EESTech Challenge 2019 - Tasks 1-2

# Hardware

Your team got the following hardware to work with:

* [Wemos D1 mini](https://wiki.wemos.cc/products:d1:d1_mini) development board
* NeoPixel (WS2812B) ring with 24 individually addressable LEDs.
* DHT11 temperature and humidity sensor
* 3\*4 matrix keypad like<https://www.adafruit.com/product/419>
* Photoresistor (GL5528)
* Potentiometer (10K)
* push buttons
* breadboards, wires, USB cable

For details see: <https://sites.google.com/corp/view/eestech-2019-zurich/preparation>

# Authentication, Credentials

Your team has:

* a **name** (you signed up with)
* a **username** (or team\_id, assigned to you)
* a **password** you use to connect to various services.

Usernames are strings in the format of "teamX", where "team" is literal and "X" stands for a number. For example "team1" or "team2".

# Services

The following services are set up for the competition. It's advised to test these connections from your laptop first, then use them from the devices.

## WiFi for the IoT devices, laptops and phones

A WiFi network is provided to be used for the IoT devices, laptops, phones, etc.

* All WiFi APs are connected to the same local network, the laptops and IoT devices are on the same broadcast medium.
* Please do not abuse the network, do not flood it, don't do DoS, don't interfere with other teams..

|  |  |
| --- | --- |
| **IoT WiFi** | |
| SSID | EESTech |
| Password (PSK) | Challenge2019Accepted |
| Security | WPA2/AES |

## MQTT Broker

A single shared MQTT broker is set up for all the tasks and some other communication. You'll have to use your team's id and MQTT-password to connect to the broker.

Note that you can connect from your laptop or phone too.

|  |  |
| --- | --- |
| **MQTT broker** | |
| Host and port | vm1.siroki.cc:8883 |
| Security | TLS, LetsEncrypt Certificate |
| Username | <teamX> (= your team *username*) |
| Password | <YYYYYY>(your team's *password*) |

### URL

Note that you can use URLs of the following format to connect:

mqtts://teamX:YYYYYY@vm1.siroki.cc:8883

### Topic ACLs

* /test/# - read/write (all teams can use these)
* /teams/teamX/# - read/write (your team only)

Publish test message using Mosquitto to the topic "/test/foo":

$ mosquitto\_pub -d --capath /etc/ssl/certs/ -L mqtts://teamX:YYYYYY@vm1.siroki.cc:8883//test/foo -m hello

Subscribe to a topic ("/test/foo") using Mosquitto:

$ mosquitto\_sub -d -v --capath /etc/ssl/certs/ -L mqtts://teamX:YYYYYY@vm1.siroki.cc:8883//test/foo

The commands above have verbose and debug output, you can remove -d or -v if you want.

## InfluxDB server

There is a shared InfluxDB instance set up for the challenge.

|  |  |
| --- | --- |
| **InfluxDB** | |
| Host and port | vm1.siroki.cc:8086 |
| Security | TLS, LetsEncrypt Certificate |
| Username | <teamX> (= your team *username*) |
| Password | <YYYYYY>(your team's *password*) |
| Database name | <teamXdb> (the only database your team can use) |

You can freely create different "measurement"s (like database tables) in your database.

Connect using the influx command line tool:

$ influx -host vm1.siroki.cc -port 8086 -ssl -username 'teamX' -password 'YYYYYY'

Connected to https://vm1.siroki.cc:8086 version 1.7.4

InfluxDB shell version: 1.7.4

Enter an InfluxQL query

> show databases

name: databases

name

----

teamXdb

> use teamXdb

Using database teamXdb

> show measurements

See <https://docs.influxdata.com/influxdb/v1.7/tools/shell/> for more details about how to use influx.

Example insert, select, delete:

> INSERT treasures,captain\_id=pirate\_king value=2

> SELECT \* FROM treasures

name: treasures

time captain\_id value

---- ---------- -----

1553978854664959911 pirate\_king 2

> INSERT treasures,captain\_id=pirate\_fool value=1

> SELECT \* FROM treasures

name: treasures

time captain\_id value

---- ---------- -----

1553978854664959911 pirate\_king 2

1553978984064921686 pirate\_fool 1

> DELETE FROM treasures WHERE captain\_id=pirate\_fool

> SELECT \* FROM treasures

name: treasures

time captain\_id value

---- ---------- -----

1553978854664959911 pirate\_king 2

> DELETE FROM treasures

> SELECT \* FROM treasures

Tips:

Use -precision=rfc3339 to show human-readable timestamps.

## Leaderboard

The leaderboard can be accessed under the following URL:

|  |  |
| --- | --- |
| **Leaderboard** | |
| URL | <https://vm1.siroki.cc:3000/d/eNUsZl6mz/leaderboard?orgId=1&theme=light> |
| Username | contestant |
| Password | Challenge2019Accepted |

# Software

## MQTT

See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_QYNAB0wwl3as>

## Node-RED

<https://nodered.org/docs/writing-functions.html>

# Tasks and learning material

Here you can read the actual tasks that you'll need to solve, interleaved with learning material that is needed for solving the tasks.

First start with one of the breadboards and an ESP device, then later you'll need to use both and make them interact with each other.

## Learning - about ESP8266

See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_doDj3TC4CMMB>

## Learning - LOLIN (Wemos) D1 mini

See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_ZO8EvGTvFgp4>

## Learning - Arduino

See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_fYcPsAJELAn9>

## Learning - ESP8266 Arduino platform and libraries

See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_KbGqYD15LzA0>

## Task 1.1 - Blink

**Goal: Set up your development environment and try the Blink example.**

**Score: 5**

Select the correct board in Arduino IDE: "LOLIN(WEMOS) D1 R2 & mini"

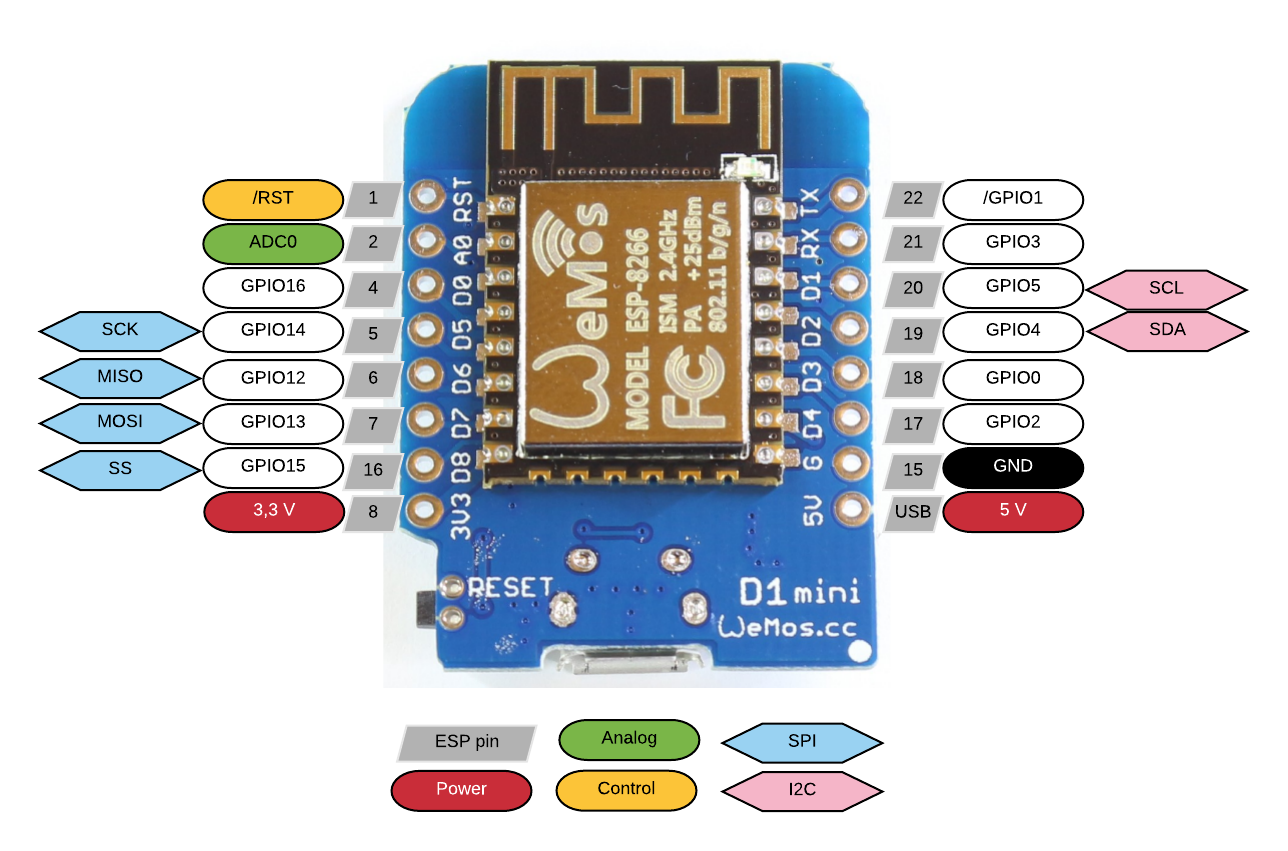
See the d1\_mini board definition here if you're interested in the details:

<https://github.com/esp8266/Arduino/blob/master/boards.txt#L3347>

Use the LED\_BUILTIN macro constant to identify the GPIO that the LED is connected to. It's active low, so digitalWrite(LED\_BUILTIN, LOW) will turn the LED on.

By the way LED\_BUILTIN is GPIO2, which is the same as D4.

You can see the mapping between GPIO pins and D\* markings on the board:



See the relevant pin definitions here:

<https://github.com/esp8266/Arduino/blob/master/variants/d1_mini/pins_arduino.h>

So the following are equivalent:

digitalWrite(2, HIGH); // means GPIO2

digitalWrite(D4, HIGH); // D1-D8 are defined for the specific board in the pins\_arduino.h file.

Use another pin to wire up an **external LED**. Make the **two LEDs blink synchronously at about 4 Hz**.

**Evaluation:**

If you're ready, call for one of the organizers to visually inspect the LEDs.

## Learning - ESP and WiFi global objects

## 

## The ESP global object is provided by the *Arduino core for ESP8266 WiFi chip* library. It provides some functions that return static or runtime information about the chip.

The ESP8266WiFi library provides wide collection of C++ methods (functions) and properties to configure and operate an ESP8266 module in station and / or soft access point mode.

The **WiFi global object** is provided by the *ESP8266WiFi* library and can be used to access WiFi-related functionality.

## Some useful APIs

## From <https://github.com/esp8266/Arduino/blob/master/cores/esp8266/Esp.h>:

## ESP.getChipId(), …

## 

## <https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/ESP8266WiFi.h>

## <https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/ESP8266WiFiSTA.h>

## WiFi.SSID(), WiFi.BSSIDstr(), …

## 

## Learning - Arduino String class

## See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_fG-27X8BT4CB>

## Learning - HTTP

See <https://sites.google.com/corp/view/eestech-2019-zurich/study-material#h.p_MirIoJ8fm6-R>

The WebServer class found in ESP8266WebServer.h header, is a simple web server that knows how to handle HTTP requests such as GET and POST and can only support one simultaneous client.

To learn more:

<https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer>

## Task 2.1 - StaticWebServer

**Goal: Connect to WiFi and serve a static page on HTTP**

**Score: 10**

Read the [ESP8266WiFi library documentation](https://github.com/esp8266/Arduino/blob/master/doc/esp8266wifi/readme.rst) and use the [ESP8266WebServer](https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer) library to set up a Web (HTTP) server on the device.

On http://<device IP>/, serve any response that contains your team name, "teamX".

Try the ESP8266WebServer > [HelloServer example](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WebServer/examples/HelloServer/HelloServer.ino), connect your device to the EESTech WiFi network and serve HTML pages.

Hint: Print out the IP address of the device to the serial console so you can see what to connect to.

Set up a simple web server on port 80 that serves your team's identifier as content at the root path (/).

So if you go to http://<device IP>/, you'll see "teamX" in the browser.

**Evaluation:**

If you're ready, call for one of the organizers to visually inspect the LEDs.

## Learning - HTTP URL format and parameters

Every [HTTP URL](https://en.wikipedia.org/wiki/URL) conforms to the syntax of a generic URI. The URI generic syntax consists of a hierarchical sequence of five components:

URI = scheme:[//authority]path[?query][#fragment]

When the HTTP GET method is used, a set a key-value pairs can be sent as part of the [query string](https://en.wikipedia.org/wiki/Query_string). One of the original uses was to contain the content of a webform, but it can be used to send arbitrary key-value pairs.

This is an example URL with a query string that has two parameters:

http://example.com/path/to/page?key1=value1&key2=value2

## Task 2.2 - HttpLedControl

**Goal: Control a LED using HTTP**

**Score: 20**

On http://<device IP>/led?set=1, turn on the LED,  
on http://<device IP>/led?set=0, turn off the LED.

On http://<device IP>/led, serve the state of the LED connected to the ESP: "1" for on, "0" for off state (just one character). Also return the same character if the set=1 or set=0 parameter was set. If set=<something invalid>, return "ERROR".

In an HTTP request like /led?set=1, "/led" is part of the "path", and parameters can come after the "?" character, in key=value1&key2=value2 format.

This method of class ESP8266WebServer can come handy:

const String& arg(String name) const; // get request argument value by name

See <https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WebServer/src/ESP8266WebServer.h>

This readme section is covering this, but the function parameters are not mentioned there:

<https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer#getting-information-about-request-arguments>

**Evaluation:**

If you're ready, call for one of the organizers to visually inspect the LEDs.

## Task 2.3 - HttpBasicStatusText

**Goal: Serve some info about your device over HTTP**

**Score: 20**

On http://<device IP>/status, serve the following content:

team: teamX

chipId: {chip id in HEX format}

mac: {MAC address}

ip: {IP address}

ssid: {WiFi SSID}

bssid: {BSSID}

rssi: {RSSI}

led: {true/false}

Use "text/plain" MIME type, LINUX-style line breaks (\n).

This task will be evaluated automatically.

## Learning - JSON

See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_WSVsuMcykG0f>

## Learning - ArduinoJson

See <https://sites.google.com/view/eestech-2019-zurich/study-material#h.p_j57tWryCkHaZ>

## Task 2.4 - HttpBasicStatusJson

**Goal: Serve the same info also in JSON format**

**Score: 20**

On http://<device IP>/status?format=json, serve a content like this, with your respective values:

{

"team":"teamX",

"chipId":"56eed8",

"WiFi":{

"MAC":"DC:4F:22:56:EE:D8",

"IP":"192.168.0.234",

"GatewayIP":"192.168.0.1",

"hostname":"ESP\_56EED8",

"SSID":"EESTech",

"BSSID":"50:46:5D:01:55:92",

"RSSI":-62

},

"system":{

"freeHeap":6520,

"sketchSize":501200,

"sketchMD5":"d7b99f15b2b0c8923d2e0a18b5a8053f",

"resetReason":"Power on",

"upTime":4497215

},

"state":{

"led": false

}

}

The result doesn't have to be pretty-printed like above. Line breaks and indentation are not necessary, it just needs to be in a proper JSON format.

Description of values:

* team - username (or team identifier), e.g. team1, same as the user name that you got.
* chipId - value of ESP.getChipId() in hex format (lowercase)
* WiFi.MAC - MAC address of the WiFi chip. In the format WiFi.macAddress() returns it.
* Various data about the WiFi connection
  + WiFi.IP - The IP address of the device
  + WiFi.GatewayIP - Gateway address
  + WiFi.hostname - the hostname of the device
  + WiFi.SSID
  + WiFi.BSSID - Use WiFi.BSSIDstr()
  + WiFi.RSSI - WiFi
* system properties: get from the ESP object
  + system.upTime: the result of millis()
* state.led: whether the LED is on, from the previous task.

## Learning - Network Time Protocol - (S)NTP

The Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. It is extremely widely used to keep the clocks of online computers in sync.

<https://en.wikipedia.org/wiki/Network_Time_Protocol>

**Evaluation:**

If you're ready, call for one of the organizers to visually inspect the LEDs.

## Task 2.5 - SntpTime

**Goal: Get the current time using the Internet (SNTP) and also expose it over HTTP.**

**Score: 10**

On http://<device IP>/time, serve the current UNIX timestamp (seconds since Jan 1 1970).

This will be needed for verifying server certificates for TLS (used for HTTPS and MQTTS), because certificates have expiration times, so the client has to be able to tell what the current time is.

<https://en.wikipedia.org/wiki/Network_Time_Protocol>

You can find code for getting the time using NTP in the [ESP8266WiFi > BearSSL\_CertStore](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/examples/BearSSL_CertStore/BearSSL_CertStore.ino#L135) example.

The example uses the time.h / time.cc from ESP8266 core:

See the configTime() function here:

<https://github.com/esp8266/Arduino/blob/master/cores/esp8266/Arduino.h>

<https://github.com/esp8266/Arduino/blob/master/tools/sdk/lwip/include/lwip/app/time.h>

<https://github.com/esp8266/Arduino/blob/master/cores/esp8266/time.cpp>

We recommend using the following time servers (to be quick to get the time):

1. time.google.com
2. pool.ntp.org