Usage of IoT in automation routine processes

Mircescu Cornel
The Bucharest University of Economic Studies, Romania
mircescucornel18@stud.ase.ro

Internet of Things or IoT is a representation of the automatic processes by integrating processors, sensors and data collections through a huge amount of processing techniques. The contribution of IoT in technology is significant, offering revolutionary changes in a big amount of industries. [1]
Actually, these devices became so powerful that they can work with data in real-time, doing ***machine learning*** *and* ***artificial intelligence****. In big words, a single processor got the frequency up to 2.4 GHz, with integrated Wi-Fi and many connectors, for the connection with sensors, being capable of simultaneously working with all of them, depending of the back-end efficiency programming.*
Initially, the IoT devices were programmable only by the Arduino IDE, which is based on C++. After some time, languages like Python and C# opened their support by creating or opening open-source projects for libraries to work with them, processors being recognized as a programmable mini PC. USB was the only solution for code upload, the IDE doing the compilation and the transfer, then the project could just work with a source of electricity (for smaller projects, a powerbank could be enough). The programming language Python evolved and they released, through a open-source project, OTA (Over The Air) upload, which means the project will need just the usual electricity source and Wi-Fi, no more physical connection to the PC.
Keywords: *iot, big data, database, programming, protocol, sensor, module, upload.*

1 Introduction

Arduino is a brand which is formed with hardware (processors) and software (IDE) components, where anyone can build automation projects. Most libraries from the IDE are open-source, so the users are able to copy and alter the code as they like. Besides the platform, Arduino got a lot of processors and modules, at low-moderate cost, for every type of project needed. Almost all processors got attachments, for a small fee, which can extend their capabilities, like: Wi-Fi, GSM, storage shields.

Speaking of the power of a single processor, the Arduino can stand as a server with ease, being even able to work with the data received from the client. The sensors will complete the board, meaning there will be data received from the IoT device; they can also be analogical or digital, Arduino being able to handle both with specific ports, controlling the voltage it drives through the sensors, careful enough not to burn them.
Some analogical sensors will need physical intervention, because they usually have buttons.

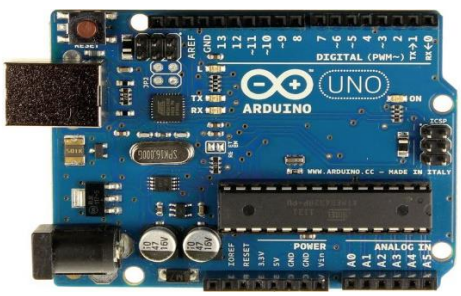


Fig. 1 Arduino UNO Board

2. Software application for routine processes

The project have the following components:

- Application in the Arduino IDE: for decision control for every sensor and module, offering analysis data by inserting it into the SQL Server database. Arduino hardware is meant to give LOW and HIGH signals to the output modules and to read data measured by the input sensors.

When the data comes in critical values, the C# client will be instantly contacting the user via e-mail, showing a page with the values. For this to be done, a SMTP server will be contacted, in our case, Google. Speaking of network, the processor will be the kernel of the effective processes upon the plants, having the highest priority and access at the point of taking decisions. In this platform, we will use TCP model for transmission of data, where we can create a 2-way communication between the server (Arduino) and the client (C# app). The client app will be uploaded on the processor, being ready for execution, even when there is no physical access to the PC. It will only need a power supply and Wi-Fi connection.

In the Arduino IDE, the graphical user interface will be 0, because it will be only used to transmit data to the client. C# will take all the variable passed through the TCP model and will start working with the available data. The speed will be slightly higher without a user interface on the server, just command line.

- C# application as client, having technologies like: ASP .NET for server-side controllers, JavaScript for client-side scripts, SQL Server for the database storage and data processing, CSS for adding a friendly interface.

This application will be quite useful at monitoring data and taking manual actions, where the interface is simple and friendly, where relevant data is shown into the interface, especially on the Chart control. Speaking of network, the web application will be the client, where the user can request data at any moment or by an interval created by the user. The WebForm will be hosted via IIS on a local computer. For a fresh-looking website, Dark Mode has been also added.

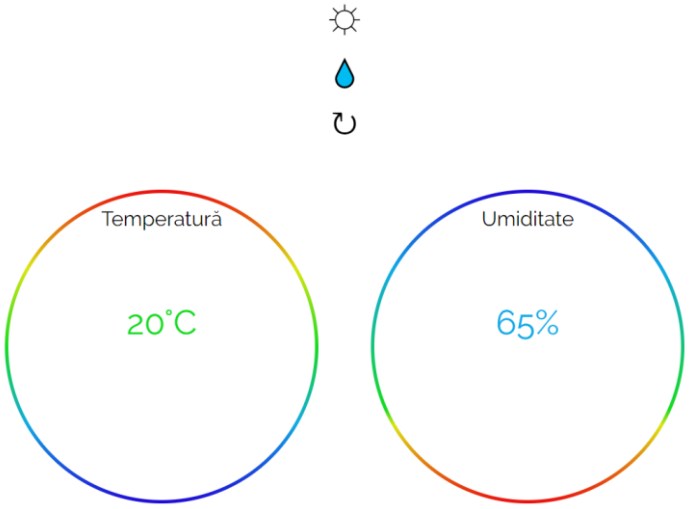


Fig. 2. Light Mode client interface

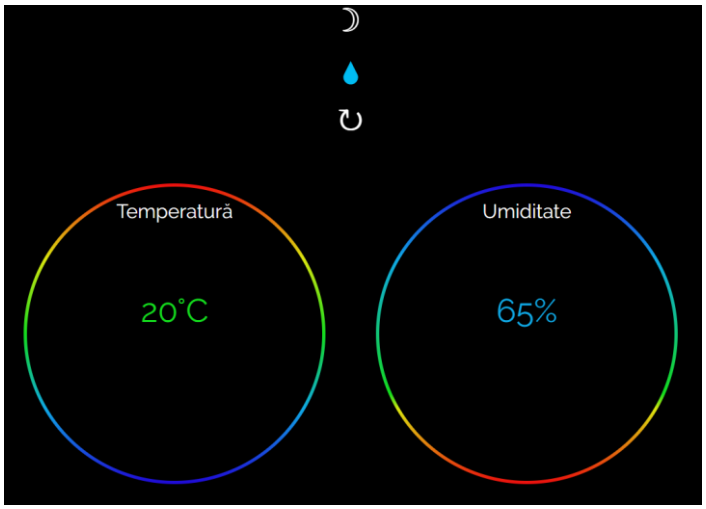


Fig. 3. Dark Mode client interface

To get the Dark Mode working, JavaScript and CSS keyframes are making the work easier, creating a smooth animation for the background change.

```css
@keyframes darkMode {
    0% {
        background-color: white;
        color: black;
    }

    100% {
        background-color: black;
        color: white;
    }
}

@keyframes lightMode {
    0% {
        background-color: black;
        color: black;
    }

    100% {
        background-color: white;
        color: black;
    }
}
```

Fig. 4. CSS Dark/Light mode

In order to get this working in C#, we need to use a server-side script, like JavaScript.

```
ClientScriptManager clientDark = Page.ClientScript;
StringBuilder cstext1 = new StringBuilder();
String csname1 = "darkMode";
cstext1.Append("<script type=text/javascript> document.body.style.animation = 'darkMode 1s forwards'; document.getElementById('tempcircle').style.animation = 'darkMode 1s forwards';
cstext1.Append("document.getElementById('umidcircle').style.animation = 'darkMode 1s forwards'; document.getElementById('monthFilter').style.animation = 'darkMode 1s forwards';");
cstext1.Append("document.getElementById('yearFilter').style.animation = 'darkMode 1s forwards';</ script >");
clientDark.RegisterStartupScript(this.GetType(), csname1, cstext1.ToString());
```

Fig. 5. JavaScript Dark Mode integration C#

```
ClientScriptManager clientLight = Page.ClientScript;
StringBuilder cstext1 = new StringBuilder();
String csname1 = "lightMode";
cstext1.Append("<script type=text/javascript> document.body.style.animation = 'lightMode 1s forwards'; document.getElementById('tempcircle').style.animation = 'lightMode 1s forwards';
cstext1.Append("document.getElementById('umidcircle').style.animation = 'lightMode 1s forwards'; document.getElementById('monthFilter').style.animation = 'lightMode 1s forwards';");
cstext1.Append("document.getElementById('yearFilter').style.animation = 'lightMode 1s forwards';</ script >");
clientLight.RegisterStartupScript(this.GetType(), csname1, cstext1.ToString());
```

Fig. 6. JavaScript Light Mode integration C#

In the client application, buttons with custom text are used (unicode characters) for a more self-explanatory functionality, being recognized by both ASP HTML and C#, but with different characters.

As we can see in the above example, the Web client interface is created with centered unicode characters and 2 variables in the interior of a gradient circle.

Firstly, the Sun and the Moon unicode characters are used as symbols for Dark/Light mode. The action is working client-side with JavaScript, so the server will not interpret any design. The drop of water,

as the second unicode character is meant to sent a signal to the Arduino to immediately water the plants running under this system for approximately 5 seconds. Furthermore, the client is sending a request to the server with a signal sent via the TCP model. When the server finish the job, it will send a confirmation of success to the client, the data being processed and, finally, inserted into the specific database. The 3rd element, the unicode character of refresh is created to force a request to the server, so it can return newer data, anytime the user wants to. Anytime when a value exceeds or goes under the critical low value, the C# will instantly create a mail and send it to the recipients.

```
if (((temp<5) || (temp>40) || ((umid<45) || (umid>90)))
{
  digitalWrite(pompa, HIGH);
  client.send("pompa a fost pornita!")
  smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount, emailSenderPassword);
  smtpData.setSender("NodeMCU", emailSenderAccount);
  smtpData.setPriority("High");
  smtpData.setSubject(emailSubject);
  smtpData.setMessage("<div style=\"color:#2f4468;\"><h1>Alerta notifiare</h1><p>- Temperatura:" + temp ", Umiditate: " + umid"</p></div>", true);
  smtpData.addRecipient(emailRecipient);
  smtpData.setSendCallback(sendCallback);
  if (!MailClient.sendMail(smtpData))
    Serial.println("Mail-ul nu se poate trimite; " + MailClient.smtpErrorReason());
  smtpData.empty();

}
```

Fig. 4. Automatic mail creation and transmission

3. Hardware application for routine processes

For this project to be done, the following components were needed:

- Processor NodeMCU 12E – low-cost, efficient, open-source, integrated Wi-Fi, powerful processor – 2.4 GHz.

Arduino IDE compatible, the NodeMCU is the ideal module for small-medium projects in IoT, being able to work with many Arduino libraries. It has 9 digital pins, meaning that we can attach maximum 9 digital sensors or less (depending of the number of pins the sensors use). [2]

As example, a sensor of humidity and temperature needs only a pin for transmitting data, while an OLED display will use 2 pins for showing data. Thanks to the Wi-Fi integrated module, the microprocessor will be able to communicate data to the client Web with the TCP model of data transmission.

Fig. 5. NodeMCU 12E

- Temperature and humidity sensor Adafruit DHT22: digital, efficient, accurate.

The DHT22 is a modern sensor which can take accurate values of temperature and humidity, with a 2 seconds minimum interval between measurements. [3]

These values will be sent to the Web client for immediate availability in the recalculation of statistic analysis.

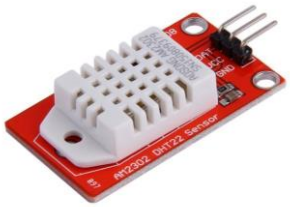


Fig. 6. Adafruit DHT22

- Breadboard – fast connection, no soldering, useful for prototyping.

The breadboard is a fundamental piece for IoT, making the work easier, because of the electrical conductors integrated on every pin. This connectivity board is a friendly welcome to everyone who is

new in Internet of Things, because of the simplicity of it. The processor links with sensors and modules via soldering or breadboard. For example, with the help of jumper cables, we can insert one into the GND (”-” polarity) pin of the processor, and the other side into the GND pin of the sensor/module, and analog to the VCC (”+” polarity), depending on the power the sensor needs (3.3V or 5V). At this point, we created a connection of power, but we will need a pin to take or to send data to a sensor/module. In the same manner, we will just insert the jumper cable into the DAT pin of the sensor and the other side to any pin from D1-9 of the processor, depending of the preferences of the user.

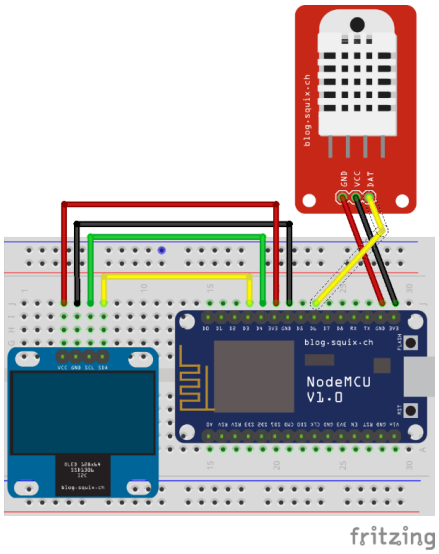


Fig. 7. Breadboard connectivity

As we can see in the above example, the DHT22 sensor is conected to the NodeMCU via 3 pins – 3.3V, GND and D6, meaning that it will have power from the microprocessor and data transmission. The OLED screen will need one more additional data pin for slave-master connection. [4]

- Water pump.

The water pump was connected as a ”Do It Yourself” project, because it is a traditional one and it has normal connectors, not the one we need to connect to the board. But thanks to IoT relays, we can connect the water pump to it and the relay to the NodeMCU, where will get 0 or 1 signals, meaning that the pump is on or off.

This action can be triggered automatic or manual, depending on the stats of the DHT22 sensor and the user intervention.

Fig. 8. Water pump

- Switch button.

This type of module will be used to water the plants manually, with physical intervention, without any need of access to the Web application. For additional functionality, the button have an embedded led to announce the user that the press changed its state. As long as the button is pressed, the water pump will transfer water to the plants.

Fig. 9. Water switch

4. Functionalities and implementation methods

In the Arduino IDE, we must declare every sensor with the number of pin assigned earlier, because from there the data stream will start. In our case, we will declare the library, the pin and the type of the

temperature and humidity sensor, as follows:

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

#define DHTPIN            2
#define DHTTYPE           DHT22     // DHT 22 (AM2302)

DHT_Unified dht(DHTPIN, DHTTYPE);

dht.begin();
  sensor_t sensor;
  dht.temperature().getSensor(&sensor);
  dht.humidity().getSensor(&sensor);
  delayMS = sensor.min_delay / 500;
```

Fig. 9. DHT22 declaration

For a better view of the data stated into the database, I created a Chart control, which will show the temperature and humidity on days/years. The data is directly queried from the SQL Server.

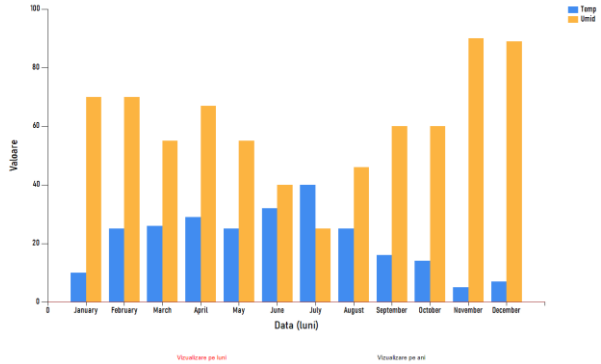


Fig. 10. Temperature/humidity Chart (Light)

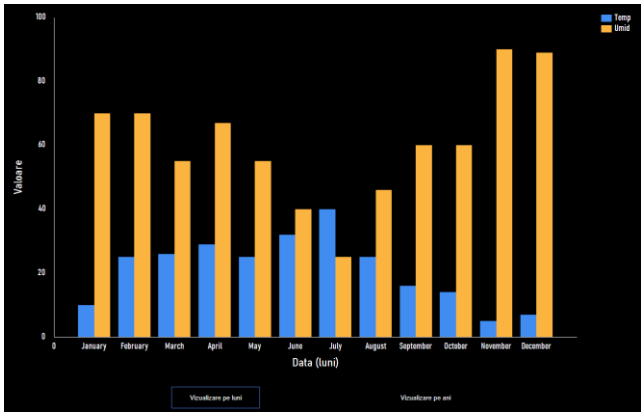


Fig. 11. Temperature/humidity Chart (Light)

4.1. Database structure and management

The creation of database is essential here for long or permanent time interval storage. Moreover, a database was created with 2 tables, one for the temperature and humidity and one for the watering management.

	Name	Data Type	Allow Nulls	Default
🔑	Id	int	☐	
	Temp	float	☑	
	Umid	float	☑	
			☐	

Fig. 12. Temperature/humidity table

I used the **ID** attribute as primary key, with auto-incrementation for easier management of data collection. The attributes **Temp** and **Umid** are floats, there is where the C# will place the values from the DHT22.

	Name	Data Type	Allow Nulls	Default
🔑	Id	int	☐	
	dataOra	datetime2(7)	☑	
	durata	float	☑	
			☐	

Fig. 13. Watering management table

In the same manner, I used ID for better management, **dataOra** attribute will be created as a time stamp when the watering will be done, plus the duration, marked with the attribute **durata**.
Both tables will accept null values, because the temperature can be 0, for example, and we can manage to see when it begun to show errors in data.

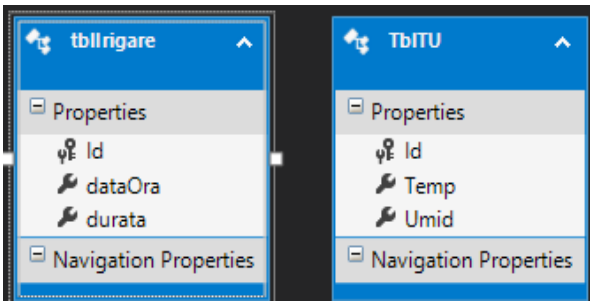


Fig. 14. Database tables

In order to be working with the database, we will need to declare variables to create a variable which will store data received, the connection to the database, the query string, the command and the parameter.

```
SqlConnection myCon=null;
string[] TempUmid;
string myIns = null;
SqlCommand myCmd = null;
SqlParameter pT, pU;
```

Fig. 14. Variables for database interaction

```
public Form1()
{

    pT = new SqlParameter("@Temp", SqlDbType.Float, 8);
    pU = new SqlParameter("@Umid", SqlDbType.Float, 8);


    myIns = "insert into TblTU values (@Temp, @Umid)";
    myCon = new SqlConnection(@"Data Source=(LocalDB)\v11.0;Initial Catalog=tempUmid;Integrated Security=True;Pooling=False");
    myCmd = new SqlCommand(myIns, myCon);
    myCmd.Parameters.Add(pT); myCmd.Parameters.Add(pU);

}
```

Fig. 15. Values for database interaction

In order to take the data from Arduino, we need to use some variables for data received and classification. To work with the data easier, I made the Arduino server to put comma between temperature and humidity so I can manipulate the variable with separator.

```
data = new Byte[256];
String responseData = String.Empty;
Int32 bytes = stream.Read(data, 0, data.Length);
responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes);
string[] TempUmid = responseData.Split(',');
valoareTemp.Text = TempUmid[0] + "°C";
valoareUmid.Text = TempUmid[1] + "%";
```

Fig. 16. Variables for data reception

4.2. Connectivity between the server and client

Because a keep-alive session between Arduino and the web can cause damage to the board due to overheating, a network stream can be opened any time the data is requested. Newer data can be requested: manually, when the refresh control is pressed or automatically, when the page is loaded or when the C# timer interval control is reached. The entire application is meant to work with short sessions.

```
Int32 port = 8090;
TcpClient client = new TcpClient("172.20.10.3", port);
String message = "t";
Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);
NetworkStream stream = client.GetStream();
stream.Write(data, 0, data.Length);

data = new Byte[256];
String responseData = String.Empty;
Int32 bytes = stream.Read(data, 0, data.Length);
responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes);
string[] TempUmid = responseData.Split(',');
valoareTemp.Text = TempUmid[0] + "°C";
valoareUmid.Text = TempUmid[1] + "%";
```

Fig. 17. Creating network stream for fresh data

As we can see in the earlier image, a port was opened both on the Arduino and C# for the 2-way communication. The control TcpClient is a good tool to communicate via internal network. Once the client started, the client will go to the specified IP address (fixed from the Arduino IDE), taking the message "t" and sending it through the network stream to the specified IP.

To receive what NodeMCU wrote on the network, we created a byte variable named data and a string named ResponseData, so we can take both the bytes received and the string received from the IoT processor. The string is then split into 2 pieces with the comma separator, followed by validating the data and proceeding to the insert query to the database.

Speaking of server, Arduino IDE has some features which enables manual configuration of the Wi-Fi server, as follows:

```
char* ssid = "wifiSSID";
char* password =  "test123";

IPAddress ip(172,20,10,3);
IPAddress gateway(172,20,10,1);
IPAddress subnet(255,255,255,240);

WiFiServer wifiServer(8090);
```

Fig. 18. (Creating manual connection)

As we can see, we must provide the SSID of the Wi-Fi network and the password in plain text, which can be a security threat. Because the IDE is not supporting only the fixed IP,

we must set the gateway (local router) and the subnet, then we config the port we want by creating an object WiFiServer with int parameter.

```
WiFi.begin(ssid, password);
WiFi.config(ip, gateway, subnet);
Serial.print("Conectare la WiFi");
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println(".");
}
Serial.println("");
Serial.println("Conectat!");

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

Fig. 19. (Connecting to the Wi-Fi)

The Wi-Fi library is working very well with NodeMCU, so we can setup all the data before we turn on the connection.

```
WiFiClient client = wifiServer.available();

char status;
status = Serial.read();
if (status=='s')
{
  Serial.println(WiFi.localIP());
}

if (client) {
  while (client.connected()) {
    while (client.available()>0) {
      char c = client.read();
      if (c == 't')
      {
        temp = dht.readTemperature();
        umid = dht.readHumidity();
        String tempUmid=String(temp);
        tempUmid.concat(',');
        tempUmid.concat(umid);
        client.print(tempUmid);
      }
    }
  }
}
```

Fig. 20. (Sending data to the client)

Firstly, we need to make the server available via the loop function in the Arduino IDE. For debugging reasons, I created a variable named **status**, which will show me the IP address of the NodeMCU everytime I press the letter ‘s’.

The nested if statements if will validate the connection and will send data only when the client is ready to receive.

5. Conclusions

In this model, a project can be easily created and upgraded with better processors, multiple sensors, and, probably, a permanent solution by soldering the pins and creating a box to look like a commercial product. I used the TCP model because I wanted a good support from both Arduino and C#, plus getting confirmation on every packet sent into the network. This small project is capable of showing relevant data about the plant under observation, doing the routine tasks, the user being alerted only when the plant’s quality of life will be under or above the critical values.

As an improvement at the moment, Arduino Over The Air upload can be done with the Python scripts.

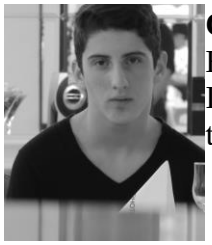
References

[1] LWIG Working Group. (2018). IETF. TCP Usage Guidance in the Internet of Things: https://tools.ietf.org/id/draft-ietf-lwig-tcp-constrained-node-networks-04.html

[2] Last Minute Engineers. Insight into ESP8266. Last Minute Engineers: https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/

[3] Adafruit. (2012). DHT11, DHT22. Adafruit: https://learn.adafruit.com/dht

[4] Sparkfun. How to use a Breadboard. Sparkfun: https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all



Cornel MIRCESCU is a student pursuing Master's Degree at Databases: Business Support at the Faculty of Cybernetics, Statistics and Economic Informatics from the Bucharest University of Economic Studies. He graduated the "Petrol-Gaze" University from Ploiesti, at Cybernetics specialization