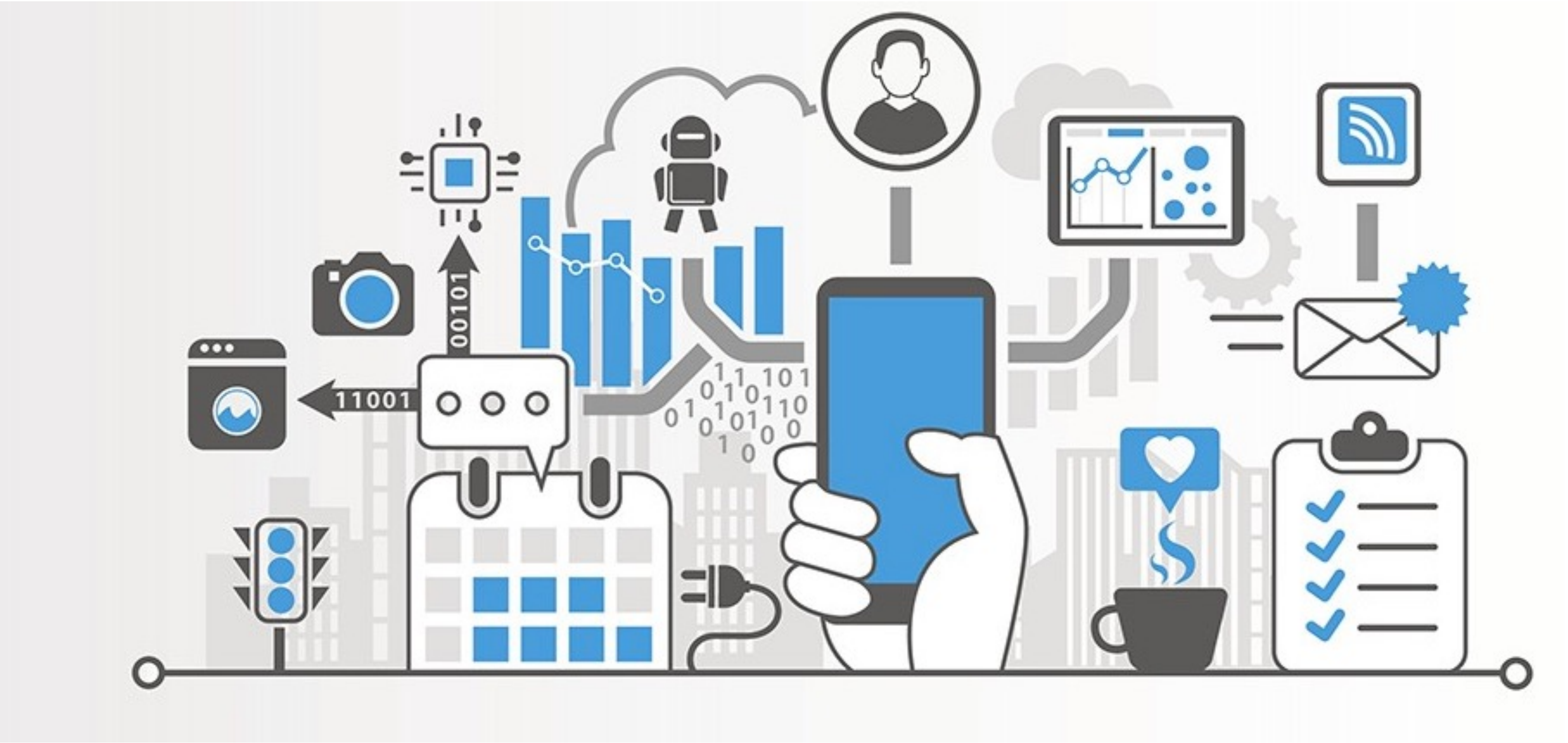
Tecnologie per IoT

Daniele Jahier Pagliari

Lab1: Hardware



Overview

- All three exercises in Part3 use the same HW components
  - One of the two LEDs *(or the internal LED)*
  - The temperature sensor *(or the internal temperature sensor)*
  - The Serial interface (for debugging and error reporting)
  - The **WiFi interface** of the Arduino
- The goal is to let the Arduino communicate via REST and MQTT

PART3: EXERCISE 1

The WiFiNINA Library

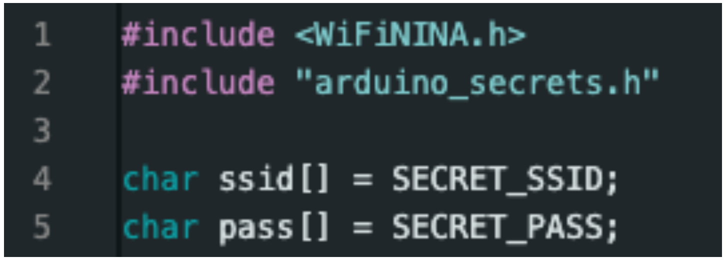
- Used to control the WiFi and BLE Interface of the RP2040.

- Reference: here

- In this exercise we will use two components of the library
  - WiFiServer (to setup an HTTP server on the Arduino)
  - WiFiClient (to handle individual requests to the server)

New Code Elements: Global

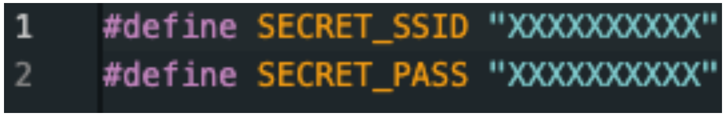
- Include required libraries and define the SSID (aka “network name”) and Password of the WiFi network you want to connect to.



```
#include <WiFiNINA.h>
#include "arduino_secrets.h"

char ssid[] = SECRET_SSID;
char pass[] = SECRET_PASS;
```

In the lab, you must use your phone’s hotspot. The “Polito” WiFi won’t work.

  - The best practice suggested by Arduino is to define your SSID and Password in a different header file (“arduino_secrets.h”)
  - Better than defining them in the .ino “main file”, because it’s more difficult to accidentally share “secrets”.

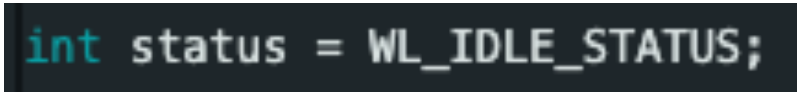


```
#define SECRET_SSID "XXXXXXXXXX"
#define SECRET_PASS "XXXXXXXXXX"
```

  - Careful, this is not really secure!!! Delete this file after testing, and **don’t send me your WiFi passwords when you upload the labs!**

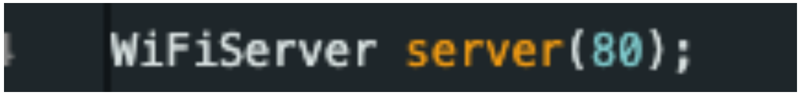
New Code Elements: Global

- Define a global variable to hold the WiFi connection status.



```
int status = WL_IDLE_STATUS;
```

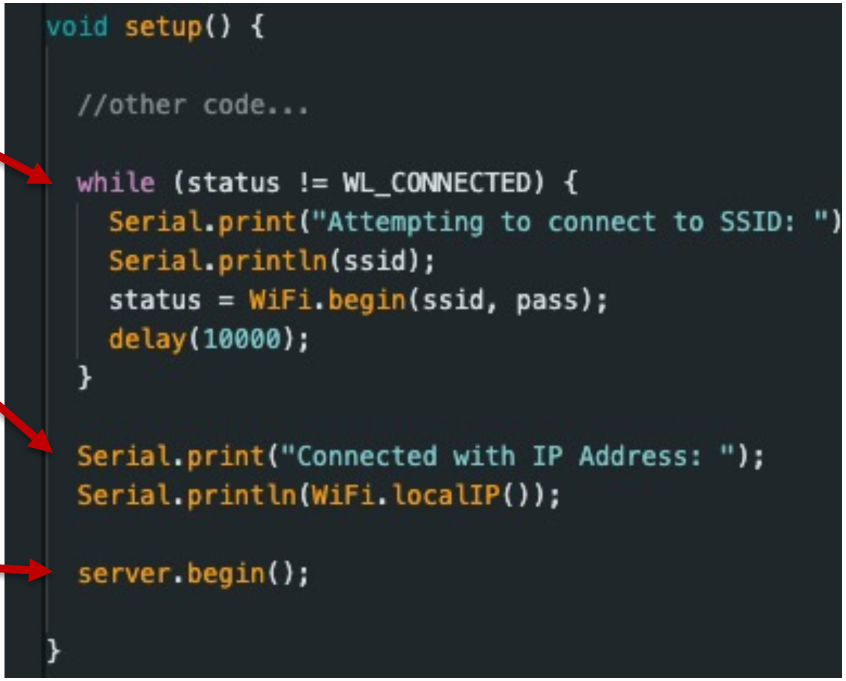
- Define a WiFiServer object that will listen on Port 80 (HTTP)



```
WiFiServer server(80);
```

New Code Elements: Setup

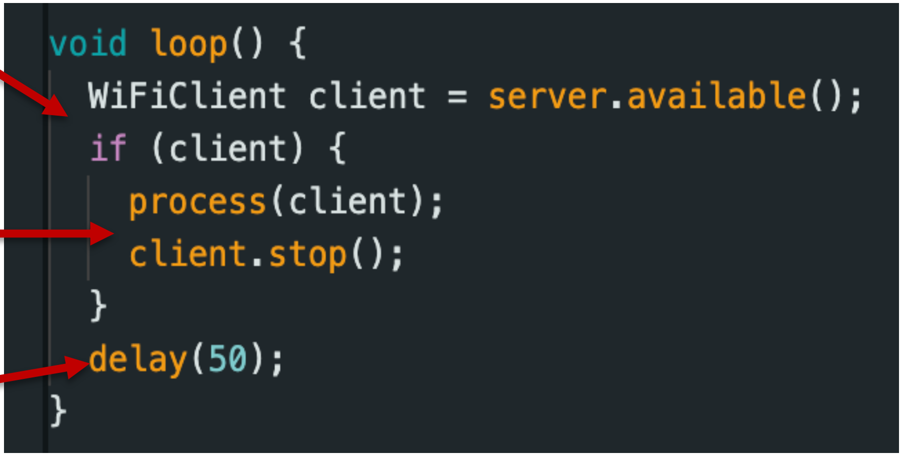
- In setup():
  1. Try connecting to the WiFi until you succeed.
  2. When connected, print the IP address assigned by DHCP on the Serial.
     - You need to know the IP address in order to connect to the Arduino from your PC.
  3. Start the server on Port 80



```
void setup() {

  //other code...

  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ")
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
    delay(10000);
  }

  Serial.print("Connected with IP Address: ");
  Serial.println(WiFi.localIP());

  server.begin();

}
```

New Code Elements: Loop

- In loop():
  1. Check if a connection (client) is available
  2. Process the incoming connection and terminate it
  3. Wait for 50ms to avoid overloading



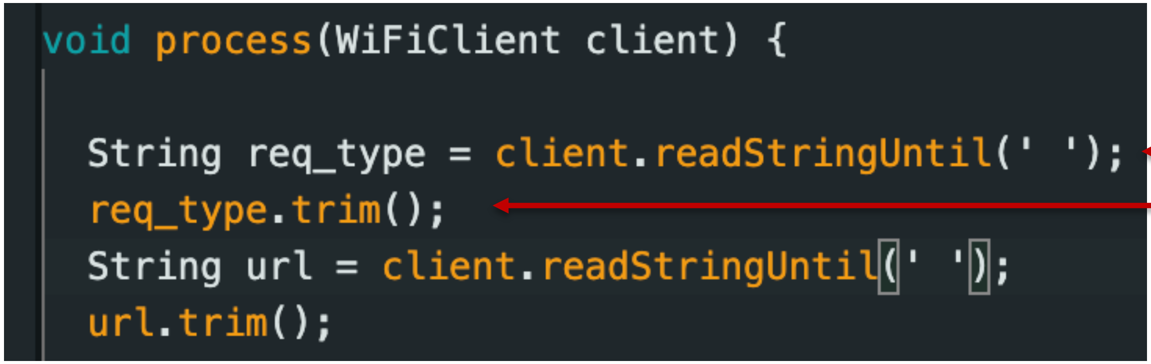
```
void loop() {
  WiFiClient client = server.available();
  if (client) {
    process(client);
    client.stop();
  }
  delay(50);
}
```

New Code Elements: Process

- You can read from the client as any other `Stream`, exactly as the `Serial` class
- The HTTP request "start line" tells you the type of request and the URL of the requested resource. Example:

```
GET /led/1
```

- Parse this line:



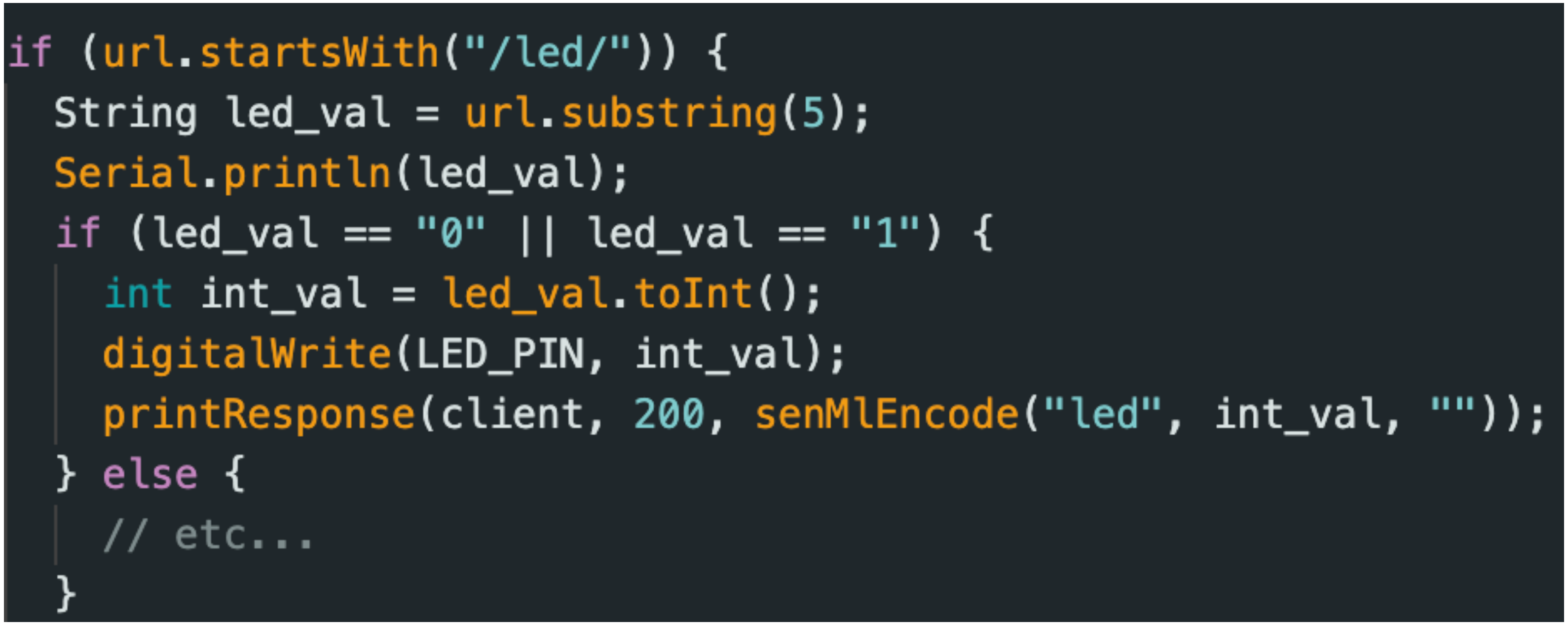
```
void process(WiFiClient client) {

  String req_type = client.readStringUntil(' ');
  req_type.trim();
  String url = client.readStringUntil(' ');
  url.trim();
```

Read until next space

Delete \n or similar

New Code Elements: Process

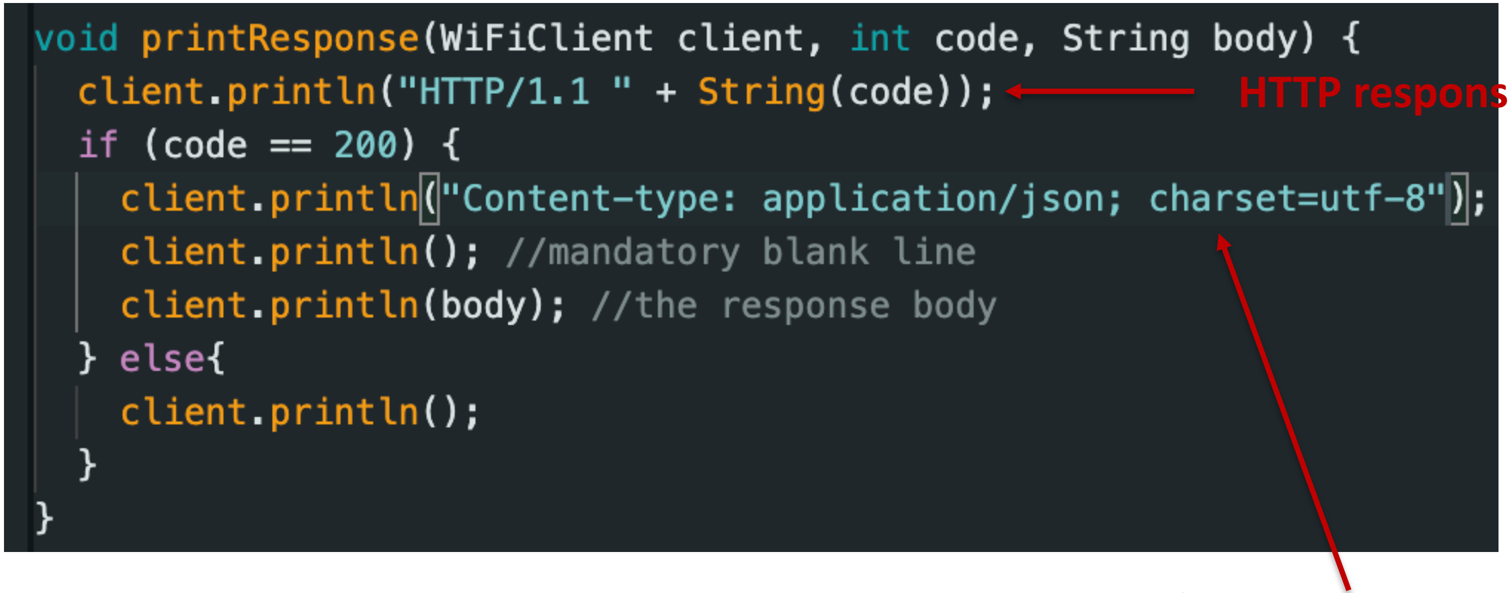
- Respond based on the request type and URL:
  - **Note:** You should implement more robust checks than these…



```
if (url.startsWith("/led/")) {
  String led_val = url.substring(5);
  Serial.println(led_val);
  if (led_val == "0" || led_val == "1") {
    int int_val = led_val.toInt();
    digitalWrite(LED_PIN, int_val);
    printResponse(client, 200, senMlEncode("led", int_val, ""));
  } else {
    // etc...
  }
```

New Code Elements: printResponse

- In the `printResponse()` function, print the header and body of the HTTP response



```
void printResponse(WiFiClient client, int code, String body) {
  client.println("HTTP/1.1 " + String(code));
  if (code == 200) {
    client.println("Content-type: application/json; charset=utf-8");
    client.println(); //mandatory blank line
    client.println(body); //the response body
  } else{
    client.println();
  }
}
```

HTTP response start line

Here the `Content-type` header helps having a better formatting in the browser (in the next exercise it will be fundamental)

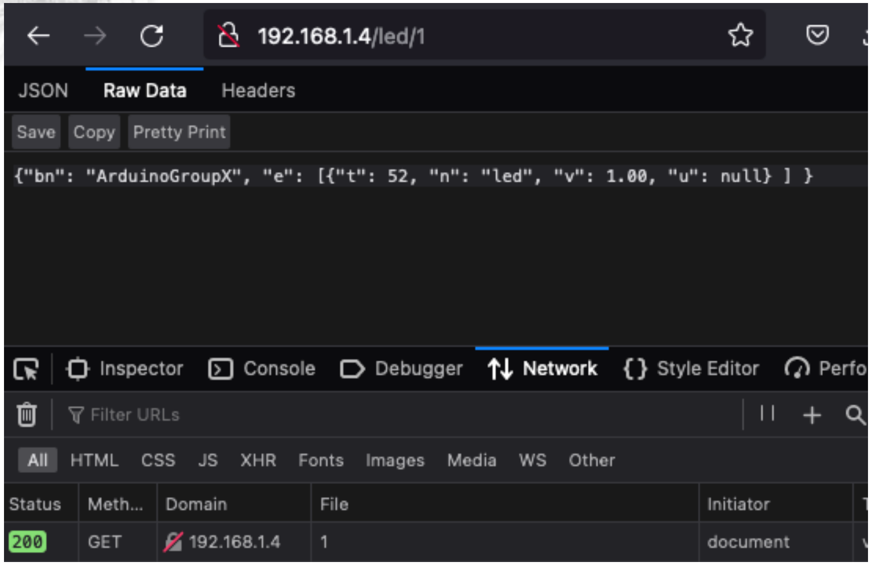
New Code Elements

- The body must be encoded in SenML JSON format (job of the `senMlEncode` function in my example):

```
{
  “bn”: “ArduinoGroupX”
  “e”: [
     {
     “n”: <”temperature”>/<”led”>,
     “t”: <timestamp using millis()>,
     “v”: value,
     “u”: “Cel”/null
     }
  ]
}
```

- You can do this “by hand” (it’s just a concatenation of strings):
  - In Exercise 3, we’ll see how to do more flexible encoding and (most importantly) decoding with the `ArduinoJson` library.

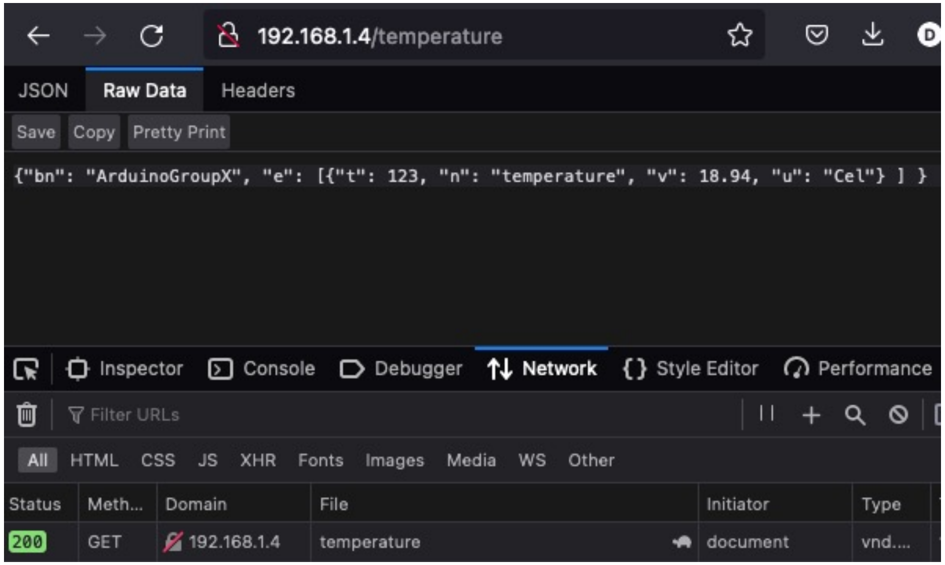
Results



192.168.1.4/led/1

```
{"bn": "ArduinoGroupX", "e": [{"t": 52, "n": "led", "v": 1.00, "u": null} ] }
```

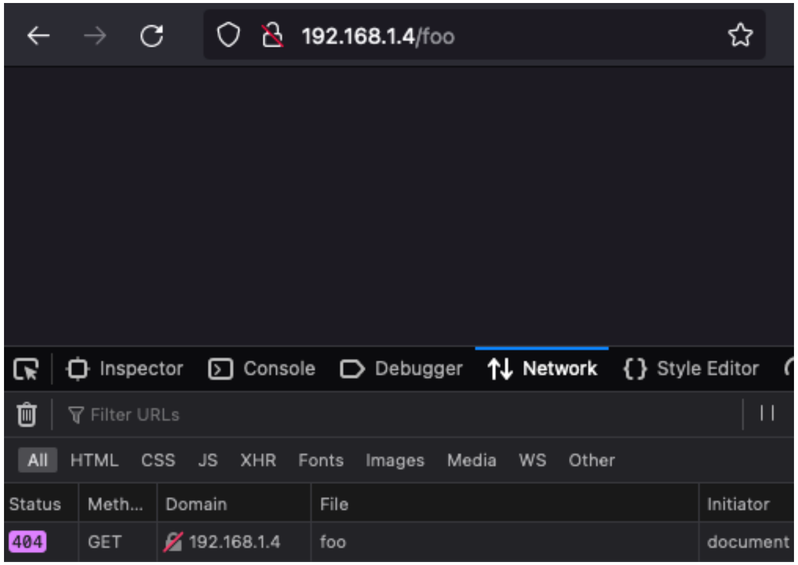
Status	Meth...	Domain	File	Initiator
200	GET	192.168.1.4	1	document



192.168.1.4/temperature

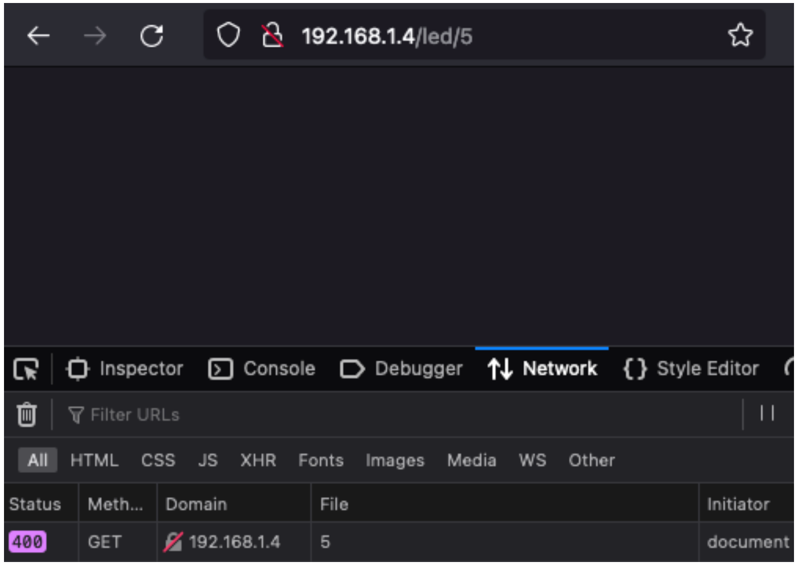
```
{"bn": "ArduinoGroupX", "e": [{"t": 123, "n": "temperature", "v": 18.94, "u": "Cel"} ] }
```

Status	Meth...	Domain	File	Initiator	Type
200	GET	192.168.1.4	temperature	document	vnd....



192.168.1.4/foo

Status	Meth...	Domain	File	Initiator
404	GET	192.168.1.4	foo	document



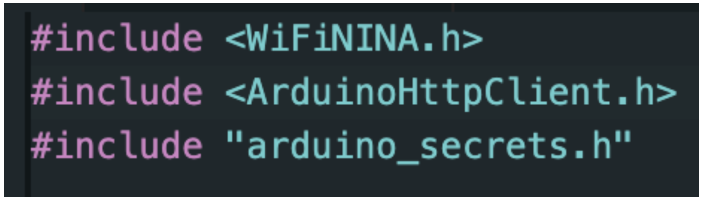
192.168.1.4/led/5

Status	Meth...	Domain	File	Initiator
400	GET	192.168.1.4	5	document

PART3: EXERCISE 2

Arduino HTTP Client

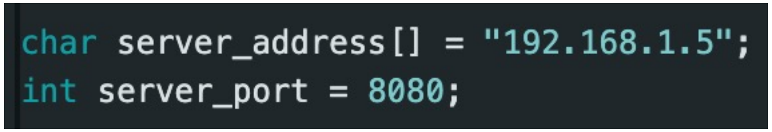
- Include libraries:



```
#include <WiFiNINA.h>
#include <ArduinoHttpClient.h>
#include "arduino_secrets.h"
```

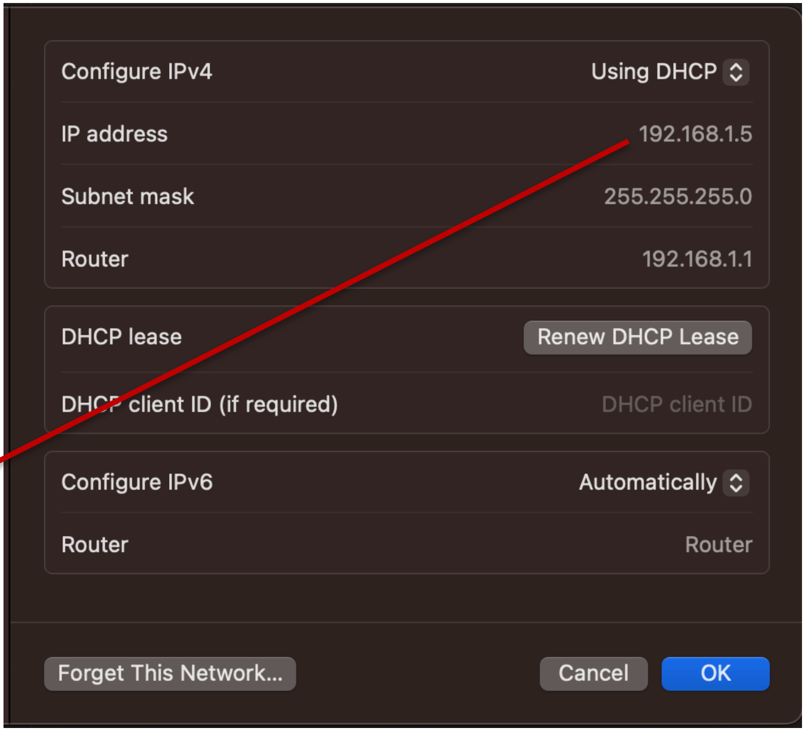
  - `ArduinoHTTPClient` reference (link)

- Define the server address and port:



```
char server_address[] = "192.168.1.5";
int server_port = 8080;
```

- How to get your computer IP depends on the OS:



Configure IPv4: Using DHCP

IP address: 192.168.1.5

Subnet mask: 255.255.255.0

Router: 192.168.1.1

DHCP lease: Renew DHCP Lease

DHCP client ID (if required): DHCP client ID

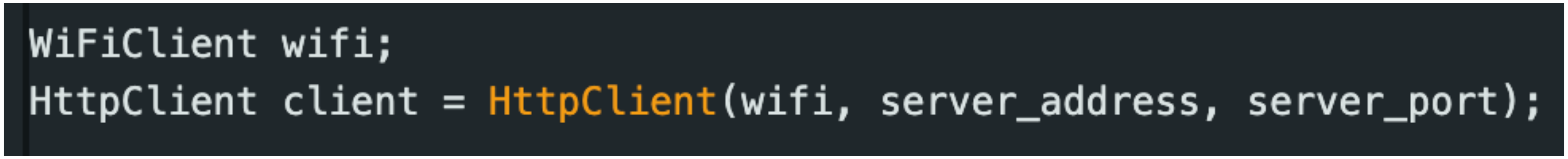
Configure IPv6: Automatically

Router: Router

Arduino HTTP Client

- Declare a `WiFiClient` instance directly
  - Not obtained from a server as in the previous exercise

- Declare a `HttpClient` instance, passing the `WiFiClient`, IP address and port
  - HttpClient facilitates the creation of HTTP requests
  - …so that you don’t have to deal with the protocol at low level (managing basic strings)

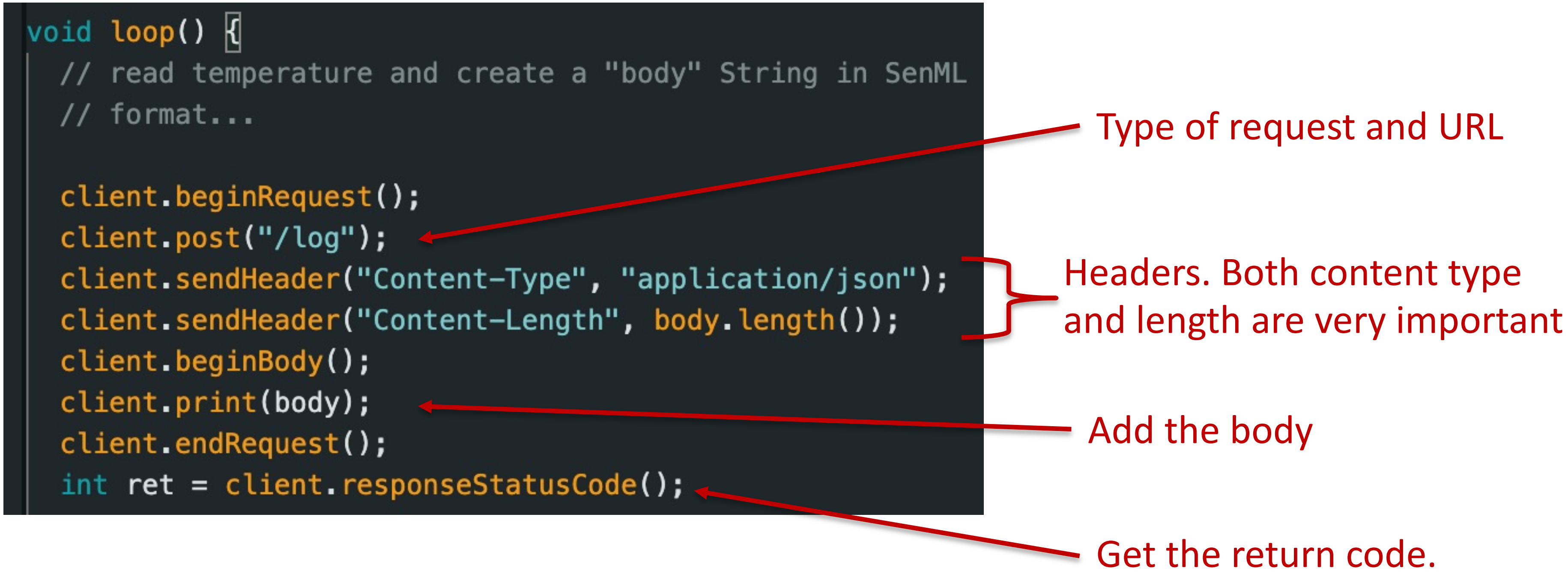


```
WiFiClient wifi;
HttpClient client = HttpClient(wifi, server_address, server_port);
```

Arduino HTTP Client

- The setup() function is identical to the previous exercise.

- In the loop():



```
void loop() {
  // read temperature and create a "body" String in SenML
  // format...

  client.beginRequest();
  client.post("/log");
  client.sendHeader("Content-Type", "application/json");
  client.sendHeader("Content-Length", body.length());
  client.beginBody();
  client.print(body);
  client.endRequest();
  int ret = client.responseStatusCode();
```

Type of request and URL

Headers. Both content type and length are very important

Add the body

Get the return code.

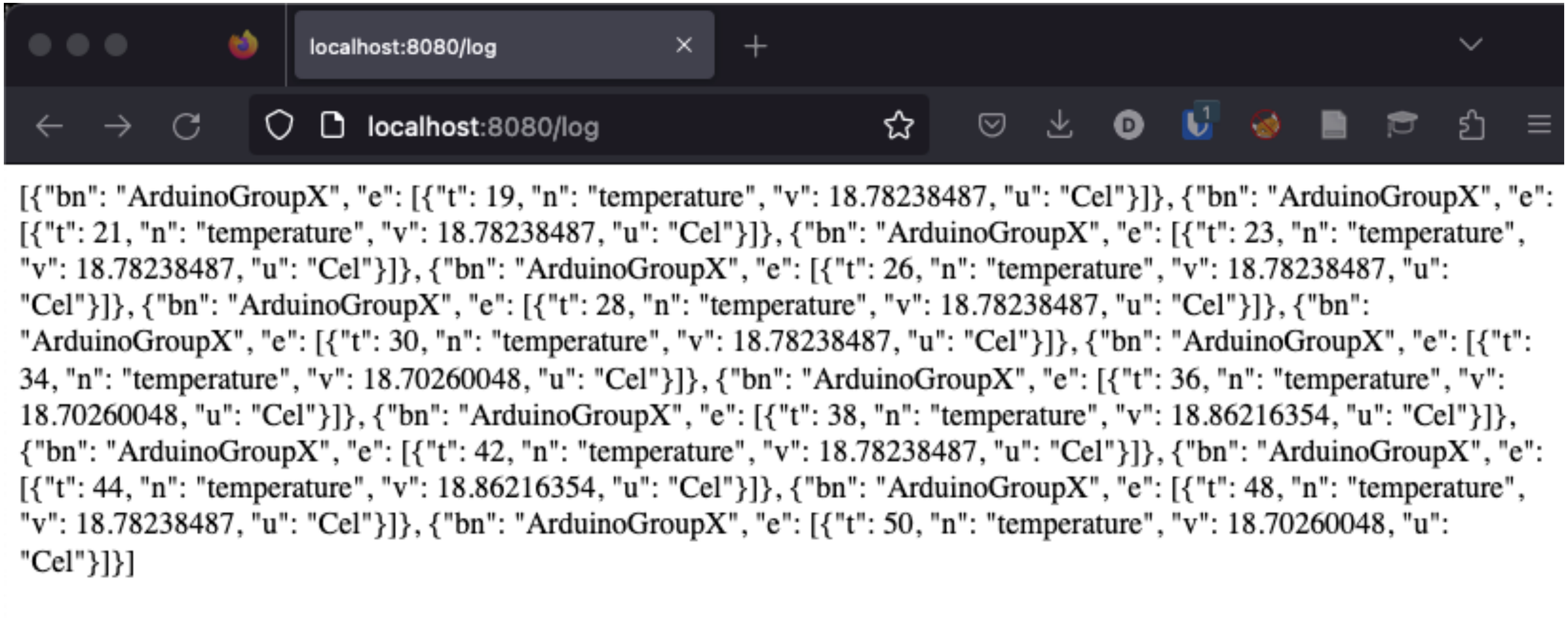
Python HTTP Server

- You also have to modify the cherrypy servers developed in Exercises 1 and 2 of the SW lab to handle both POST and GET requests to the resource:

  `http://<PC IP Address>:<port>/log`

- Nothing special about this, you just need to store all logged data and do some JSON `loads()` and `dumps()` ….

Server GET Result

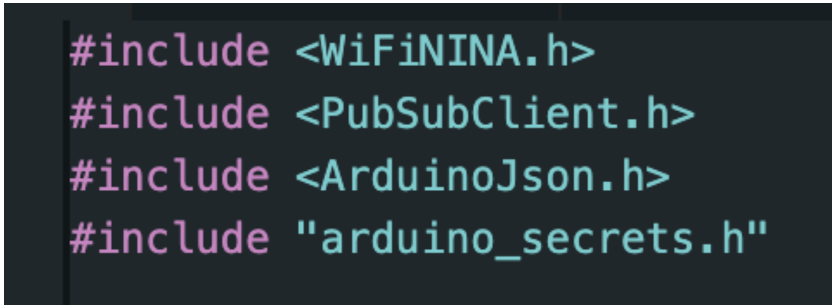


```
[{"bn": "ArduinoGroupX", "e": [{"t": 19, "n": "temperature", "v": 18.78238487, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e":
[{"t": 21, "n": "temperature", "v": 18.78238487, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t": 23, "n": "temperature",
"v": 18.78238487, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t": 26, "n": "temperature", "v": 18.78238487, "u":
"Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t": 28, "n": "temperature", "v": 18.78238487, "u": "Cel"}]}, {"bn":
"ArduinoGroupX", "e": [{"t": 30, "n": "temperature", "v": 18.78238487, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t":
34, "n": "temperature", "v": 18.70260048, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t": 36, "n": "temperature", "v":
18.70260048, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t": 38, "n": "temperature", "v": 18.86216354, "u": "Cel"}]},
{"bn": "ArduinoGroupX", "e": [{"t": 42, "n": "temperature", "v": 18.78238487, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e":
[{"t": 44, "n": "temperature", "v": 18.86216354, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t": 48, "n": "temperature",
"v": 18.78238487, "u": "Cel"}]}, {"bn": "ArduinoGroupX", "e": [{"t": 50, "n": "temperature", "v": 18.70260048, "u":
"Cel"}]}]
```

PART3: EXERCISE 3

Arduino MQTT: Global

- Include required libraries. Two new ones are introduced in this exercise:



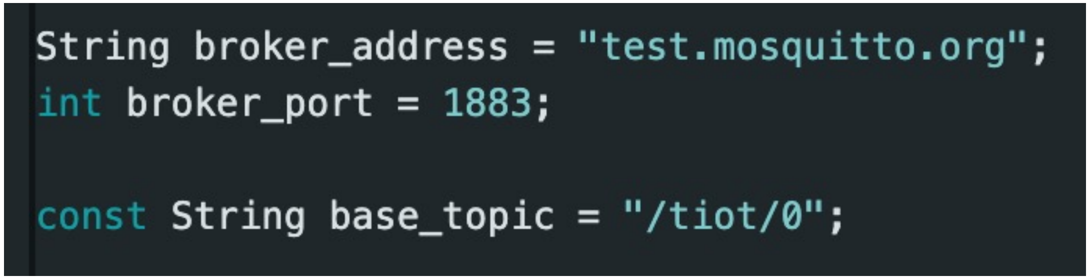
```
#include <WiFiNINA.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include "arduino_secrets.h"
```

- `PubSubClient` implements MQTT publish and subscribe functionalities
  - Reference (link)

- `ArduinoJson` helps you read/write JSON strings:
  - Useful because this time we need to parse a JSON (more difficult) besides generating one. Reference (link)

Arduino MQTT: Global

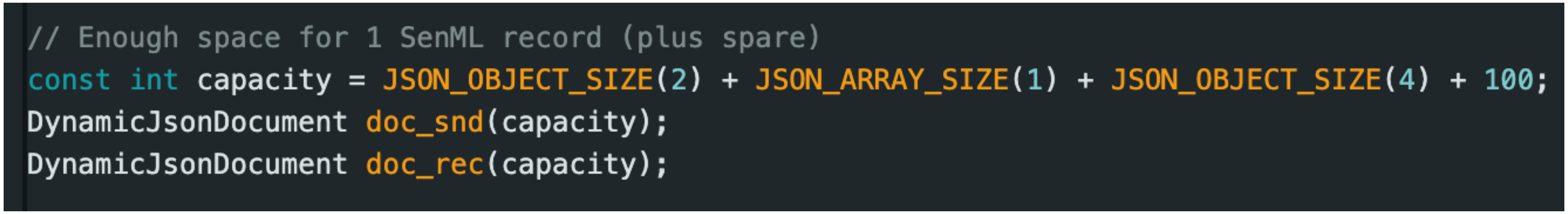
- Declare the URL (or IP address) of the broker and the prefix of the topic:



```
String broker_address = "test.mosquitto.org";
int broker_port = 1883;

const String base_topic = "/tiot/0";
```

Arduino MQTT: Global

- Create two global objects to store the sent and received JSON strings:

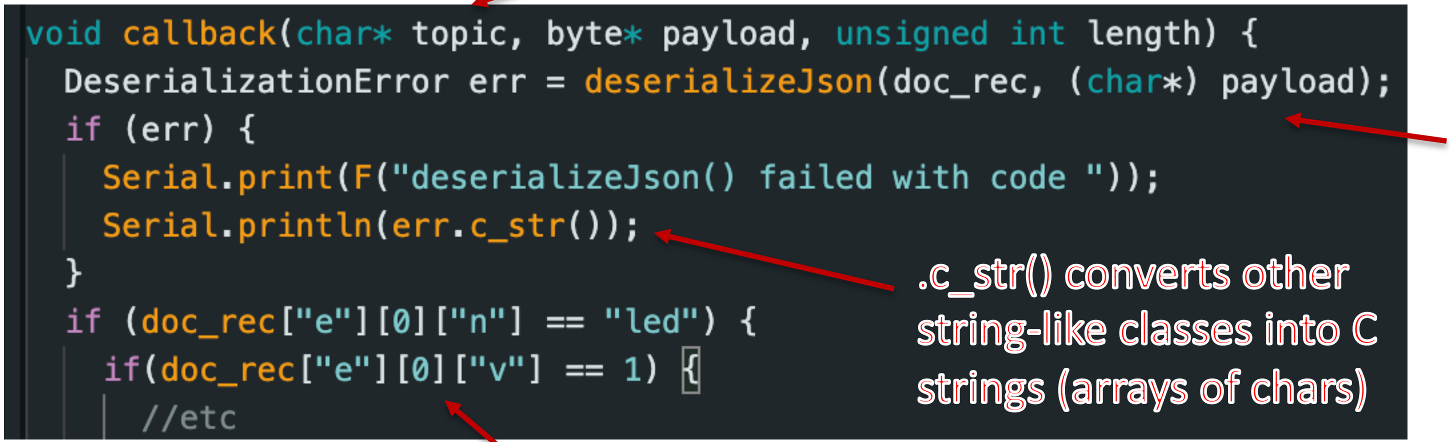


```
// Enough space for 1 SenML record (plus spare)
const int capacity = JSON_OBJECT_SIZE(2) + JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(4) + 100;
DynamicJsonDocument doc_snd(capacity);
DynamicJsonDocument doc_rec(capacity);
```

Arduino MQTT: Callback

- Define a callback to handle messages arriving on subscribed topics:
  - ArduinoJson helps you transform the received string (array of bytes) into an easily-accessible object:

”topic” can be used to share a single callback for multiple topics



```
void callback(char* topic, byte* payload, unsigned int length) {
  DeserializationError err = deserializeJson(doc_rec, (char*) payload);
  if (err) {
    Serial.print(F("deserializeJson() failed with code "));
    Serial.println(err.c_str());
  }
  if (doc_rec["e"][0]["n"] == "led") {
    if(doc_rec["e"][0]["v"] == 1) {
      //etc
```

.c_str() converts other string-like classes into C strings (arrays of chars)

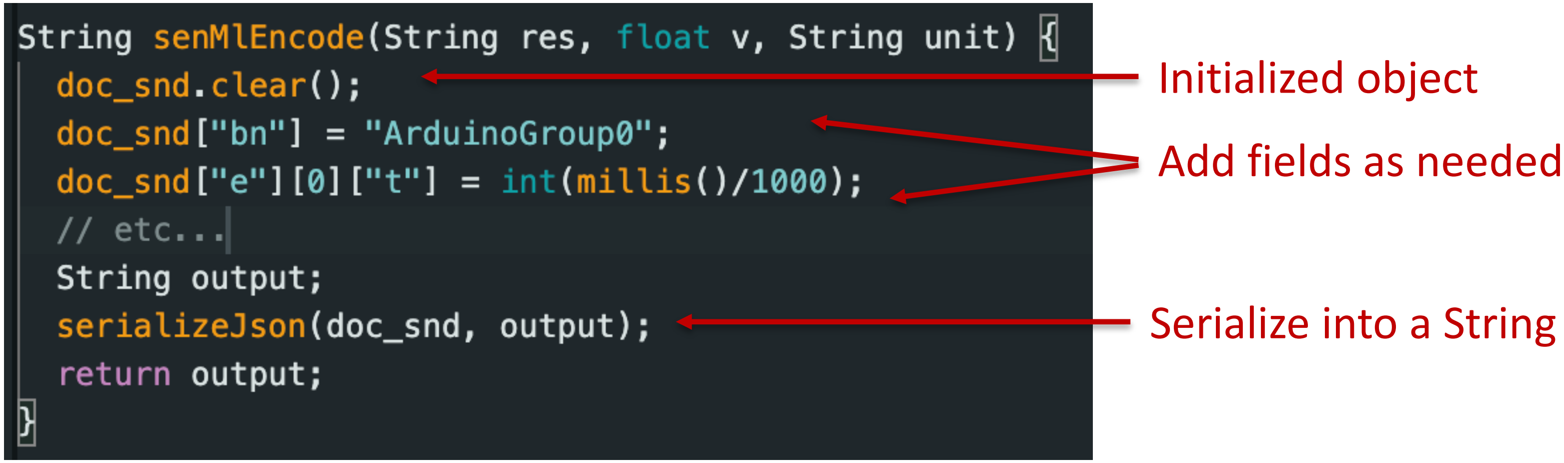
Cast byte array into char array (C string) Then de-serialize it, i.e., read it into the `doc_rec` object.

Access the object’s fields by name (key) or by index, corresponding to the received JSON format (SenML in our case).

ArduinoJson Details

- With `ArduinoJson` you can also check if:
  - The JSON was correclty parsed: `.isNull()` method
  - A given field is present in the received object: `.containsKey()` method

- Moreover, you can also use it as a better alternative to manual String concatenation to generate JSON strings (aka “serialize” a JSON object):



```
String senMlEncode(String res, float v, String unit) {
  doc_snd.clear();
  doc_snd["bn"] = "ArduinoGroup0";
  doc_snd["e"][0]["t"] = int(millis()/1000);
  // etc...
  String output;
  serializeJson(doc_snd, output);
  return output;
}
```

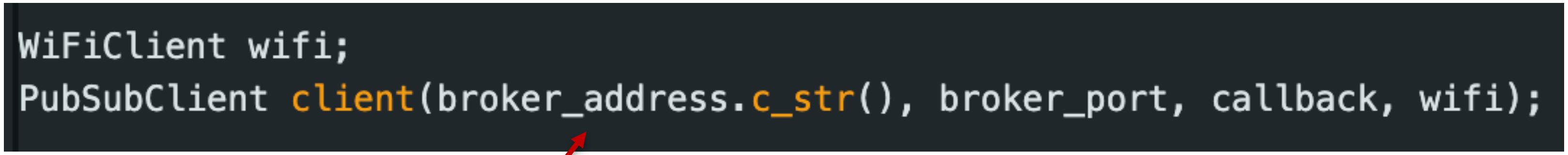
Initialized object

Add fields as needed

Serialize into a String

Arduino MQTT: Global

- In the global section, declare a `PubSubClient` object:
  - This must be done after defining the callback!
  - Similarly to `ArduinoHTTPClient`, this object uses a `WiFiClient` internally to transmit/receive messages
  - Other parameters are the broker's address and port, and the callback function just defined to handle incoming messages

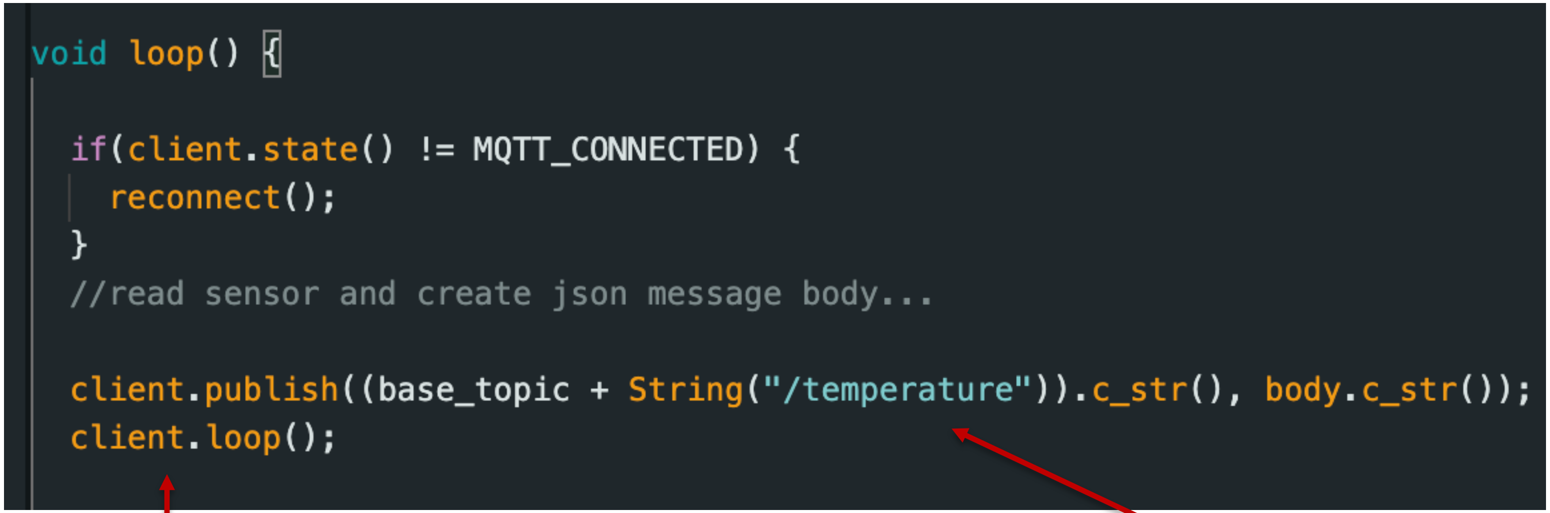


```
WiFiClient wifi;
PubSubClient client(broker_address.c_str(), broker_port, callback, wifi);
```

Differently from ArduinoHTTPClient, the address must be a C char array.

Arduino MQTT: Loop

- The setup() is identical to the previous 2 exercises
- In the loop():



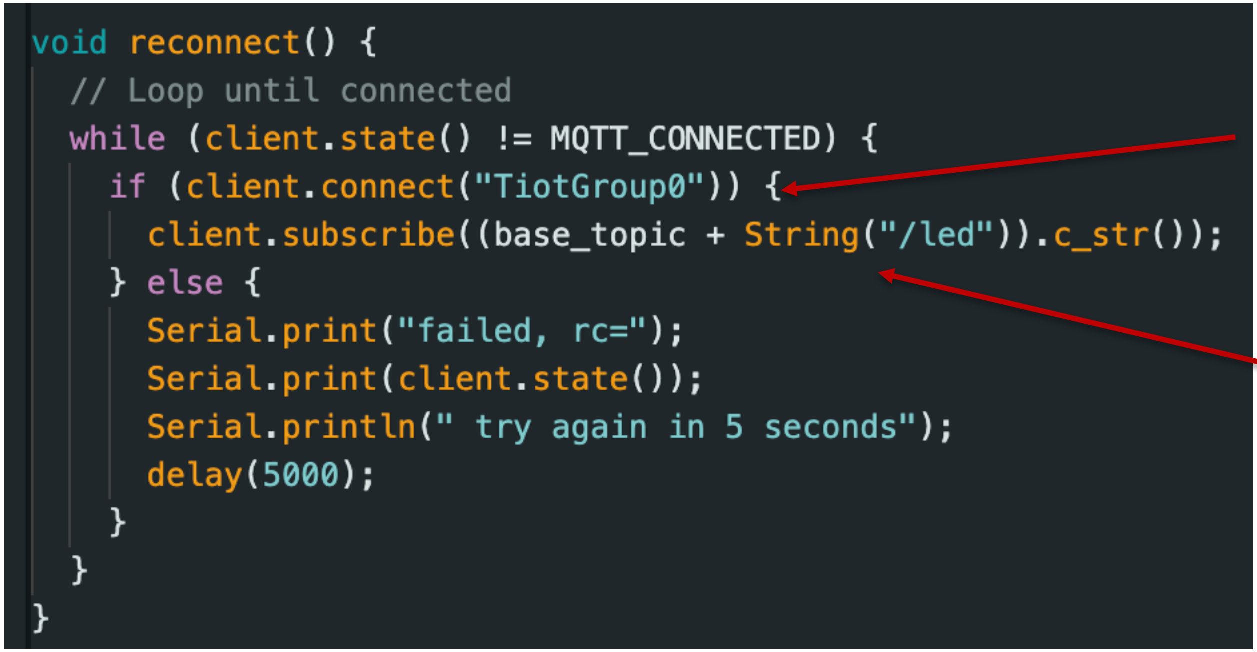
```
void loop() {

  if(client.state() != MQTT_CONNECTED) {
    reconnect();
  }
  //read sensor and create json message body...

  client.publish((base_topic + String("/temperature")).c_str(), body.c_str());
  client.loop();
```

Publish a message: `publish(topic, body)`

Check if there are new messages
on subscribed topics

Arduino MQTT: Reconnect

- Function to connect (or re-connect) to the broker and subscribe (re-subscribe) to topics. Called by the loop():



```
void reconnect() {
  // Loop until connected
  while (client.state() != MQTT_CONNECTED) {
    if (client.connect("TiotGroup0")) {
      client.subscribe((base_topic + String("/led")).c_str());
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

Connection requires a unique ClientID. Use your group number

`.subscribe(topic)`

Testing Exercise 3.3

- To test the exercise, you’ll need to install the Mosquitto client (or another MQTT client) on your laptop (instructions can be found here)

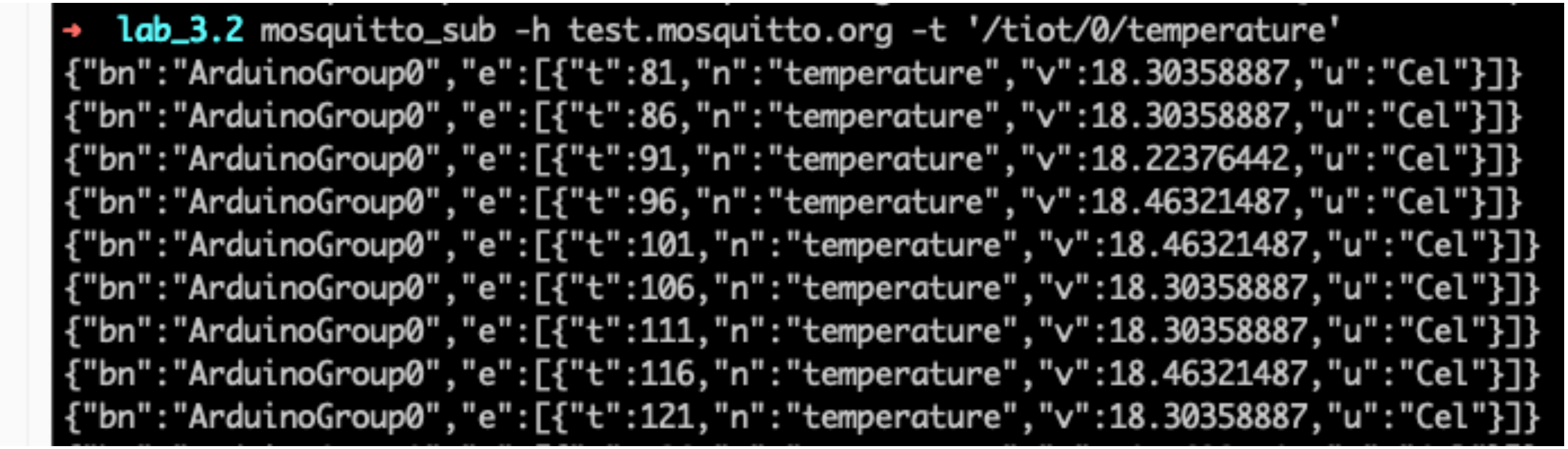
- Subscribe command on your terminal (to read temperature logs):

```
mosquitto_sub -h test.mosquitto.org -t '/tiot/group0/temperature’
```

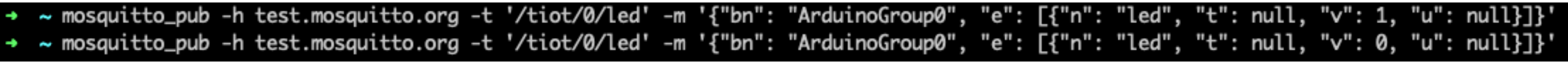
- Publish command on your terminal (to turn ON the LED):

```
mosquitto_pub -h test.mosquitto.org -t '/tiot/group0/led' -m
'{"bn": "Yun", "e": [{"n": "led", "t": null, "v": 1, "u":
null}]}’
```

Final Result



```
➜  lab_3.2 mosquitto_sub -h test.mosquitto.org -t '/tiot/0/temperature'
{"bn":"ArduinoGroup0","e":[{"t":81,"n":"temperature","v":18.30358887,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":86,"n":"temperature","v":18.30358887,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":91,"n":"temperature","v":18.22376442,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":96,"n":"temperature","v":18.46321487,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":101,"n":"temperature","v":18.46321487,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":106,"n":"temperature","v":18.30358887,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":111,"n":"temperature","v":18.30358887,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":116,"n":"temperature","v":18.46321487,"u":"Cel"}]}
{"bn":"ArduinoGroup0","e":[{"t":121,"n":"temperature","v":18.30358887,"u":"Cel"}]}
```



```
➜  ~ mosquitto_pub -h test.mosquitto.org -t '/tiot/0/led' -m '{"bn": "ArduinoGroup0", "e": [{"n": "led", "t": null, "v": 1, "u": null}]}'
➜  ~ mosquitto_pub -h test.mosquitto.org -t '/tiot/0/led' -m '{"bn": "ArduinoGroup0", "e": [{"n": "led", "t": null, "v": 0, "u": null}]}'
```