Table of Contents

1. InT Networks: Laboratory 2 - MQTT 2
1.1. Hardware 2
1.1.1. Hardware-software components and libraries 2
1.2. Software 4
1.2.1. Software libraries 4
1.3. Networking 6
1.4. Network services 6
1.4.1. MQTT 6
1.4.2. Node Red and Node Red Dashboard 7
1.5. Laboratory scenarios 7
1.5.1. Scenario 1 8
1.5.2. Scenario 2 9
1.5.3. Scenario 3 10

1. InT Networks: Laboratory 2 - MQTT

Network programming for the IoT Devices using MQTT protocol.

Following manual assumes you're familiar with InT Networks: Laboratory 1 - Introduction. In case you're not, please get yourself familiar with it and spend some time practising. You can start your project stub using hints and guidance in the InT Networks: Laboratory 1 - Introduction manual and modify it / extend as guided below.



This manual is not intended to be stand alone. You need to follow Laboratory 1 manual to prepare basics for the project then continue here.



Computers in laboratory rooms 320, 321 and 336 use freezing software which means that for every startup, computers are restored to their initial state.

1.1. Hardware

You will use a sensor that measures temperature + humidity, to publish this information to the MQTT broker:

- DHT 11 temperature and humidity sensor



Fig. 1: DHT 11 temperature and humidity sensors

1.1.1. Hardware-software components and libraries

Interfacing hardware requires the use of particular GPIOs. Laboratory nodes and hardware used needs specific configuration. Following chunks of code will help to set up proper hardware-software configuration.

1.1.1.1. LCD display

Built-in LED on the microcontroller development board can help to manage simple tracing of the code execution:

```
#define LED_BUILTIN 14
```

Your LCD shield is a 2×16 character one, driven by a Hitachi HD44780 controller. You can use a library to use it easily. The PlatformIO will pull the library automatically if defined in *platformio.ini* file:

```
lib_deps = arduino-libraries/LiquidCrystal
```

```
#include <LiquidCrystal.h>
```

To initialise it and instantiate controller object, use following GPIO definition and initialisation:

```
const int rs = 12, en = 13, d4 = 4, d5 = 0, d6 = 2, d7 = 14;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

It is also necessary to tell the library about the display organisation as there is a variety of configurations possible. Here, the organisation is 2 lines, 16 columns each, so in the code you need to explicitly initialise it:

```
...
lcd.begin(16,2);
...
```

1.1.1.2. Temperature and humidity sensor

There is a DHT11 (blue) sensor connected to the GPIO 5.
There are two libraries needed to handle reading from the sensor: a common sensor communication library and a specific for DHT sensors, both by Adafruit, so two more entries are needed in the *platformio.ini* file, and obviously you need to refer to it in the source code:

```
lib_deps =
    adafruit/Adafruit Unified Sensor@^1.1.4
    adafruit/DHT sensor library@^1.4.3
```

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

To access sensor readings it is necessary to instantiate a communication class that binds the hardware to the code:

```
#define DHTTYPE DHT11   // DHT 11
#define DHTPIN 5
DHT dht(DHTPIN, DHTTYPE,50);
```

It is also necessary to initialise the sensor class and give it some time to set up:

```
...
dht.begin();
delay(1000);
...
```

The sensor class returns data (among others) in a float containing temperature (in C) and relative humidity (in per cent):

```
...
  float hum = dht.readHumidity();
  float temp = dht.readTemperature();
...
```

1.2. Software

String formatting may be a challenge. An easy approach is to allocate a character buffer of the size of the LCD or just one display's line, or any other size, large enough to handle your text.

```
...
char buffer [50];
...
```

To convert float into the string with formatting use `sprintf` function:

```
...
  sprintf(buffer,"%2.1f C",temp);
...
  sprintf(buffer,"%2.1f%%Rh",hum);
...
```



`sprintf` uses number of wildcards that are rendered with data. Refer to the c/c++ documentation on `sprintf`. Here

```
%2.1f
```

means: having float number render it to string using two digits before decimal point and one after. Temperature in our lab is always above 10C.

```
%%
```

is an escape character to print a

```
%
```

(percent) symbol.

```
delay(time)
```

is measured in milliseconds.

1.2.1. Software libraries

Software libraries are needed to handle communication between your IoT device and network and

services. In this laboratory, you will connect your device to the WiFi access point (see the Networking section below for credentials) and use TCP. Your device will be automatically given an IP address that you may show on the LCD screen. Once WiFi is connected and works (PHY+MAC+NET layer protocols) then you can connect over it to the MQTT Broker (APP layer protocol).
Mind, the connection may become broken so ensure re-connection mechanism:

- dummy: in case the connection is broken, restart the device to let it perform the connection process again;
- smart: include checking and re-connection into your code using connection status tracking, mind you not only need to reconnect WiFi but also high-level communication protocol: MQTT.

1.2.1.1. WIFI and TCP

A WiFi library is already included in the ESP8266 framework for the Arduino programming model, so there is no need to use library manager to add it. All is necessary is to include a header to your C code:

```
...
#include <ESP8266WiFi.h>
...
```

Then create some constant variables with `sSsid` and `sPassword` (see Networking section for credentials) and instantiate WiFi communication object:

```
...
#define sSsid "<here comes ssid to connect to>"
#define sPassword "<here comes a password>"
...
```

The `WiFi` object is a singleton, static one. Try to connect to the AP, i.e. this way:

```
...
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
...
```

Some useful code snippets are below.
Checking if WiFi connection is successful:

```
...
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
...
```

Checking the current IP address:

```
  Serial.println(WiFi.localIP());
```

Once WiFi is connected, it is necessary to use a `WiFiClient` object to pass the connection object to the application layer libraries.
The declaration can be as follows:

```
== MQTT ==
To ease and speed up implementation, you will use an MQTT library that
implements acting both as a publisher and subscriber ''platformio.ini''
file:
<code>
lib_deps = ..., PubSubClient, ...
```



Please note, on the NET protocol level, in this scenario, you're connecting to the WiFi AP (STA), not setting your device to be a WiFi server (AP), so you need to set appropriate WiFi mode, as it may be previously used in AP, not STA mode.

1.3. Networking

There is a WiFi 2.4GHz, IPv4 network available to the students. This is a PRIVATE network. This network is separated from the Internet but all connected devices to it can talk to another. There is a DHCP server with networking addressing scope 192.168.90.10-192.168.90.254.



Do not use static IP! Some services present in the network have fixed IP addresses, overlapping it causes unavailability of your solution and this service, also to other teams! Use the only DHCP for obtaining a network address.

To connect to the wireless network, use the following credentials:

- SSID: internal.IOT
- Passkey: IoTlab32768
- IP address: DHCP, 192.168.90.10-192.168.90.254

1.4. Network services

1.4.1. MQTT

There is an MQTT Broker, binding WiFi PRIVATE network and the PUBLIC Internet. This way you can route MQTT messages from your end node devices connected to the `internal.IOT` network to the external tools i.e. cloud-based dashboard or storage via i.e. publishing from your end node IoT device through the PRIVATE network to MQTT Broker and subscribing to those messages from the cloud using PUBLIC Internet.

You can access it from the Internet PUBLIC network, using the following configuration and credentials:

- Address: 157.158.56.54
- Port: 1883 for plain text, 8883 for secure communication
- User: vrel
- Password: vrel2018

You can also access it from the aforementioned PRIVATE network, using the following configuration and credentials:

- Address: 192.168.90.5
- Port: 1883 for plain text, 8883 for secure communication
- User: vrel
- Password: vrel2018

1.4.2. Node Red and Node Red Dashboard

A Node RED server is working on same Raspberry Pi 2 as MQTT and in private address pool only, available through *internal.IoT* on 192.168.90.5 address, port 1880:

- http://192.168.90.5:1880 - main editor
- http://192.168.90.5:1880/ui - Node RED dashboard

Login using the following credentials:

- User: *contact supervisor*
- Password: *contact supervisor*

1.5. Laboratory scenarios

All teams share the same scenario yet please note that in particular, MQTT messages exchanged between devices shall be coordinated among team members not to overlay one another.



Mind keeping your code between meetings. It is regular that every next laboratory meeting you will extend your code by something extra.

1.5.1. Scenario 1

Getting ready with your development environment
In this scenario, you will write your first “Hello World” application for ESP8266. Using the selected IDE, write code that:

1. Flashes built-in LED every 1 second (1s on; 1s off);
2. Sends a message via serial port to the serial port monitor: `Hello World`



The GPIO the built-in LED is connected to is number 14. Physically it is located almost in the middle of the mainboard, under the display and its colour is blue.

```
...
#define LED_BUILTIN 14
...
```

The standard serial port speed is 9600 (bps). Connect COM port monitor (a plug icon in the PlatformIO toolbar), once you compile and upload your code, to see the “Hello World” message.

```
...
Serial.begin(9600);
...
```

Advanced:

- Use timer/interrupt to flash the LED instead of an infinite loop.

Report:

- Describe your approach and include source code in the report

1.5.2. Scenario 2

** Using LCD **
In this scenario, you will write an application that writes on the LCD Display.
Your code shall implement a message `Hello IoT` in the upper line, and a counter ticking every second in the lower one.

Report:

- Present algorithm.
- Include source code.

1.5.3. Scenario 3

** Using Keypad **
Extend Scenario 2 with a counter that increments on the keypress of the Keypad Shield (any but SELECT!).

Report:

- Describe how did you solve the problem of the asynchronous processing of the keypresses/releases.
- Include source code.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:iotn:lab2?rev=1647554388

Last update: **2022/03/17 22:59**