Co-funded by the
Erasmus+ Programme
of the European Union

UNIT 14: COMMUNICATION PROTOCOLS

AIMS

Purpose of this unit is to become acquainted with communication protocols like Bluetooth, Ethernet and WiFi.

THEORY SECTION

- INTRODUCTION
- BLUETOOTH
  - Design, specifications and features
- ETHERNET
  - Hardware and speed ratings
- WiFi
  - Range and speed
  - Security
- The HTTP protocol
  - Formatting the request

PRACTICE SECTION

- EXAMPLE 1: Exchanging data with a mobile device through an established Bluetooth connection.
- EXAMPLE 2: Sending sensor data to a remote HTTP server via a wired Ethernet connection.
- EXAMPLE 2: How to set-up the web server to run this example.
- EXAMPLE 3: Sending sensor data to a remote HTTP server via a wireless WiFi connection (ESP32-DEVKIT).

REQUIRED MATERIALS

- A desktop or laptop computer.
- The Arduino IDE: this should include the supplementary material already installed and configured.
- An Arduino UNO microcontroller board.
- A USB cable.

Table of Contents

THEORY SECTION

1. INTRODUCTION

Protocols like Bluetooth, Ethernet and WiFi are popular for IoT (Internet of Things) applications. We will not examine each and every protocol in extent; instead we will follow a shorter, introductory course, with fewer details.

2. BLUETOOTH

Bluetooth provides a standardized way of exchanging data between smartphones, tablets, personal computers, peripherals and other devices.

The protocol was conceptualized with the purpose to replace industrial RS-232 data cables, but later evolved into the driving force behind WPANs (wireless personal area networks) and ad-hoc communications.

A. Design, specification and features

Bluetooth is a packet-based protocol with a master/slave architecture. It can accept up to 7 simultaneously connected devices in what is called a “piconet”; not all Bluetooth devices can reach this maximum, though.

It operates at frequencies between 2402 and 2480 MHz. These frequencies are named the “industrial, scientific and medical” (ISM) short-range frequency bands, and they are unlicensed, but not unregulated.

Bluetooth has a number of different “profiles”. These are predefined behaviours that Bluetooth-enabled devices adhere to, so they can communicate with each other in a standardized manner.

With the exception of the newer Bluetooth 5.0, Bluetooth has a rather short –up to 100m- physical range, although its actual performance depends on a number of factors, such as transmitter power, receiver sensitivity, obstacles in the line-of-sight, etc.

Consult the table below for an approximation of the protocol’s physical range.

Class	Maximum permitted power		Typical range (in meters)
	(in mW)	(in dBm)	
1	100	20	~100
2	2.5	4	~10
3	1	0	~1
4	0.5	-3	~0.5

Its maximum attainable speed, just like its physical range, can vary. In practise, speeds greater than 1 Mbit/s are rarely observable.

3. ETHERNET

Unlike Bluetooth, Ethernet is a family of technologies for wired communications, used extensively in various networking schemes. It has been standardized as 802.3 by IEEE in 1985 for use in wired local area networks.

Various Ethernet standards can provide services up to the data link layer of the OSI model. Higher layers, like the transport layer, will be examined later on.

Ethernet slices streams of data into shorter pieces called frames. These frames, along with the actual data, also carry destination and source addresses and error-checking information, in order for the system to detect and discard damaged frames.

A. Hardware and speed ratings

At the beginning, Ethernet used coaxial cabling on a bus-style topology with limited bandwidth topping at 3 Mbit/s (Token Ring LAN), but evolved into using shielded twisted pair cabling, optic fiber media and point-to-point links.

Various Ethernet implementations provide for a speed magnitude of 1 Mbit/s and up to 400 Gbit/s (as of 2018).

4. WiFi

One of the most popular wireless communication protocols is WiFi, since it is available at almost any public or private space. It is based on the IEEE 802.11 standards, and whilst it is very versatile, it lacks power efficiency, which makes its use for IoT applications rather cumbersome.

Different versions of WiFi exist, each one with different ranges, radio bands and speeds. The most commonly used bands are 2.4 GHz (UHF) and 5.0 GHz (SHF). These bands are divided into multiple channels, and each channel can be shared across multiple networks using time-division techniques.

A. Range and speed

Just like any radio technology, WiFi speed and range is dependent upon various factors, such as the frequency band, the transmitter's power output, the receiver's sensitivity, the type of the antenna used, its gain, etc.

In general, WiFi transmitters are considered low power devices, with several regulations existing for specifying its upper power limits. In the European Union, the equivalent isotropically radiated power (EIRP) is limited to 20 dBm, or 100 mW.

B. Security

With WiFi being a wireless technology, one simply has to be in the network's range to gain access to it. With wired technologies, such as Ethernet, this is not much of a problem since the network is physically limited.

Several ways of securing a wireless network exist. Common ones include:

- Hiding the access point's broadcasted name (SSID);
- Only allowing know clients to join the network (MAC address filtering);
- Utilizing encryption, as to protect the transmitted data.

The latter is considered the most secure way of interacting with a WiFi network, although, in practice, several security options are employed in conjunction.

5. The HTTP protocol

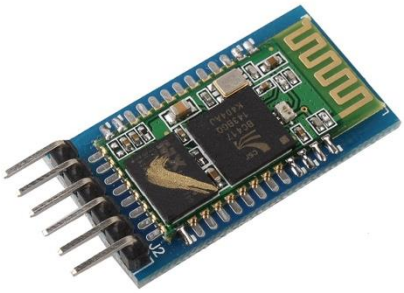
The hypertext transfer protocol (HTTP) allows for a number of request methods for a client to communicate with a server. One of the most common of them, is the GET request. It is specified by the RFC 2616 specification and it is part of HTTP/1.1.

A. Formatting an HTTP request

A request from a client to a server includes, within the first line of that message, the method to be applied to the resource (GET, POST, etc.), the identifier of the resource, and the protocol version in use (this is called “the request line”). These are followed by a number of (optional) headers which allow the client to pass additional information about the request, or about the client itself, to the server. An empty line (a line with nothing preceding the CRLF) indicates the end of the header fields, and the message-body follows.

PRACTICE SECTION: EXAMPLES

EXAMPLE 1: Exchanging data with a mobile device through an established Bluetooth connection.

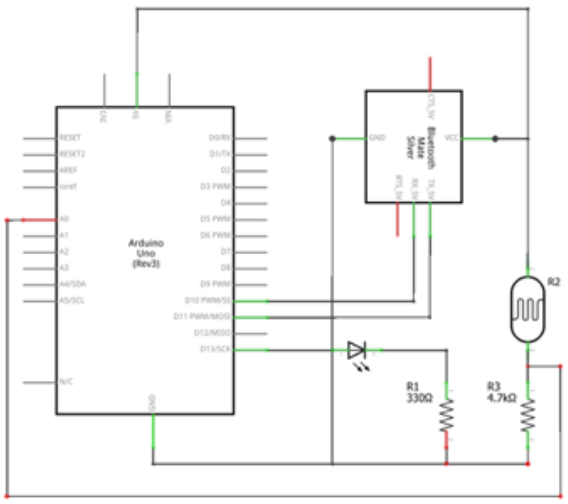


Bluetooth can be a “handy” protocol for Arduino-based applications, since the available solutions are affordable and easy to work with.

One of the most widely available Bluetooth modules for the Arduino is the HC-05 (see the photo on the right) which allows for easy interfacing via serial (UART) communication. This is the module that we are going to use in this exercise.

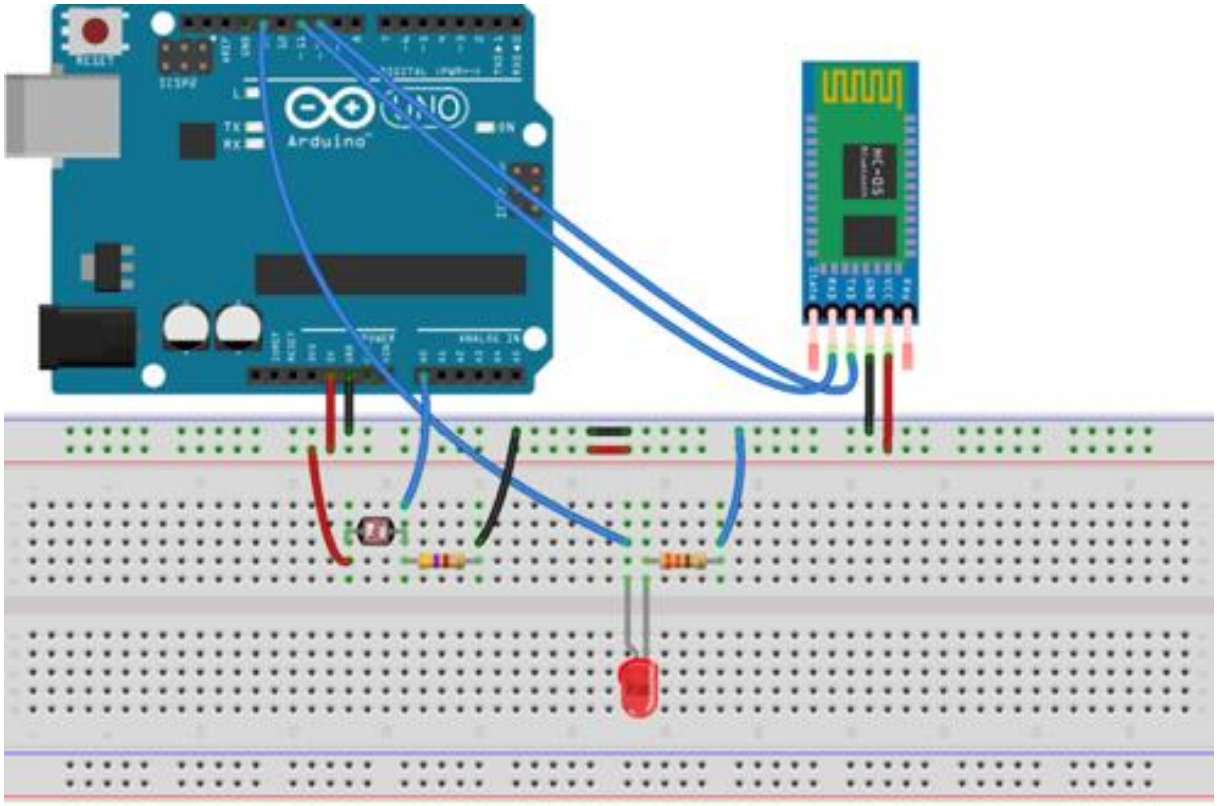
Alongside with the Bluetooth module, we are going to use a (freely available) application for the Android operating system, called “Serial Bluetooth Terminal” by Kai Morich. As its’ name implies, it is a terminal emulator.

Consult the following schematic diagram to implement the circuit for this example;



The light-dependent resistor –R2- alongside with R3, form a voltage divider from which we will be reading the luminance levels of the room.

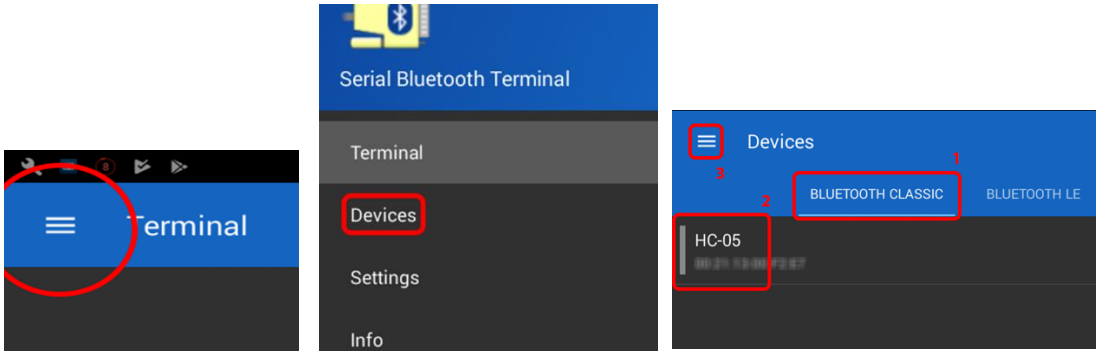
The same circuit is presented on a breadboard, to ease your prototyping.



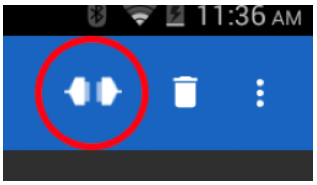
As soon as you complete the circuit, upload the provided sketch to your Arduino, and fire the application on your mobile phone or tablet.

Before being able to exchange data, we will need to establish a Bluetooth connection between the module and the mobile device.

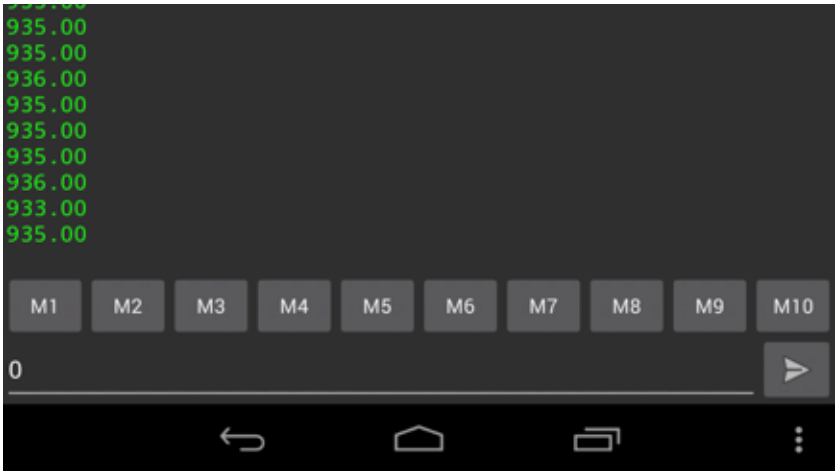
Tap the icon next to the ‘Terminal’ label to open the application’s main menu, and from it, tap ‘Devices’.



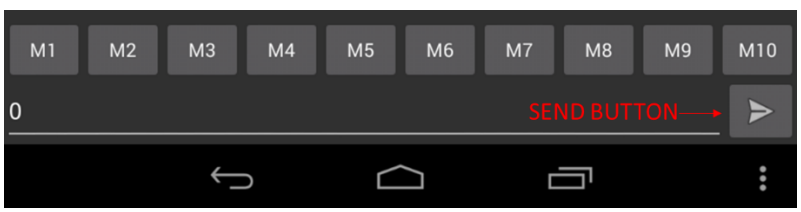
For the HC-05 module that we are using, we will need to select 'Bluetooth Classic', not 'Bluetooth LE'. Then tap on the module's name to select it. Tap again on the menu icon and pick 'Terminal' to return to the main screen. The last step is to tap the connect icon, to initiate the connection.



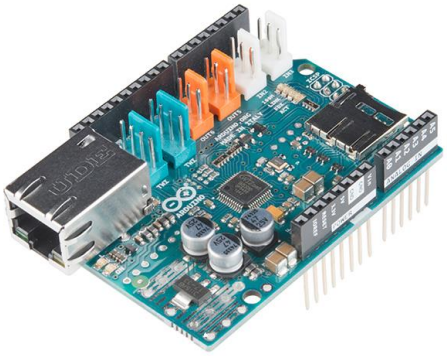
That was it! Now the application is ready to exchange data with the module! In fact, you should already be seeing the analog voltage readings from the Arduino being transmitted to the application.



Other than that, you should also be able to flick the connected LED on or off, simply by issuing single-character commands, '0' or '1'.



EXAMPLE 2: Sending sensor data to a remote HTTP server via a wired Ethernet connection.

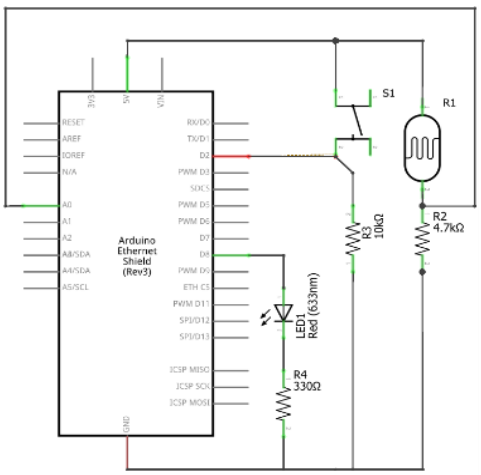


Arduino cannot communicate on either wired or wireless networks by itself, since it lacks the required hardware.

On this example, we are going to used what is called an "Ethernet Shield" to provide Arduino with the required hardware to connect to a wired network. Apart from the rather obvious RJ-45 connector, which is physically required for connecting the network cable, what is also important is the Wiznet W5500 Ethernet IC, which hosts a network stack capable for both TCP and UDP. The Ethernet Shield communicates with the Arduino over SPI (pins 10, 11, 12 and 13 on the Arduino Uno). Be aware that those pins, cannot be used for other purposes.

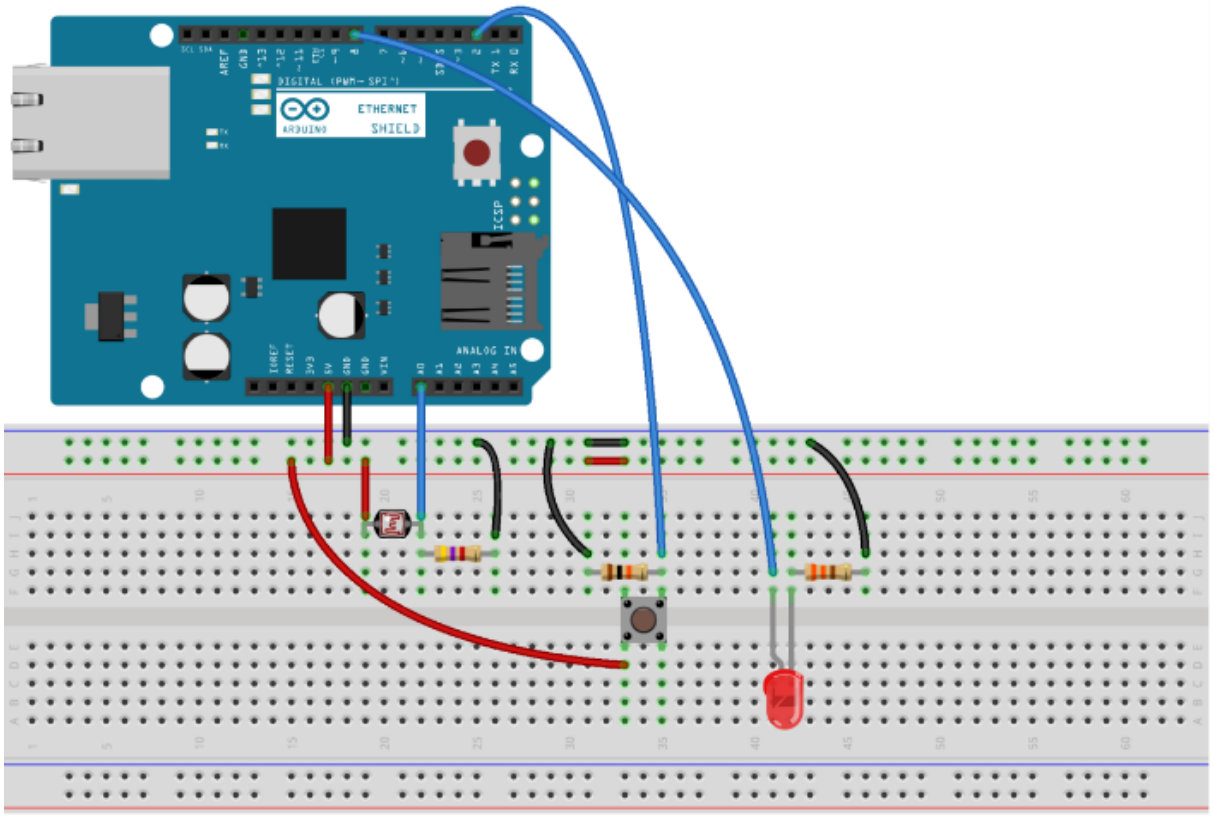
A custom web application has been developed for us to work on. The development the application itself, however, is outside the scope of the course.

Consult the following schematic diagram to implement the circuit for this example;



The LDR (R1) alongside with R2 form a voltage divider from which we will be making analog voltage measurements The switch (S1) triggers an interrupt on GPIO pin 2 when clicked.

The same circuit is presented on a breadboard, to ease your prototyping.



As soon as you complete the circuit and connect the Ethernet Shield to your network, upload the provided sketch to your Arduino, and launch a web browser on your desktop or laptop computer.

It is important to remember to add a few network settings (such as the MAC address, IP address, subnet mask, DNS server and gateway) if you are on a network incapable of DHCP. If you are on a DHCP-enabled network, setting those values is not required.

Let’s examine these values, before we continue with the example;

A media access control address (MAC address) is a unique identifier assigned to a network interface controller by its manufacturer. It is applicable for both Ethernet and WiFi. The Ethernet Shield has its MAC address printed on a sticker underneath it. If using a brand other

than Arduino/Genuino, you may need to manually pick a MAC address (pick one at random; the chances that you will come across the same address in your network are nil). Declaring the MAC address of the Ethernet Shield in your code is mandatory for the sketch to work.

The internet protocol address (IP address) is a 32-bit (in IPv4) numerical label assigned to each device on your network.

The subnet mask is a bitmask applicable to your network, that when applied by a bitwise AND operation to any IP address in the network, yields the routing prefix.

The DNS server is a server that translates domain names (e.g. “google.com”) into their corresponding IP addresses (e.g. “74.125.224.72”).

The gateway, finally, refers loosely to your router.

Typically, the values discussed above (with the exception of the MAC address) will get assigned automatically through a DHCP network sever. However, it is possible for the Arduino to request for specific addresses.

Now, let’s talk a bit about the code…

```
client.stop();
```

The above method stops any previous connection left open.

```
client.connect(host, 80);
```

In this method, a new connection is established. Arduino tries to connect to port 80 (HTTP server) of “host” (defined earlied).

```
if ( client.connected() )  {
    client.print((String)"GET /arduino/getdata.php?deviceid=" + String(device) + " HTTP/1.0\r\n"
                + "User-Agent: Arduino\r\n"
```

```
                + "host: " + host + "\r\n"
                + "Connection: close\r\n\r\n");
    timeout_timer = millis();
  }  else  {
    Serial.println("Client not connected.");
    return;
  }
```

The code above sends an HTTP GET request to a specific script that is meant to return either '0' or '1', so that the Arduino can change the state of the LED. It first checks for an active connection. If the check succeeds, the GET request get formatted and sent to the host. If on the other hand, the check fails, a message is printed on the serial port and loop returns.

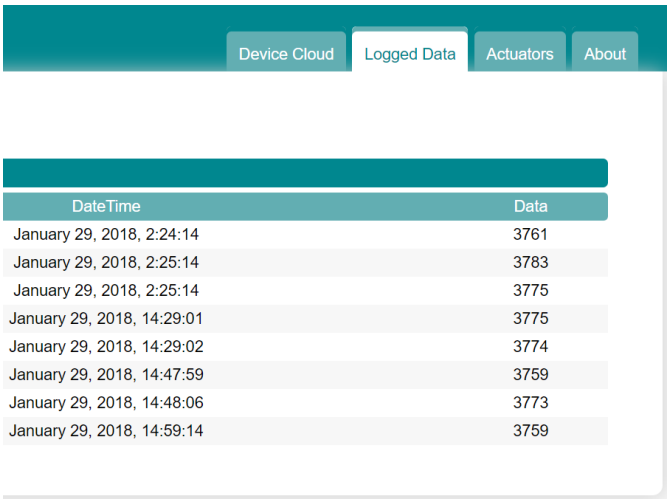
```
while ( client.available() )  {
    buffer = client.read();
  }
  if (buffer == '1')  {
    digitalWrite(GPIO_led, HIGH);
  }
  else  {
    digitalWrite(GPIO_led, LOW);
  }
```

The above code checks for any incoming data (i.e. the server has responded) and reacts accordingly: if the data received is a '1' it turns the LED on. If it's a '0', it turns it off.

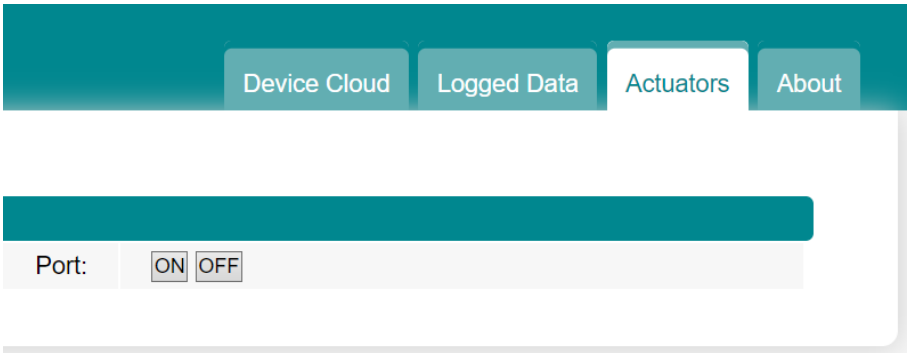
Lastly, the code inside the `if (button)` block, acts in a similar manner: terminates any connection that has been kept-alive, re-connects to the host and sends a request. Only this time, instead of sending the device identifier and waiting for a response, it just sends some data (read from the ADC) and then exits.

`Ethernet.maintain()` just before exiting the loop renews the lease for the Arduino on the gateway.

When you click the button, the interrupt fires and the values from the Arduino's ADC get uploaded to the web application. You should be able to see them by refreshing the web application (hit F5 - it does not refresh dynamically).



You should also be able to change the status of the connected LED by simply clicking the 'ON' / 'OFF' links on the web application, under your client's name.



EXAMPLE 2: How to set-up the web server to run this example.

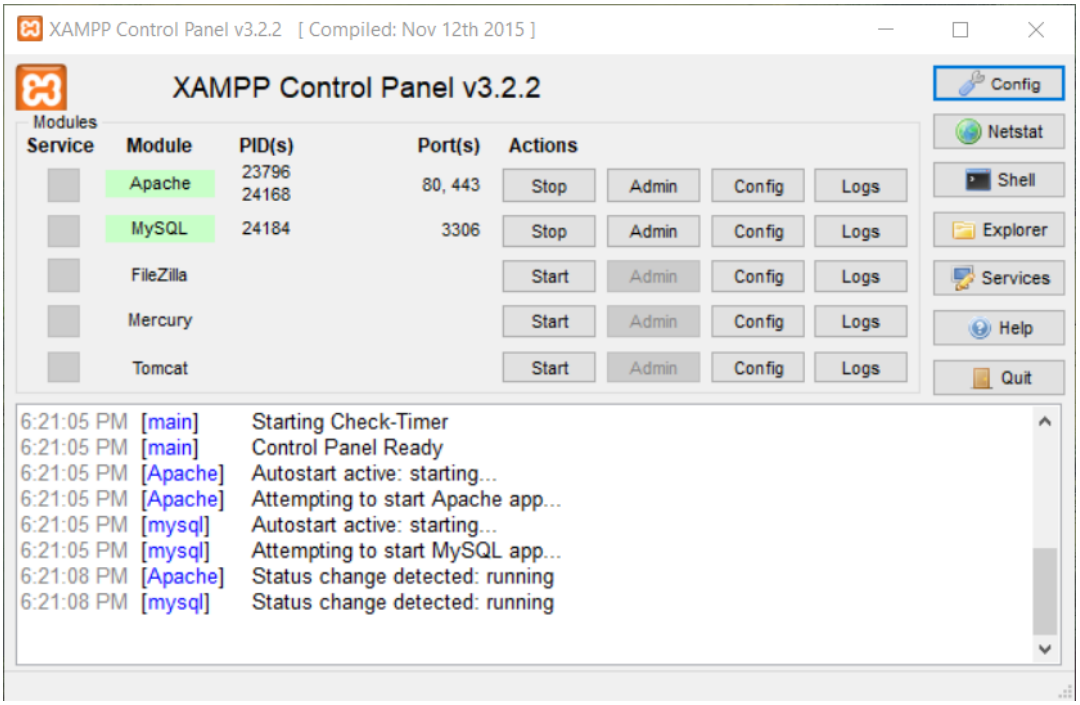
Whilst the web application has already been developed, running it requires an HTTP server alongside with a few key components to be installed.

The web application has been developed with WAMP/LAMP setups in mind, so you will need to provide the appropriate software stack for it to run. Luckily, there are quite a few easily installable packages that provide the required software and configuration settings.

In this guide we are going to install the XAMPP package for Windows, which is one of them.

Download and install the latest XAMPP package from https://www.apachefriends.org/. Pick the latest PHP version (7.2+) that is available. The installation is pretty straightforward, since it is a point-and-click guided install, just like most software packages you already run in your computer.

After the installation is finished, launch the XAMPP Control Panel and make sure that the Apache and MySQL servers are up and running.



Also, launch a terminal (click on the Start button, type '`cmd.exe`' and hit Enter) and issue the `ipconfig` command to find your computer's local IP address. You will need to enter this address in your Arduino sketch.

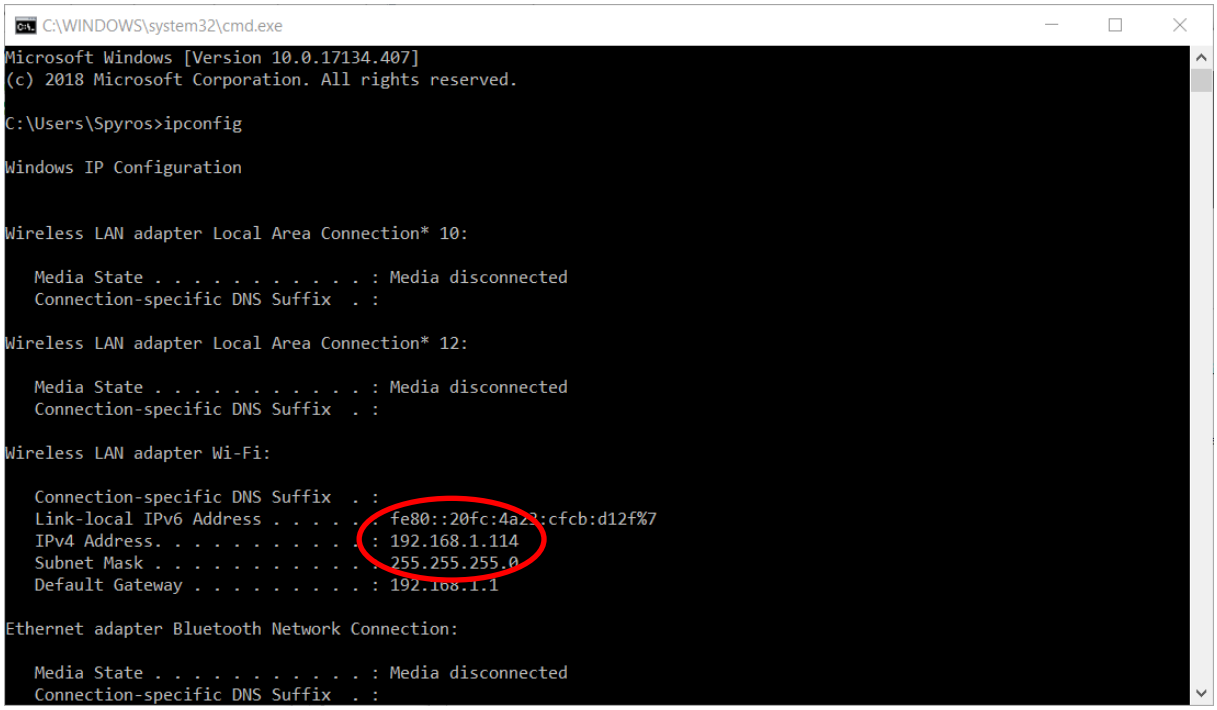


Figure: A command-line window presenting the computer's local IP address.

The next step is to navigate to `localhost` from a web browser and launch phpMyAdmin from the top navigation menu. We are going to use this application to import the database.

The left sidebar presents the available databases. We will create a new one by clicking on the 'New' button. Name the new database '`arduino`' and pick '`utf8_bin`' as the collation. But do not create any tables yet!

With the newly created database selected, click on the '`Import`' tab atop. Browse for the provided `arduino.sql` schema file leaving the default options selected and hit Go.

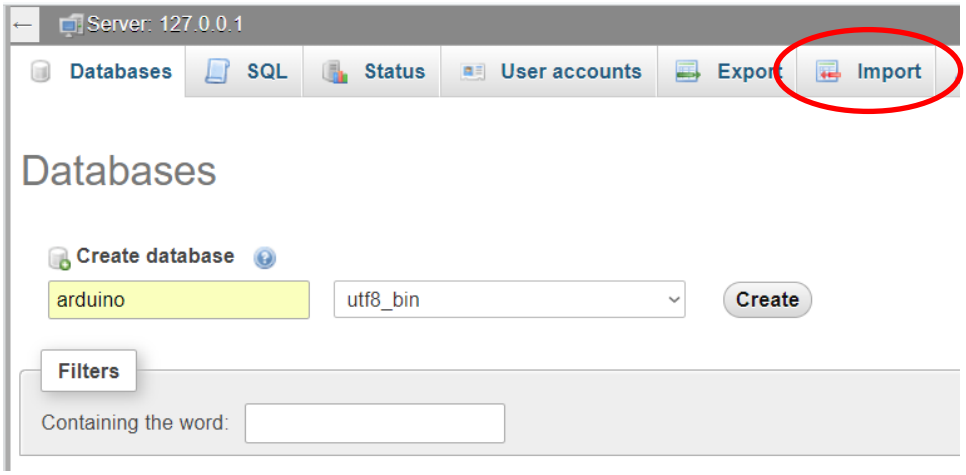


Figure: Creating a new database with phpMyAdmin. The 'Import' button is also visible.

That was it! Now the database structure has been imported, and the database is ready to accept queries from the web application.

The last step is to copy and paste the '`arduinopanel`' directory inside your web server's root directory. If you followed along and installed XAMPP with its default options selected, this should be `<ROOT_DISK>\XAMPP\htdocs\`. Your `<ROOT_DISK>` is probably your `C:\` drive.

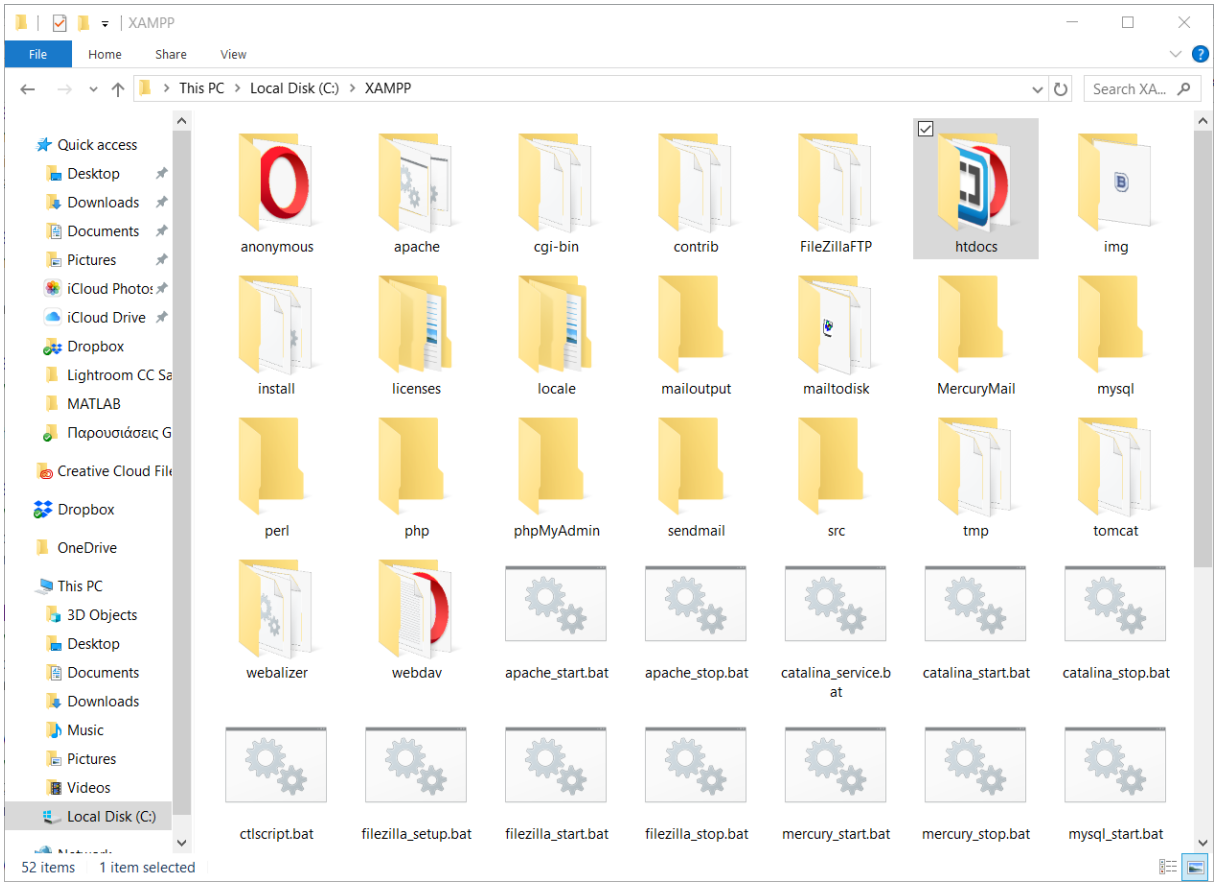


Figure: XAMPP's directory contents. The web application is mean to live inside the 'htdocs' directory (the highlighted one).

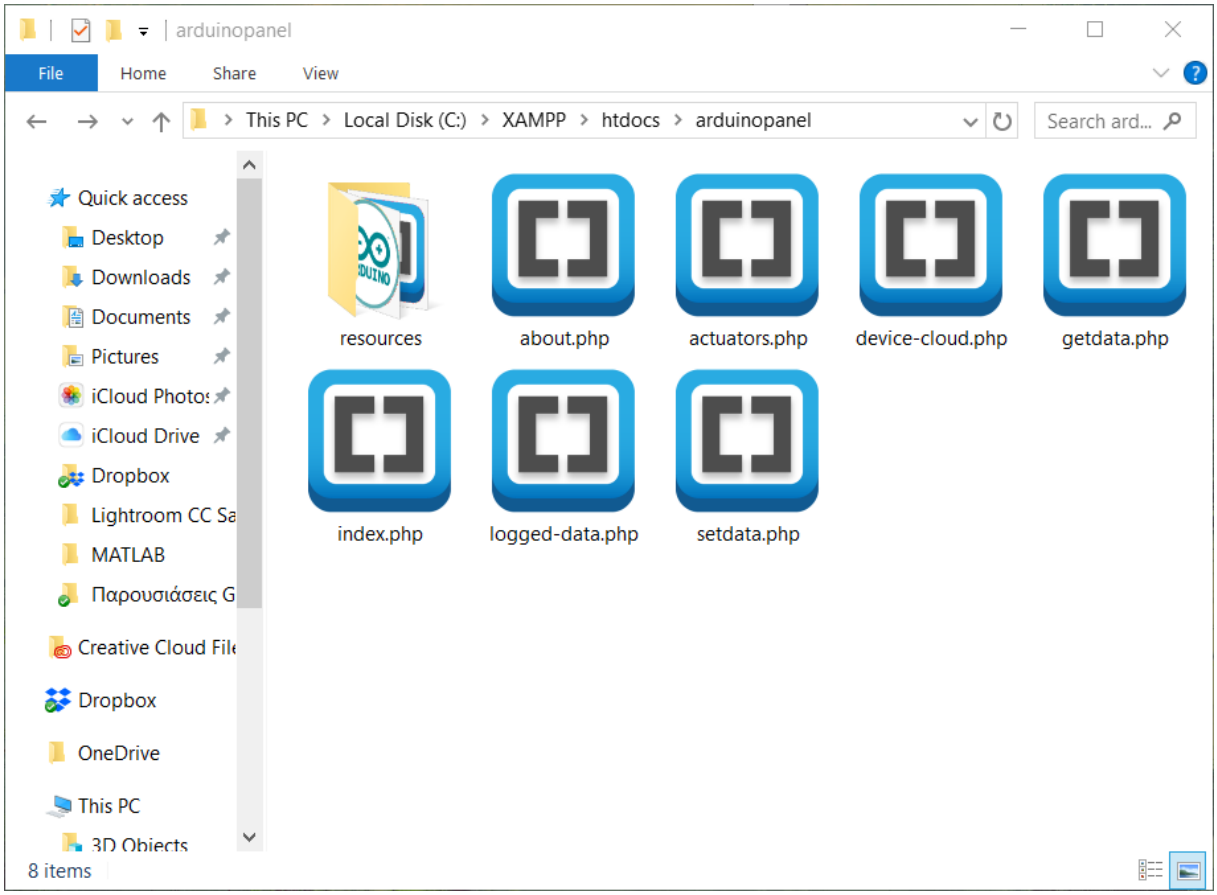


Figure: The contents of the web application.

Now visit http://localhost/arduinopanel/ to run the web application.

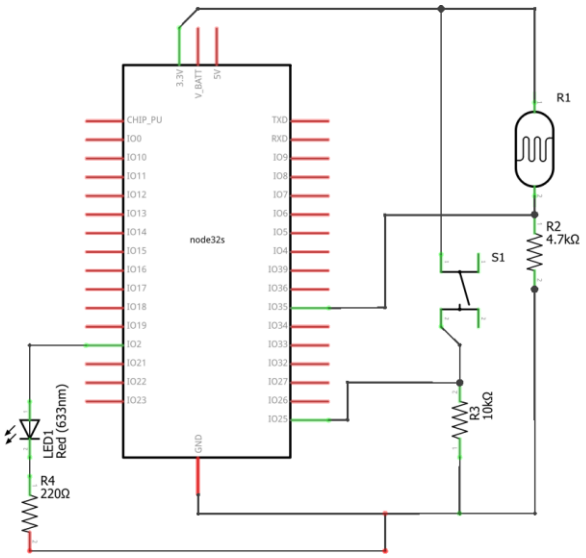
EXAMPLE 3: Sending sensor data to a remote HTTP server via a wireless WiFi connection (ESP32-DEVKIT).

For enabling WiFi communication on the Arduino, there is no straightforward option, as the equivalent “WiFi Shield” is a rather expensive solution.

Luckily for us, a China-based company called Espressif has developed a chip called ‘ESP32’ which is exactly what we were looking for: an affordable, Arduino-programmable microcontroller with WiFi capabilities.

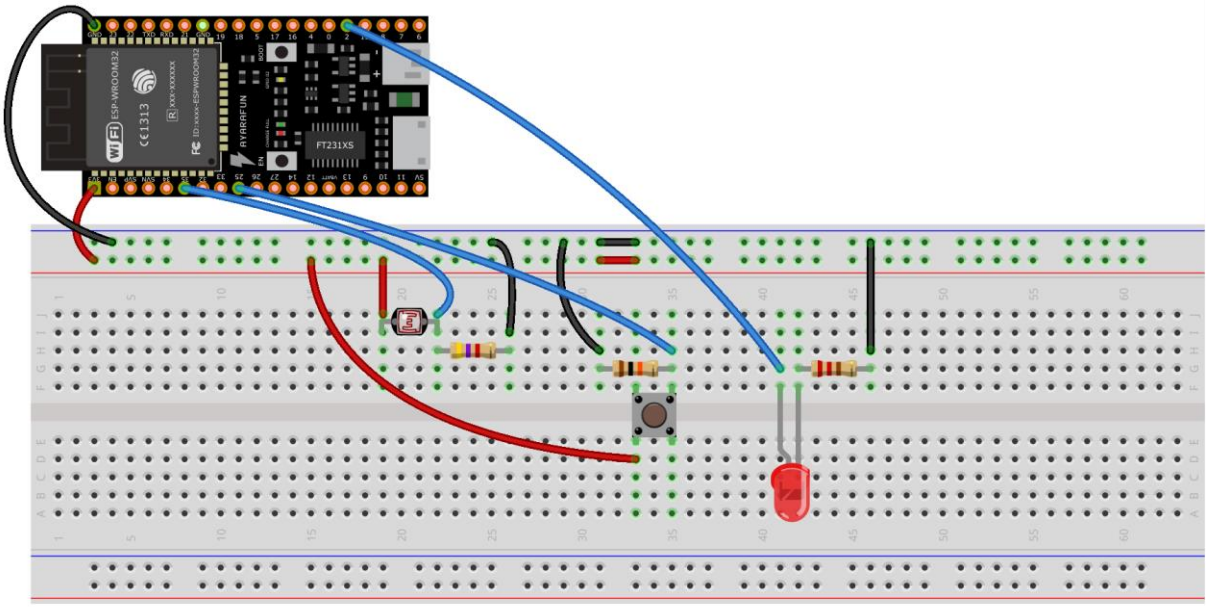
The ESP32-DEVKIT by doit.am is a microcontroller board that hosts the aforementioned chip, and we are going to use it –in place of the Arduino- to communicate with the web application that we used in the previous example.

The schematic diagram for this exercise is very similar to the previous one:



Note that the GPIO pins for the LED and the switch have been changed, to facilitate the differences the ESP32 has.

The same circuit is presented on a breadboard, to ease your prototyping.



A WiFi network, unless configured not to do so, broadcasts an SSID (read: network's name). This SSID, alongside with the network's password, for password-protected networks, are the two required attributes for a successful connection.

The code changes in this exercise facilitate this:

```
const char ssid[] = "ssid";

const char pass[] = "pass";
```

Inside `setup()`, we will try to connect to the wireless network. This is accomplished by the `WiFi.begin(ssid, pass)` method.

Following that, a while loop with a conditional statement that checks for the status of the connection makes sure that the code thereafter will not be executed until the connection has been established.

```
while ( WiFi.status() != WL_CONNECTED )

  {

    delay(500);
```

```
    Serial.print(".");
  }
```

The exercise actually follows the same scenario; hence, you already know how to use the web application in order to interact with the ESP32 and the logic of the sketch.

REFERENCES

BOOKS

WEBSITES

[1]. https://www.arduino.cc/

[2]. https://electronics.stackexchange.com/

[3]. http://www.espressif.com/

[4]. https://www.w3.org/