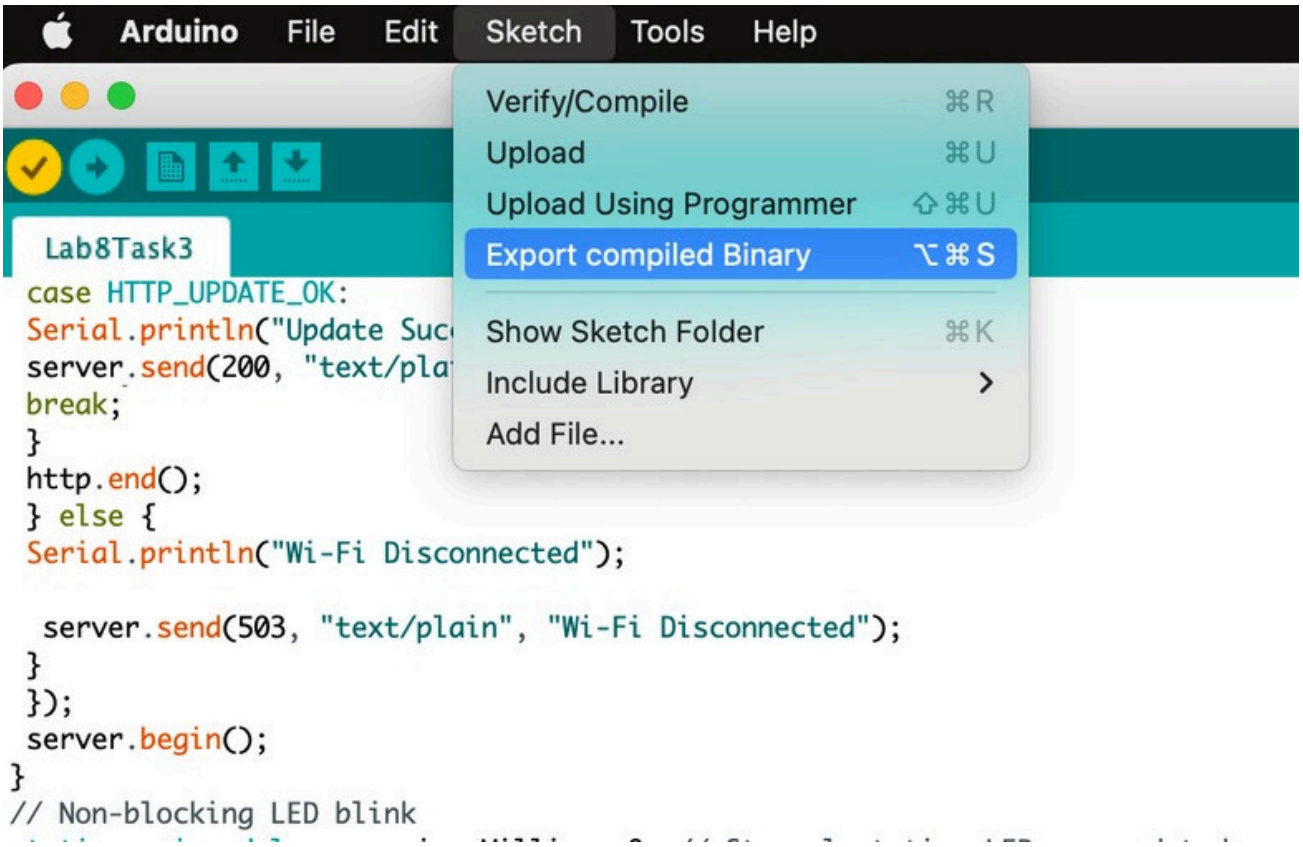
Directory listing for /

- blink_v2.bin

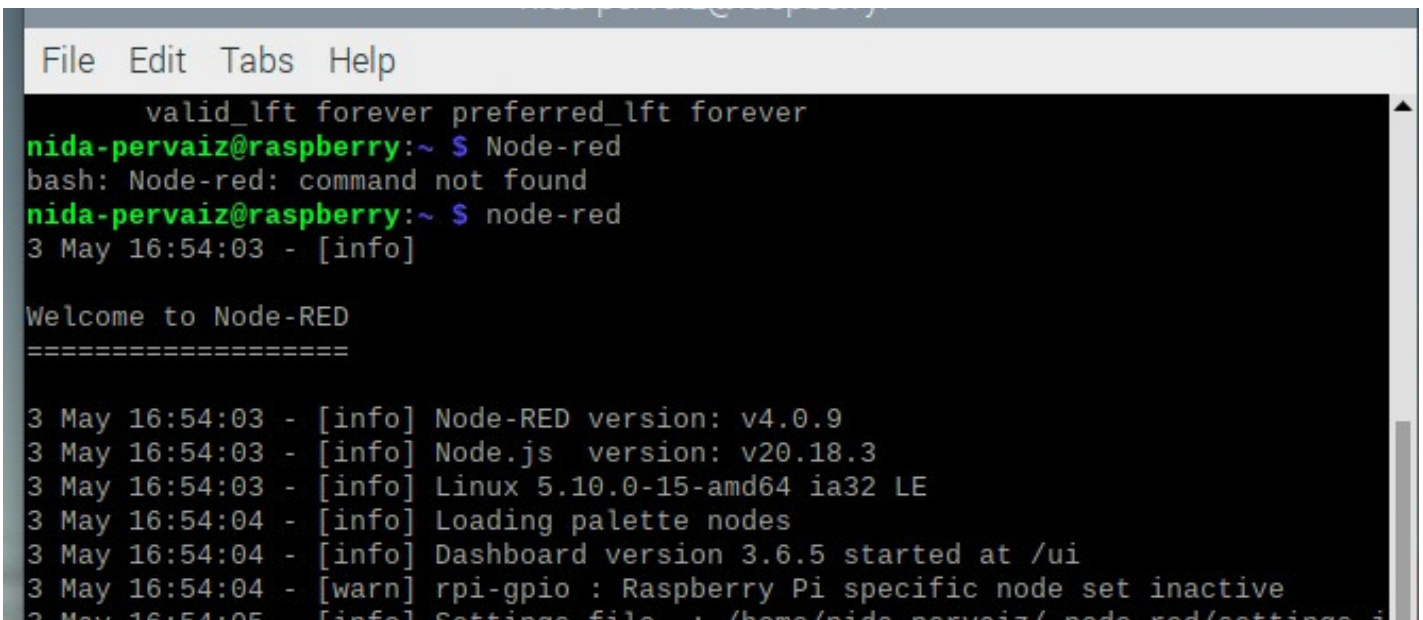


changed to 0.5 second

```
 }
 });
 server.begin();
}
// Non-blocking LED blink
static unsigned long previousMillis = 0; // Store last time LED was updated
const long interval = 500; // Interval for 1-second blink (on/off)
static bool ledState = false; // Track LED state (HIGH/LOW)
void loop() {
 server.handleClient();
 ElegantOTA.loop();
 unsigned long currentMillis = millis(); // Get current time
 // Check if it's time to toggle the LED
 if (currentMillis - previousMillis >= interval) {
 ledState = !ledState; // Toggle state
 digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW); // Update LED
 previousMillis = currentMillis; // Save the current time
 }
}
```



Started node-red in Raspberry pi OS

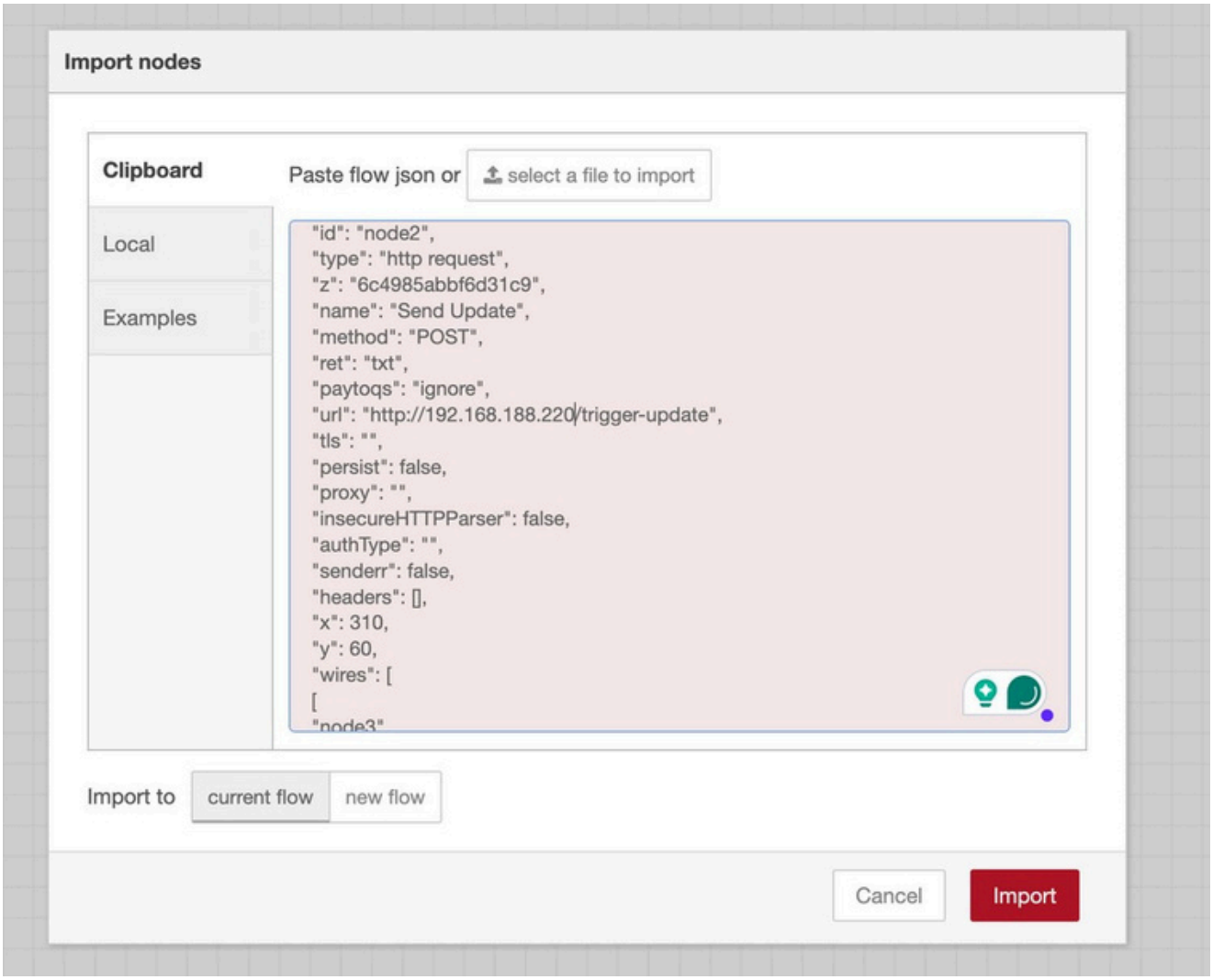


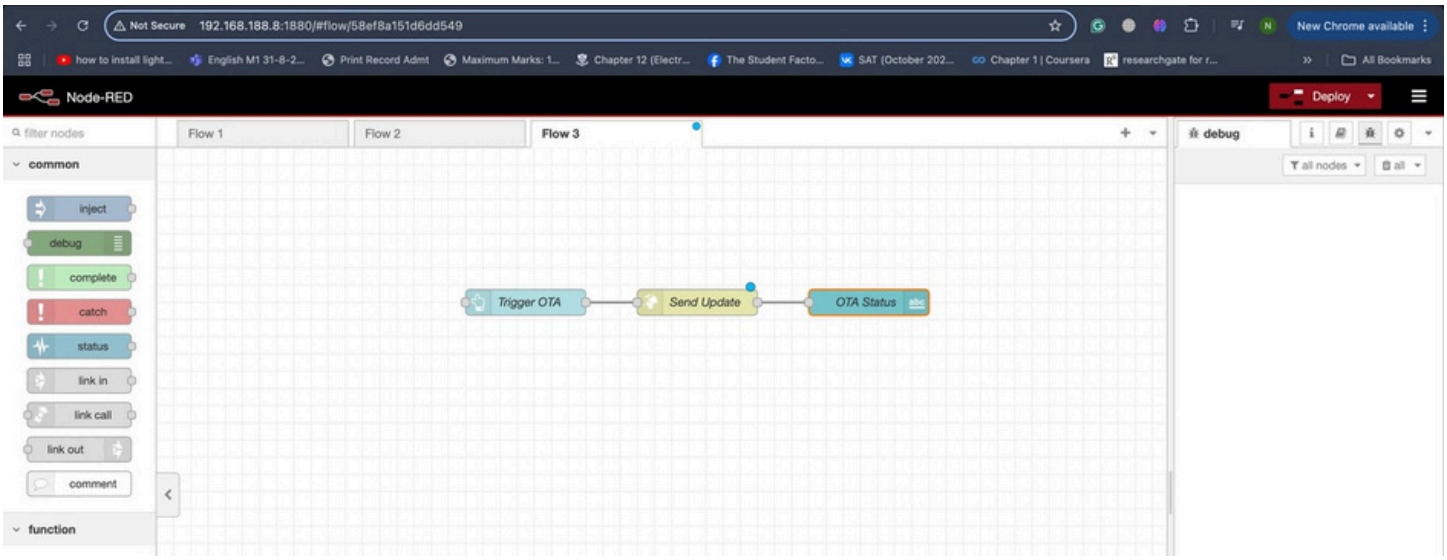
```
Lab8Task3
 break;
 case HTTP_UPDATE_OK:
 Serial.println("Update Success");
 server.send(200, "text/plain", "Update Success");
 break;
 }
 http.end();
 } else {
 Serial.println("Wi-Fi Disconnected");

  server.send(503, "text/plain", "Wi-Fi Disconnected");
 }
 });
 server.begin();
}
// Non-blocking LED blink
static unsigned long previousMillis = 0; // Store last time LED was updated
const long interval = 500; // Interval for 1-second blink (on/off)
static bool ledState = false; // Track LED state (HIGH/LOW)
void loop() {
 server.handleClient();
 ElegantOTA.loop();
 unsigned long currentMillis = millis(); // Get current time
 // Check if it's time to toggle the LED
 if (currentMillis - previousMillis >= interval) {
 ledState = !ledState; // Toggle state
 digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW); // Update LED
 previousMillis = currentMillis; // Save the current time
 }
}
```

```
 Variables and constants in RAM (global, static), used 29360 / 80192 bytes (36%)
║   SEGMENT  BYTES    DESCRIPTION
╠══ DATA     1508     initialized variables
╠══ RODATA   1428     constants
╚══ BSS      26424    zeroed variables
 Instruction RAM (IRAM_ATTR, ICACHE_RAM_ATTR), used 60447 / 65536 bytes (92%)
║   SEGMENT  BYTES    DESCRIPTION
╠══ ICACHE   32768    reserved space for flash instruction cache
╚══ IRAM     27679    code in IRAM
 Code in flash (default, ICACHE_FLASH_ATTR), used 301036 / 1048576 bytes (28%)
║   SEGMENT  BYTES    DESCRIPTION
╚══ IROM     301036   code in flash
29          LOLIN(WEMOS) D1 R2 & mini, 80 MHz, Flash, Disabled (new aborts on oom), Disabled, All SSL ciphers (most compatible), 32KB cach
```

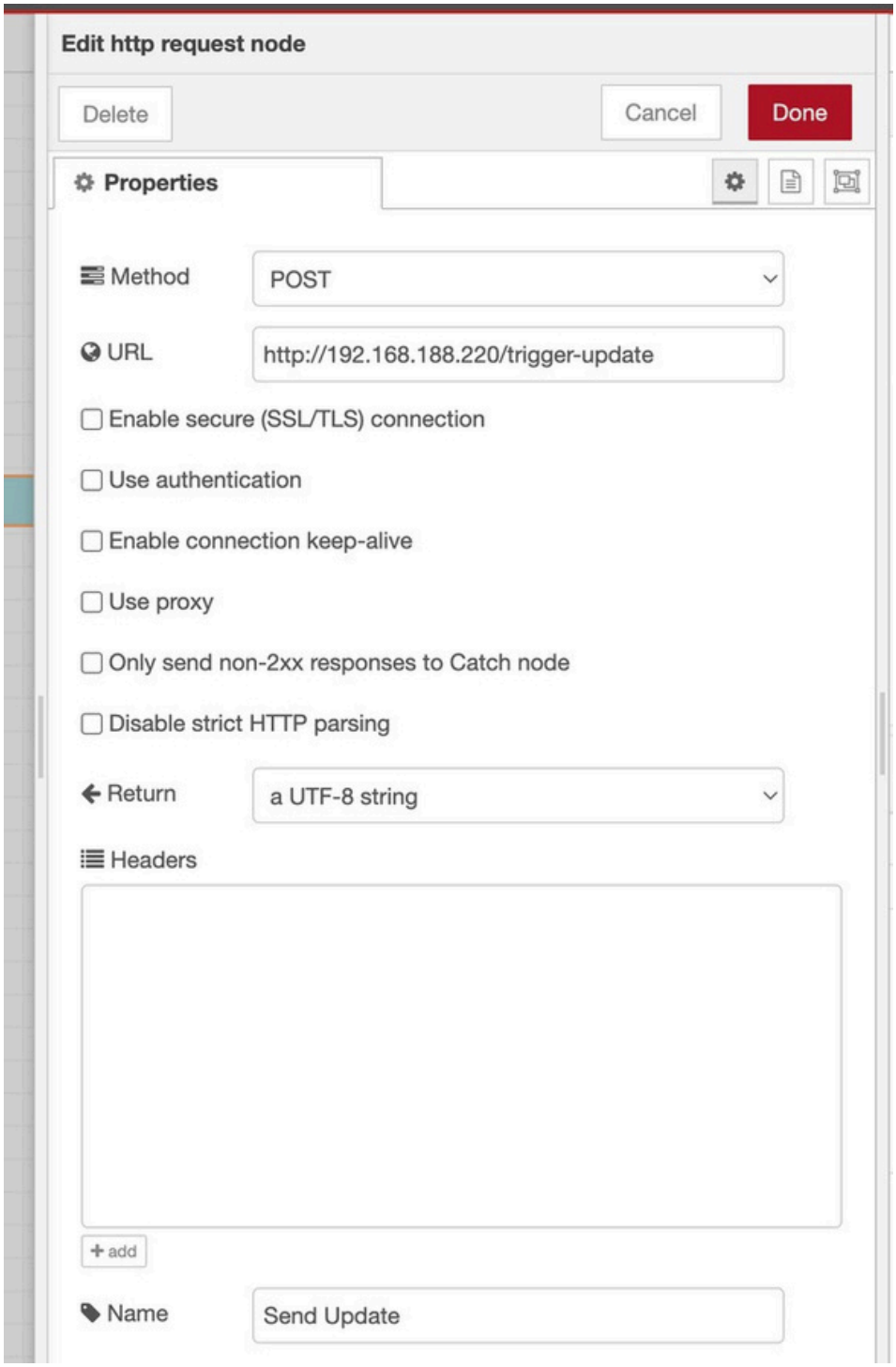
Imported the flow and set the esp ip

```
ʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕʕ
WiFi Connected. IP: 192.168.188.220
```



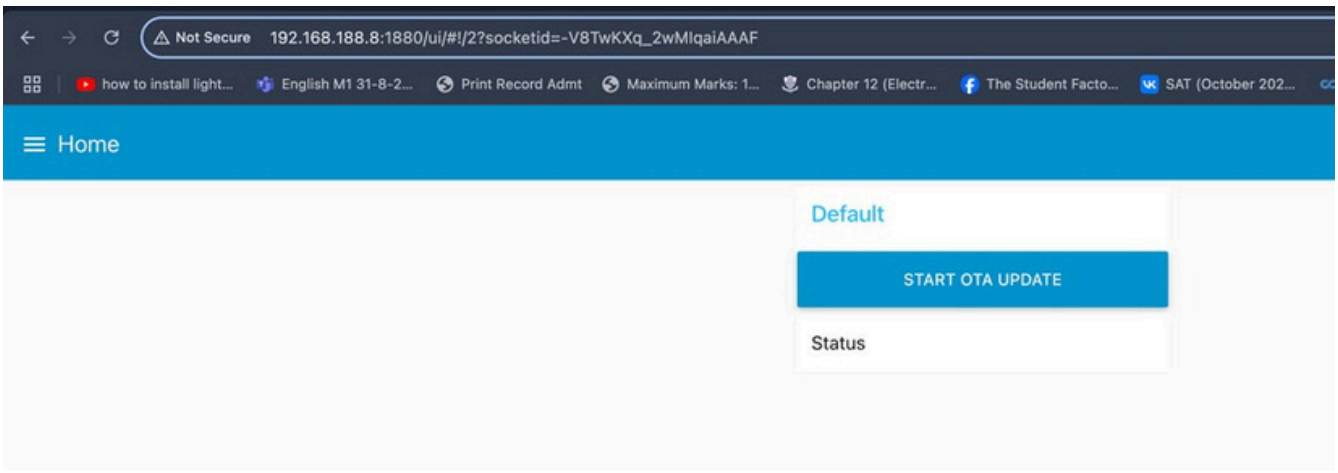


Dual verified that HTTP node is containing the device's (microcontroller esp8266's) ip address

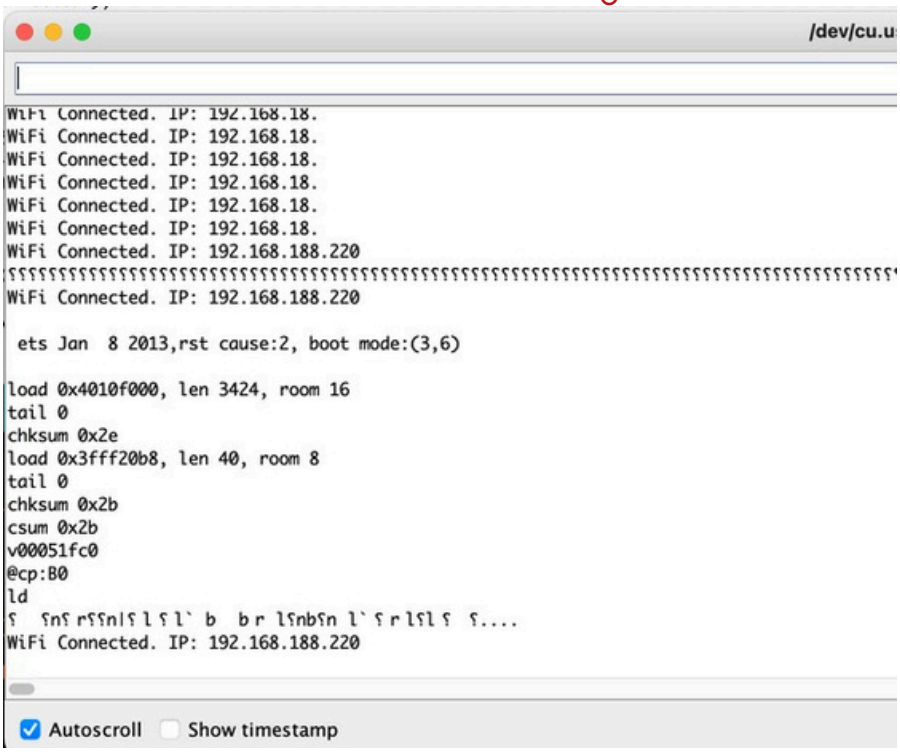

Edit http request node
Delete
Cancel
Done
Properties
Method
POST
URL
http://192.168.188.220/trigger-update
Enable secure (SSL/TLS) connection
Use authentication
Enable connection keep-alive
Use proxy
Only send non-2xx responses to Catch node
Disable strict HTTP parsing
Return
a UTF-8 string
Headers
+ add
Name
Send Update

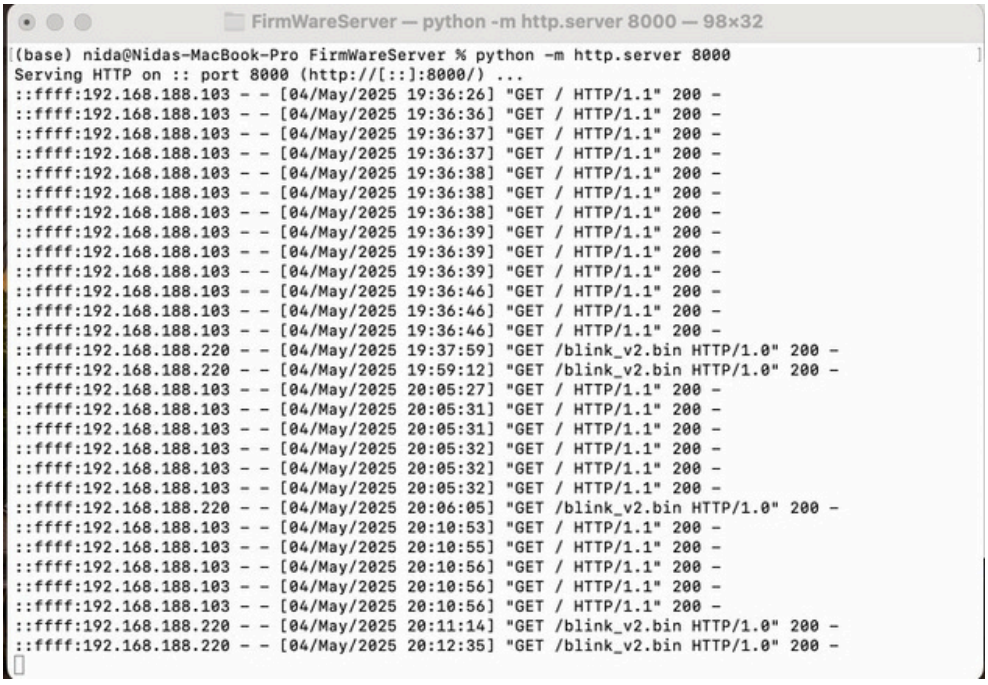

Successfully deployed
Flow 3

Accessed the UI



The esp got updated beacuse it was blinking faster than before update. The following screen shots also verify.





JSON code for flow:

[{ "id":"node1","type": "ui_button", "z": "6c4985abbf6d31c9", "name": "Trigger OTA", "group": "32afd9f10e5c45ba", "order": 1, "width": "", "height": "", "passthru": false, "label": "Start OTA Update", "tooltip": "", "color": "", "bgcolor": "", "className": "", "icon": "", "payload": "", "payloadType": "str", "topic": "", "topicType": "str", "x": 110, "y": 60, "wires": [["node2"]] }, { "id": "node2", "type": "http request", "z": "6c4985abbf6d31c9", "name": "Send Update", "method": "POST", "ret": "txt", "paytoqs": "ignore", "url": "http://192.168.188.220/trigger-update", "tls": "", "persist": false, "proxy": "", "insecureHTTPParser": false, "authType": "", "senderr": false, "headers": [], "x": 310, "y": 60, "wires": [["node3"]] }, { "id": "node3", "type": "ui_text", "z": "6c4985abbf6d31c9", "group": "32afd9f10e5c45ba", "order": 2, "width": "", "height": "", "name": "OTA Status", "label": "Status", "format": "", "layout": "", "className": "", "style": false, "font": "", "fontSize": "", "color": "#000000", "x": 510, "y": 60, "wires": [] }, { "id": "32afd9f10e5c45ba", "type": "ui_group", "name": "Default", "tab": "65cf22eab423bc27", "order": 1, "disp": true, "width": 6, "collapse": false, "className": "" }, { "id": "65cf22eab423bc27", "type": "ui_tab", "name": "Home", "icon": "dashboard", "disabled": false, "hidden": false }]

Analysis of Code:

Thiscode is made for the ESP8266 board. It lets the board connect to Wi-Fi, blink an LED, and also update its firmware (program) automatically from a website link. First, the code includes some libraries: ESP8266WiFi.h is used to connect the board to Wi-Fi. ElegantOTA.h lets us update the board using a browser. ESP8266HTTPClient.h and ESP8266httpUpdate.h help download and install new firmware from a web server (like a local Python server or hosting site).

In the setup() part, the board starts the Serial Monitor for printing messages. It tries to connect to Wi-Fi using the name and password. When it gets connected, it shows the IP address. Then ElegantOTA.begin() starts the OTA system on a webpage, and a new special page /trigger-update is added. When we visit this page in the browser, the board checks the firmware link (firmwareUrl). If it finds a new update, it downloads and installs it. If there is no update or an error, it shows a message. In the loop() part, the board checks for new page requests (server.handleClient()), runs the ElegantOTA system, and keeps blinking the built-in LED on and off every half second.

This code is useful because it lets us update the board without wires — just from a browser or a local server.

Short Description:

We created a newroute /trigger-update on the ESP server. When we visit this route, the ESP makes a request to a firmware link we gave (from our computer's local HTTP server, which we ran using Python on port 8000). This server had the .bin file (new firmware) ready for download. The ESP downloaded that file using the ESPhttpUpdate library and installed it.

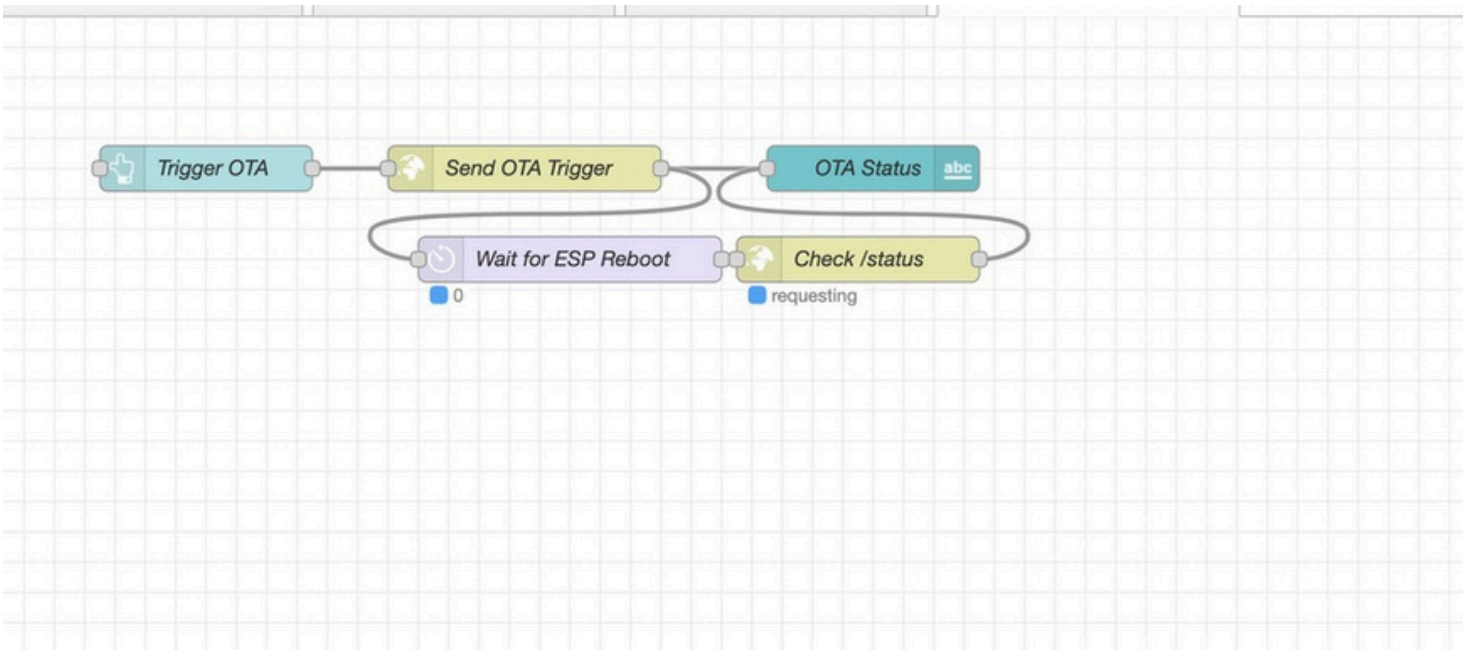
We used Node-RED to send a message that starts this process. When we pressed the button in Node-RED, it made an HTTP request to the ESP's /trigger-update route, which started the firmware update. On the Serial Monitor, we saw messages like "Update started..." which means the new firmware was installed successfully and the ESP was running again.

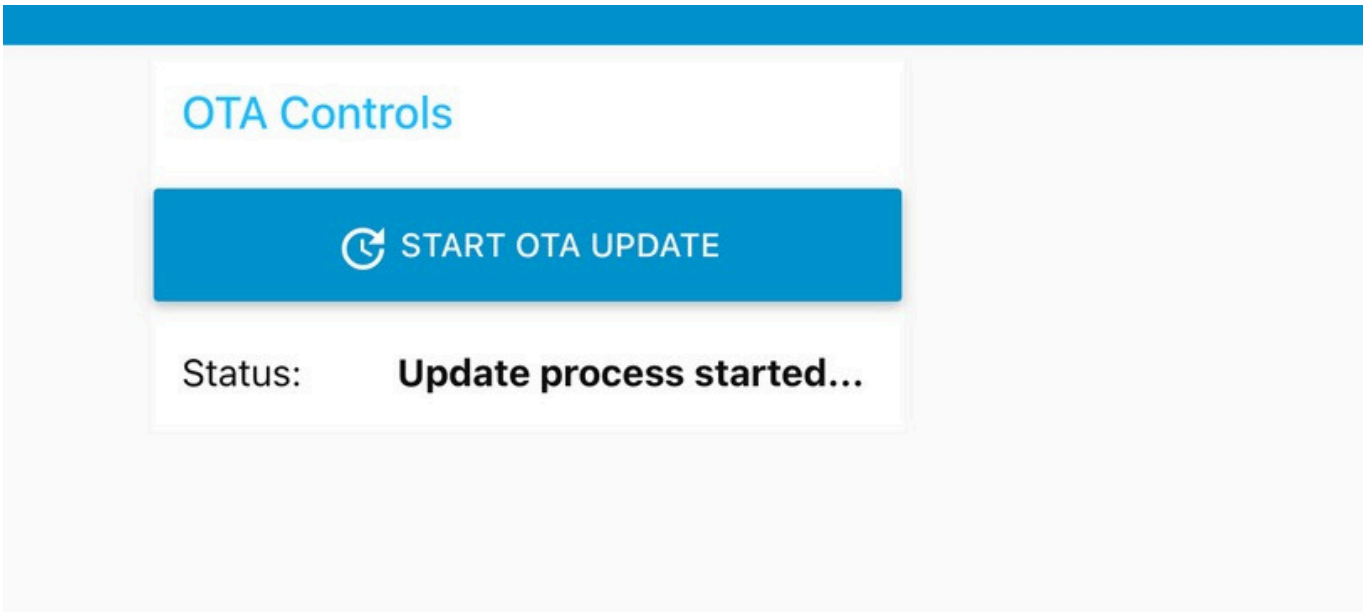
The LED on the ESP8266 also blinked faster after the update, which was another sign that the new code was working. One challenge we faced was that the status bar in Node-RED stayed empty and didn't show "Update successful." I added some delay and improved the messages sent back after the update, so now at least the "Device Online" status appears after reboot. This told me the update happened correctly, even if the full status wasn't shown.

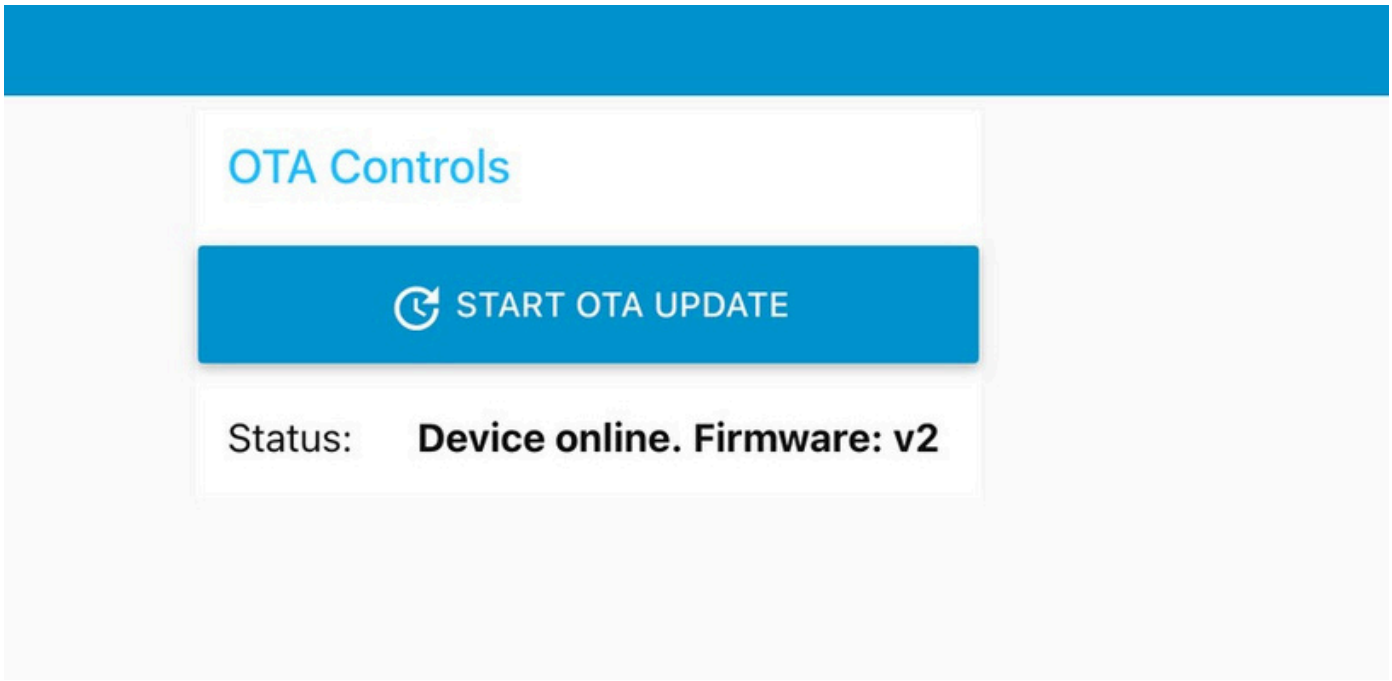
So the following all codes and screen shots is something we tried to show the status of esp on the status bar in node-red dashboard which was empty earlier

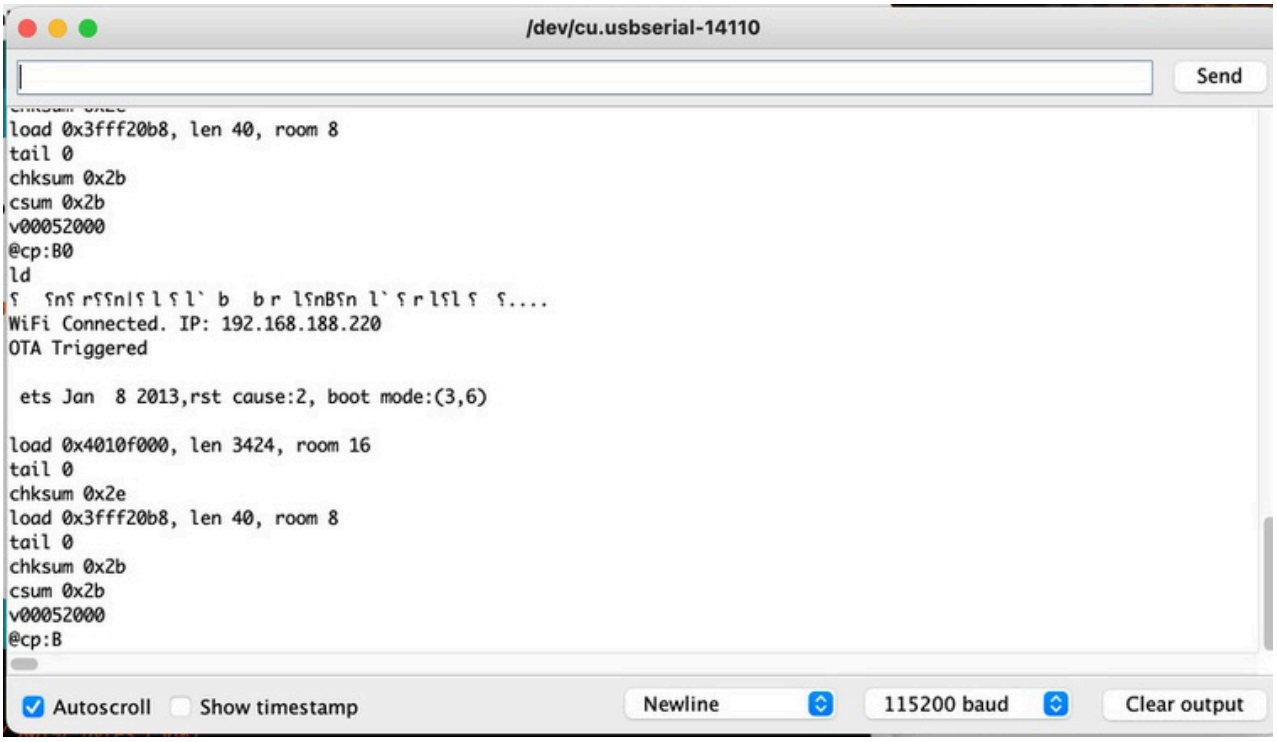

```
[
 {
 "id": "b1fdce5f38de7d63",
 "type": "tab",
 "label": "OTA Flow",
 "disabled": false,
 "info": ""
 },
 {
 "id": "9f2cfe45cfbcd641",
 "type": "ui_button",
 "z": "b1fdce5f38de7d63",
 "name": "Trigger OTA",
 "group": "20c929bcf2fc0ce2",
 "order": 1,
 "width": 6,
 "height": 1,
 "passthru": false,
 "label": "Start OTA Update",
 "tooltip": "",
 "color": "",
 "bgcolor": "",
 "className": "",
 "icon": "update",
 "payload": "",
 "payloadType": "str",
 "topic": "",
 "topicType": "str",
 "x": 160,
 "y": 100,
 "wires": [
 [
 "be8dd44aa4c279df"
 ]
 ]
 },
 {
 "id": "be8dd44aa4c279df",
 "type": "http request",
 "z": "b1fdce5f38de7d63",
 "name": "Send OTA Trigger",
 "method": "POST",
 "ret": "txt",
 "paytoqs": "ignore",
  "url": "http://192.168.188.220/trigger-update",
 "tls": "",
 "persist": false,
 "proxy": "",
 "insecureHTTPParser": false,
 "authType": "",
 "senderr": false,
 "headers": [],
 "x": 370,
 "y": 100,
 "wires": [
 [
 "c31e8c066e9146c0",
 "09b54721fc20f1a4"
 ]
 ]
 },
 {
 "id": "c31e8c066e9146c0",
 "type": "ui_text",
 "z": "b1fdce5f38de7d63",
 "group": "20c929bcf2fc0ce2",
 "order": 2,
 "width": 6,
 "height": 1,
 "name": "OTA Status",
 "label": "Status:",
 "format": "{{msg.payload}}",
 "layout": "row-spread",
 "className": "",
 "style": {
 "color": "#000000"
 },
 "x": 600,
 "y": 100,
 "wires": []
 },
 {
 "id": "09b54721fc20f1a4",
 "type": "delay",
 "z": "b1fdce5f38de7d63",
 "name": "Wait for ESP Reboot",
 "pauseType": "delay",
 "timeout": "12",
 "timeoutUnits": "seconds",
 "rate": "1",
 "nbRateUnits": "1",
 "rateUnits": "second",
 "randomFirst": "1",
 "randomLast": "5",
 "randomUnits": "seconds",
 "drop": false,
 "outputs": 1,
 "x": 400,
 "y": 160,
 "wires": [
 [
 "36db73243a301790"
 ]
 ]
 },
 {
 "id": "36db73243a301790",
 "type": "http request",
 "z": "b1fdce5f38de7d63",
 "name": "Check /status",
 "method": "GET",
 "ret": "txt",
 "paytoqs": "ignore",
 "url": "http://192.168.188.220/status",
 "tls": "",
 "persist": false,
 "proxy": "",
 "insecureHTTPParser": false,
 "authType": "",
 "senderr": false,
 "headers": [],
 "x": 590,
 "y": 160,
 "wires": [
 [
 "c31e8c066e9146c0"
 ]
 ]
 },
 {
 "id": "20c929bcf2fc0ce2",
 "type": "ui_group",
 "name": "OTA Controls",
 "tab": "1a65f1d21ab51c29",
 "order": 1,
 "disp": true,
 "width": 6,
 "collapse": false,
 "className": ""
 },
 {
 "id": "1a65f1d21ab51c29",
 "type": "ui_tab",
 "name": "OTA Dashboard",
 "icon": "memory",
 "order": 1,
 "disabled": false,
 "hidden": false
 }

]
```

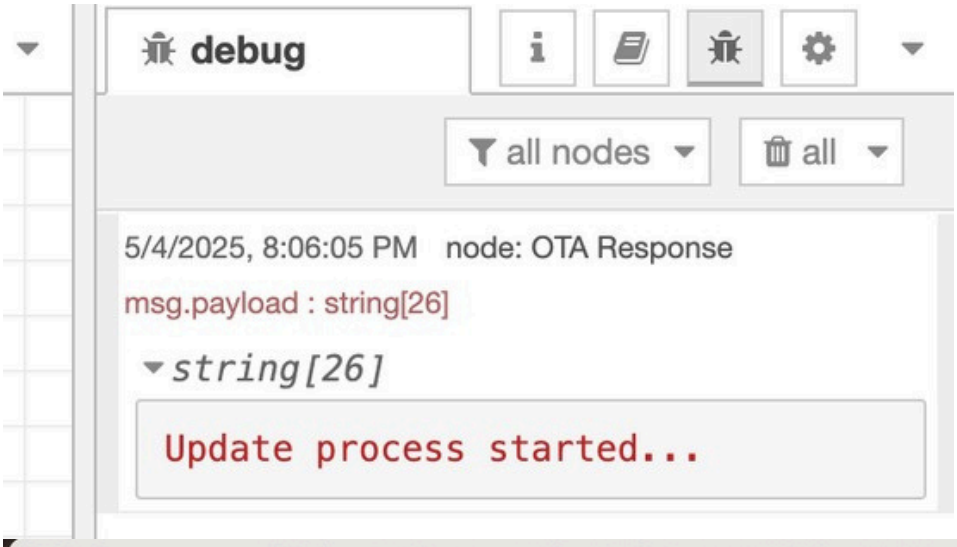

It is JSON code for the flow

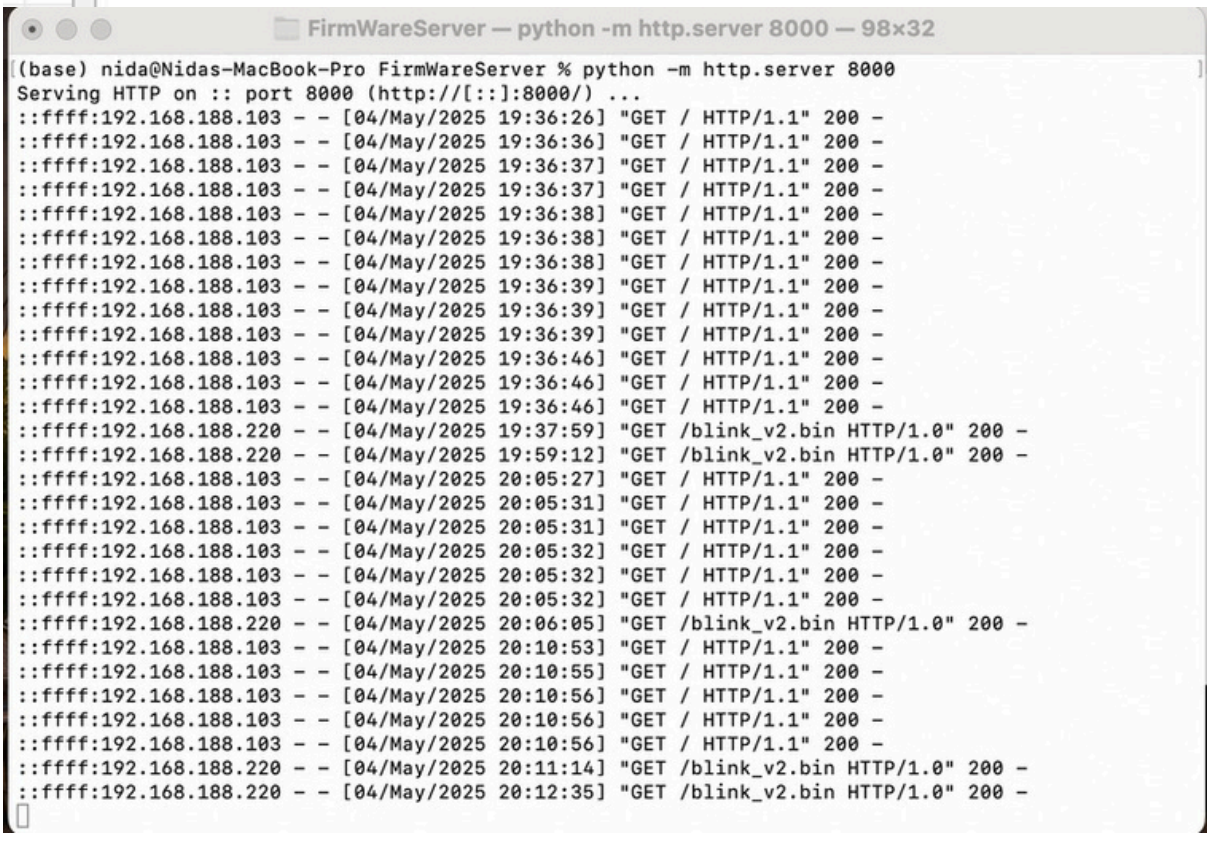

Trigger OTA
Send OTA Trigger
OTA Status
abc
Wait for ESP Reboot
Check /status
0
requesting


OTA Controls
START OTA UPDATE
Status:
Update process started...


OTA Controls
START OTA UPDATE
Status:
Device online. Firmware: v2


/dev/cu.usbserial-14110
Send
load 0x3fff20b8, len 40, room 8
tail 0
chksum 0x2b
csum 0x2b
v00052000
@cp:B0
ld
WiFi Connected. IP: 192.168.188.220
OTA Triggered
ets Jan 8 2013,rst cause:2, boot mode:(3,6)
load 0x4010f000, len 3424, room 16
tail 0
chksum 0x2e
load 0x3fff20b8, len 40, room 8
tail 0
chksum 0x2b
csum 0x2b
v00052000
@cp:B
Autoscroll
Show timestamp
Newline
115200 baud
Clear output


debug
all nodes
all
5/4/2025, 8:06:05 PM node: OTA Response
msg.payload : string[26]
string[26]
Update process started...


FirmWareServer — python -m http.server 8000 — 98×32
(base) nida@Nidas-MacBook-Pro FirmWareServer % python -m http.server 8000
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:192.168.188.103 - - [04/May/2025 19:36:26] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:36] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:37] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:37] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:38] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:38] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:38] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:39] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:39] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:39] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:46] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:46] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 19:36:46] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.220 - - [04/May/2025 19:37:59] "GET /blink_v2.bin HTTP/1.0" 200 -
::ffff:192.168.188.220 - - [04/May/2025 19:59:12] "GET /blink_v2.bin HTTP/1.0" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:05:27] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:05:31] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:05:31] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:05:32] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:05:32] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:05:32] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.220 - - [04/May/2025 20:06:05] "GET /blink_v2.bin HTTP/1.0" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:10:53] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:10:55] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:10:56] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:10:56] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.103 - - [04/May/2025 20:10:56] "GET / HTTP/1.1" 200 -
::ffff:192.168.188.220 - - [04/May/2025 20:11:14] "GET /blink_v2.bin HTTP/1.0" 200 -
::ffff:192.168.188.220 - - [04/May/2025 20:12:35] "GET /blink_v2.bin HTTP/1.0" 200 -

Code:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ElegantOTA.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266httpUpdate.h>

const char* ssid = "Galaxy A13 7AC7";
const char* password = "ucno4258";
const char* firmwareUrl = "http://192.168.188.103:8000/blink_v2.bin"; // Your OTA .bin file

ESP8266WebServer server(80);
WiFiClient client;

void setup() {
 Serial.begin(115200);
 pinMode(LED_BUILTIN, OUTPUT);

 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
   delay(1000);
   Serial.print(".");
  }
 Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());

 ElegantOTA.begin(&server);

 // Trigger OTA
 server.on("/trigger-update", []() {
   server.send(200, "text/plain", "Update process started...");
   Serial.println("OTA Triggered");

   if (WiFi.status() == WL_CONNECTED) {
     HTTPClient http;
     http.begin(client, firmwareUrl);

     t_httpUpdate_return ret = ESPhttpUpdate.update(http, "");

 s w itch (ret) {
 case HTTP_UPDATE_FAILED:
 Serial.printf("Update Failed: %s\n", ESPhttpUpdate.getLastErrorString().c_str());
 break;
 case HTTP_UPDATE_NO_UPDATES:
 Serial.println("No Updates Available");
 break;
 case HTTP_UPDATE_OK:
 Serial.println("Update Successful");
 break;
 }
```

```
http.end();
}else {
Serial.println("Wi-Fi Disconnected");
}
});
//Status Endpoint
server.on("/status", []() {
server.send(200, "text/plain", "Device online. Firmware: v2");
});
server.begin();
}
static unsigned long previousMillis = 0;
const long interval = 500;
static bool ledState = false;
void loop() {
server.handleClient();
ElegantOTA.loop();
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
ledState = !ledState;
digitalWrite(LED_BUILTIN, ledState ? HIGH : LOW);
previousMillis = currentMillis;
}
}
```

Description:

This code connects the ESP8266 to Wi-Fi and sets up a web server that handles firmware updates over the air (OTA) using a local .bin file. It includes the ElegantOTA library for web-based updates and also allows automatic updates when the /trigger-update path is visited. The ESP sends status back over Serial Monitor depending on whether the update was successful, failed, or no update was found. The blinking LED helps confirm the device is still responsive. There's also a /status route that returns the message "Device online. Firmware: v2", which means the ESP is running the updated code and is successfully connected to the network after reboot. This status is useful to confirm the ESP is alive and running the new firmware after an OTA update.

This was extra stuff from apart from the manual

END