BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

COLLECTION OF METEOROLOGICAL DATA USING THE MQTT PROTOCOL

SBĚR METEOROLOGICKÝCH DAT POMOCÍ PROTOKOLU MQTT

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Andrii Filippov**
AUTOR PRÁCE

SUPERVISOR — **doc. Ing. Ivo Lattenberg, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Andrii Filippov ***ID:*** 189830
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Sběr meteorologických dat pomocí protokolu MQTT

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a vyrobte dvě meteorologické stanice, které budou měřit teplotu a vlhkost. Zařízení budou napájené z baterie a změřené hodnoty budou pravidelně posílat přes Wi-Fi modul prostřednictvím MQTT protokolu do MQTT brokeru. Pro realizaci jedné stanice použijte modul s ESP8266, pro realizaci druhé pak modul ESP32. Napište jednoduchý program pro PC, který se přihlásí k odběru zpráv z MQTT brokeru a bude graficky zobrazovat časový průběh změřených hodnot z meteostanic.

DOPORUČENÁ LITERATURA:

[1] BRTNÍK, Bohumil a David MATOUŠEK. Mikroprocesorová technika: [práce s mikrokontroléry řady ATMEL AVR ATXmega A4]. Praha: BEN - technická literatura, 2011. ISBN 978-80-7300-406-4.

[2] MATOUŠEK, David. Práce s mikrokontroléry ATMEL. 2. vyd. Praha: BEN - technická literatura, 2006. μC & praxe. ISBN 80-7300-209-4.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

Abstrakt

Cílem této práce je sběr meteorologických dat pomocí protokolu MQTT ze dvouvytvořených meteorologických stanic založených na mikrokontrolérech typu ESP32 a ESP8266. Práce pojednává o textovému formátu JSON, jeho struktuře podle řady kritérií a jeho dalšímu využití při přenosu dat z meteorologické stanice. Další fází je studium a srovnání hlavních charakteristik mikrokontrolérů ESP32 a ESP8266. Na základě zadání byly pak vytvořeny autonomní meteorologické stanice. Dále byly porovnány existující MQTTservery pro sběr, ukládání a přenos meteorologických dat mezi klientem a serverem. Závěrečnou fází práce byla tvorba programu v jazyce C#, který generuje grafy na základě hodnot aktuální teploty a relativní vlhkosti přijatých ze serveru MQTT. Výzkumný projekt se skládá ze dvou autonomních meteorologických stanic, ze kterých jsou přenášena meteorologická data v reálném čase na server MQTT prostřednictvím bezdrátového připojení. Server pak dále poskytuje data pro aplikaci v počítači uživatele, kde program kreslí v reálném čase grafy na základě dat přijatých z MQTT serveru.

Klíčová slova

MQTT, JSON, ESP32, ESP8266, PCB, arduino

Abstract

The purpose of this thesis is collection of meteorological data using the MQTT protocol by creating two weather stations based on microcontrollers types ESP32 and ESP8266. And also their further operation. Another important stage of the study is devoted to the JSON text format and its structure according to a number of criteria, as well as its further use in creating a weather station.The next phase of the study is the search and comparison of the main characteristics of the ESP32 and ESP8266 microcontrollers based on the task of creating autonomous weather stations, as well as comparing actual MQTT servers for collecting, storing and transmitting meteorological change data from client to server and vice versa. The final phase of the study is to write a program in C # languege that will generate graphs based on temperature and relative humidity changes received from the MQTT server. The research product consists of two autonomous meteorological stations, which will transmit real-time data about meteorological changes to the MQTT server via wireless connection, while the server will transfer data to the user's computer where the program draws graphs based on data received from the MQTT server in real time.

Keywords

MQTT, JSON, C#, ESP32, ESP8266, PCB, arduino

ROZŠÍŘENÝ ABSTRAKT

Od starověku mělo počasí obrovský dopad na historii lidstva. Počasí dalo vítězství v bitvách, pomohlo vyhrát války, stavět státy a mělo klíčový vliv na mentalitu celých národů.

V dnešní době se lidé nenaučili podrobit počasí, ale dosáhli významného úspěchu v analýze a předpovídání meteorologických podmínek. Výpočetní technika se vyvinula z analogových na vysoce přesné digitální procesory, což mělo významný dopad na vytvoření moderního meteorologického centra. S pomocí moderních digitálních meteorologických stanic a vysokorychlostního internetu mají lidé na celém světě příležitost dostávat informace o změnách počasí online. Díky tomu se život lidí stává pohodlnějším, lepším a bezstarostnějším. Ale co kompaktní meteorologické stanice pro všeobecné použití? Můžeme například hlídat teplotní hladiny nebo máme požadavek na rozsah relativní vlhkosti, aby se zabránilo plísním a jiným nepříjemným problémům. V tomto případě je nutno zakoupit autonomní mini meteorologickou stanici s vysokou přesností. Většina společností na trhu však neposkytuje produkt požadované kvality.

Tato bakalářská práce nabízí návrh moderní meteorologické stanice využívající vysoce kvalitní hardware a pokročilé technologie pro přenos dat na internetu. Tato práce poskytuje srovnání a podrobnou analýzu klíčových charakteristik mikrokontrolérů ESP32 a ESP8266, jakož i snímačů teploty a relativní vlhkosti. Tyto prvky pak byly použity k vytvoření moderní autonomní meteorologické stanice. Dále byly analyzovány moderní protokoly pro přenos textových dat na internetu. Dalším důležitým aspektem práce je studium přenosového protokolu MQTT a jeho další implementace při tvorbě projektu přenosné meteorologické stanice. Velká pozornost je věnována textovému formátu JSON, protože právě tento formát bude využit k přenosu zpráv mezi koncovými uzly. V rámci vytvoření přenosné meteorologické stanice bylo navrženo provedení a byly vyrobeny desky s plošnými spoji, aby se meteorologická stanice stala kompaktnější. Pro návrh PCB byla vybrána platforma EasyEDA, která je analogem známé Eagle platformy, ale má srozumitelnější rozhraní, které umožňuje začátečníkům rychle se naučit základní funkce programu. Posledním teoretickým aspektem bylo srovnání hlavních charakteristik MQTT brokerů a výběr toho nejvhodnějšího, aby byly splněny požadavky na vysokorychlostní a rozsáhlý přenos dat z meteorologických

stanic pro další použití v uzavřeném systému Meteorologická stanice - MQTT broker - Koncový uživatel. Po závěrečné teoretické analýze následuje praktická část práce, která kombinuje analýzu a návrh koncepce připojení meteorologických stanic k brokeru MQTT pro další ukládání, zpracování a přenos dat o počasí do klientského programu napsaného pro potřeby práce v jazyce C#. Hlavními požadavky na program byla schopnost vytvářet grafy časových průběhů relativní vlhkosti a teploty v daném čase převzaté od MQTT brokeru a také možnost následného uložení těchto dat ve formátu .csv pro další statistické výpočty. Tato praktická část je také doprovázena podrobnými ilustracemi výsledků mého výzkumu a ukázkou fragmentů kódu používaných při programování mikrokontrolérů a tvorbě programu pro zobrazení grafů.

Díky této práci a výzkumu prováděnému v průběhu roku jsem získal praktické dovednosti při práci s deskami plošných spojů a mikrokontroléry různých typů. Tento projekt mi také pomohl upevnit moje znalosti v oblasti telekomunikací, programování a designu. Věřím, že díky mým zkušenostem a dovednostem v oblasti ovládání mikrokontrolérů jsem se stal všestrannějším specialistou v oblasti telekomunikačních technologií, což bude mít v budoucnu obrovský dopad na to, abych se stal profesionálem v této oblasti.

Bibliografická citace

FILIPPOV, ANDRII. Sběr meteorologických dat pomocí protokolu MQTT. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikace. Vedoucí práce doc. Ing. Ivo Lattenberg, Ph.D..

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Prohlášení autora o původnosti díla

Jméno a příjmení studenta: *Andrii Filippov*

VUT ID studenta: *189830*

Typ práce: *Bakalářská práce*

Akademický rok: 2019/20

Téma závěrečné práce: *Sběr meteorologických dat pomocí protokolu MQTT*

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: ……

Podpis autora

Poděkování

V této části bych rád poděkoval vedoucímu své bakalářské práce panu doc. Ing. Ivo Lattenbergu, Ph.D. za jeho rady, podporu a čas.

CONTENT

INTRODUCTION

With meteorology, as with the science of studying atmospheric phenomena, their properties, state and structure, humanity is familiar from time immemorial.

The word “meteorology” (ancient Greek. Μετεωρο-λογία – “reasoning about celestial phenomena”) is associated with the works of Plato and Aristotle. Meteorology as a science originated after the invention of the Galileo Galilei thermometer and Otto von Guericke’s barometer in the 17th century. Nowadays, people are faced with the measurement of meteorological indicators daily. Starting with a choice of winter clothes for a trip to the mountains and ending with the launch of a spacecraft from the cosmodrome. To measure the physical quantities associated with the weather use weather stations.

In the course of this bachelor’s thesis, the historical development of meteorology and weather stations will be studied in detail, the problems of modern weather stations, the problems of measuring meteorological indicators using analog and digital sensors and their errors will be investigated. The next step of the project will be to study the main components for building a modern weather station that meets all the requirements of the standards. The market for the main manufacturers of ESP32 and ESP8266 microcontrollers will be studied, and based on price, quality and functionality comparisons, boards will be selected for the basis of weather stations.

It is also necessary to study the main sensors measuring air temperature and relative humidity, compare them in terms of accuracy and reliability of measurements, as well as the price range of these products. This will be followed by an analysis of the transport protocol MQTT, as the main protocol for communication in the concept of Internet of Things. Based on the analysis, a text format will be selected for direct transmission of data on air temperature and relative air humidity using the MQTT transport protocol from the weather station to the broker using a wireless Wi-Fi connection.

Further, comparisons will be made of the main MQTT brokers, in order to select the most convenient and practical broker to work with the data obtained from the weather station. In the next paragraph of the project, the main development

environments for programming and compiling microcontrollers will be presented. The end point of the project will be the creation of a program in the C # programming language for building 2D graphs of changes in air temperature and relative humidity based on data obtained from the MQTT broker.

The end result will be a fully autonomous, customized connection of the weather stations via Wi-Fi to a broker based on the MQTT network protocol, which in turn will transmit data based on topic subscriptions to a computer program written in the C# programming language, which, based on the meteorological data obtained through the broker, will build graphs of temperature indicators and humidity indicators in a particular area. Recently, it has become a tradition to use the MQTT network protocol for such purposes, since most brokers support this particular protocol, and also because of its simplicity and clarity for the average user. In general, the main goal of this work is a detailed analysis of all the nuances associated with the MQTT protocol as well as its further implementation in the concept of the Internet of things (IoT), which is becoming more and more popular every day and in my opinion is the concept of the future of smart automation.

I chose the topic of the bachelor thesis because it is a new round in the development of smart automation and I believe that this topic has very great prospects for development in the future because the modern world is committed to the rational use of resources and the maximum efficiency of things improving these things with the help of artificial intelligence, thereby increasing their economic feasibility and practicality. Also I was always interested in software and hardware development and I am sure this topic will help me to improve my knowledge in these areas of science.

1 MODERN DIGITAL WEATHER STATION

Modern digital weather stations are portable devices that record weather and climate conditions using electronic sensors. They are equipped with an LCD display, which shows the temperature at a specific point in time, air humidity, pressure, and forecast for the near future. In this capture, we analyze the main types of modern weather stations as well as the main methods of measuring temperature as well as the main types of digital sensors, as well as consider and choose the best type of sensors for the successful completion of the tasks.

The structure of the digital meteorological installation, in addition to the barometer, thermometer, hygrometer, includes additional devices:

- anemometer and wind vane measuring wind direction and speed,
- rain gauge: determines the amount of precipitation,
- calendar, alarm clock, clock, etc.,
- USB - output for transferring data to a personal computer.

Advantages of digital weather stations:

- a significant set of functions for an adequate assessment of meteorological conditions and weather forecasting,
- remote controllers analyze weather data from various points to compile the most accurate picture,
- stylish and modern design,
- compact device.

Disadvantages of digital weather stations:

- required power source,
- incorrect installation of external sensors entails a false data transfer,
- the budget version of the device is not able to accurately display weather information.

1.1 Measurement of meteorological variables

The main meteorological variables used to determine the weather are temperature and humidity. Each variable is measured by different methods. For processing measured data meteorological stations are used. Below are brief descriptions of individual methods for measuring temperature and humidity using electronic sensors, some of which I used to create my own meteorological station.

1.1.1 Humidity

Humidity indicates how much water vapor contains a given amount of air. Relative humidity is the ratio between the instantaneous amount of water vapor in the air and the amount of water vapor that saturated air would have at the same pressure and temperature. To measure humidity we can use the following methods:

- weight method – It is accurate and consists in comparing the weight of air before and after drying,
- condensing method – Dew point temperature measurement by cooling the measuring surface, at the moment of drawing, the temperature on the surface is equal to the dew point temperature, then the humidity is determined,
- infrared light method – water vapor absorbs infrared radiation - the less it passes on the detector, the higher the humidity,
- hygroscopic method – Hygroscopic substances change their geometrical properties when absorbing moisture from the air (e.g., human hair),
- electric method:
    a) capacitance - Capacitor capacitance change, whose dielectric is made of special polymer, changes in moisture change the properties of the polymer,
    b) resistive - Uses changes in electrolytic conductivity.

The weather station measures humidity using a capacitive method. [1], [3]

1.1.2Temperature

Temperature is a scalar physical quantity, directly proportional to the kinetic energy of the particles. The greater the kinetic energy, the higher the temperature. Temperature has a direct influence on all living organisms of the planet Earth, therefore, it is one of the most important meteorological variables. Temperature is also an important weather indicator.The basic units of temperature measurement are Celsius (◦C), Kelvin (K) and Fahrenheits (F). Temperature is measured using thermometers, which can be divided into several types:

- expansion thermometers– devices based on the property of bodies to increase their volume when heated. (for example mercury or alcohol),
- electric thermometers:
  a) resistive temperature sensors – these devices are based on the temperature dependence of the electrical resistance of metals or semiconductors,
  b) resistance thermometers – the principle of action is based on the dependence of the electrical resistance of metals, alloys and semiconductor materials on temperature,
  c) thermoelectric thermometers – at different temperatures at the ends of the conductor at each end of the conductor there will be a different potential,
  d) PN temperature sensor – use the temperature dependence on the voltage at the PN transition in the permeable direction.
- color temperature indicators – determines the approximate surface temperature of the body. In contact with the surface of the body there is a chemical reaction and a change in the color of the indicator,
- contactless thermometers – They are based on capturing the electromagnetic radiation radiating the bodies.

The weather station measures temperature using a thermoelectric temperature sensor. [2], [3]

2 PARTS SELECTION

The key step in the implementation of my project is the selection of the main microcontrollers that will carry the bulk of the software. The next key device is a sensor for accurate measurement of meteorological indicators. These microcontrollers must be able to interact with other devices through some interface, use a wireless connection to a Wi-Fi network, and must also be able to interact with meteorological sensors. Therefore, the basic principles in choosing microcontrollers and sensors are the reliability of the devices, the accuracy of the measurements, as well as the ease of implementation in practice. It was also decided not to use Arduino microcontrollers in this project, since at the moment there are a large number of more cost-effective chinese counterparts that are not inferior to leading manufacturers as microcontrollers and contain a large number of detailed instructions and open source code.

2.1 Microcontroller comparison

This project involves the creation of two weather stations based on microcontrollers ESP32 and ESP8266. Therefore, a comparison of these models of microcontrollers will be carried out strictly within the framework of the functional advantages of these models of microcontrollers. The main emphasis in comparisons of these microcontrollers will be made on the following criteria: reliability, practicality, accuracy and ease of implementation. There are a large number of microcontrollers of these models; therefore, comparison of all models is inappropriate and time-consuming; therefore, in this comparison there are only microcontrollers selected by the author based on his subjective opinion.

2.1.1 ESP32 comparison

a) ESP32-DevKitC V4

ESP32-DevKitC V4 is a small ESP32 development board manufactured by Espressif. Almost all I / O pins are split into connectors on both sides for easy pairing with external devices. Users, if desired, have the ability to connect these contacts to other peripheral devices. Moreover, standard male headers simplify development, making it

more convenient and easy to use. [4]

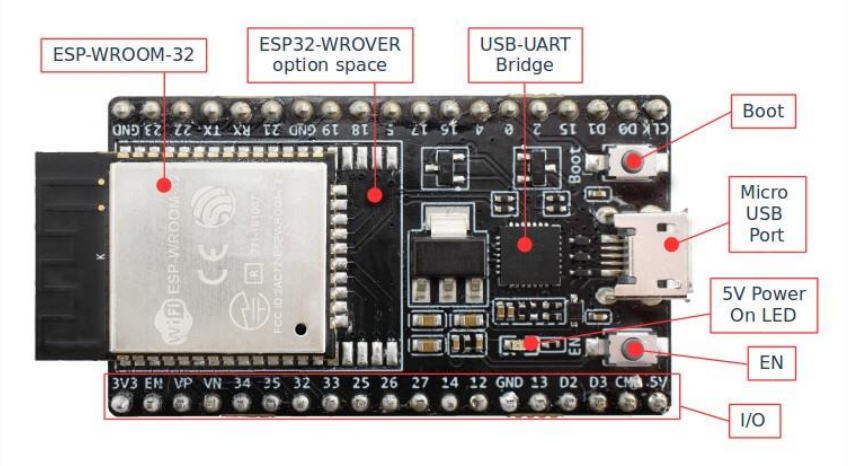


Figure 2.1: ESP32-DevKitC V4 with ESP32-WROOM-32 module soldered. [4]

Functional description:

- USB-UART bridge,
- operating voltage is5V,
- ESP32-WROOM-32 chip,
- digital input/output pins (DIO): 32,
- approximate price is 10$.

The board supports various ESP32 modules, including ESP32-WROOM-32 [4].

b) NodeMCU ESP-32S

NodeMCU ESP-32S is one of the newest and most successful development boards created by NodeMcu to implement the ESP-WROOM-32 module. Its main component is the ESP32 microcontroller, which among other functions also supports Wi-Fi, Bluetooth, Ethernet and low power technologies in one chip. [5], [6]

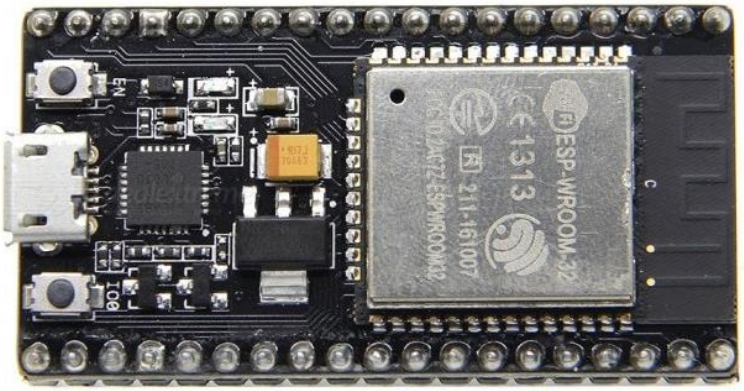


Figure 2.2: Node MCU ESP-32S with ESP32-WROOM-32 module soldered.[5]

Functional description:

- Wi-Fi: IEEE 802.11 b/g/n/e/I,
- flash memory: 4 MB,
- operating voltage is 3.3V,
- operating temperature: -40 to +125 (°C),
- network protocols: IPv4, IPv6, SSL, TCP / UDP / FTP / HTTP / MQTT,
- ESP32-WROOM-32 chip,
- digital input/output pins (DIO): 28,
- approximate price is9$.

Conclusion

In general, these microcontrollers do not have big differences. The cost and quality of each module are also on the same level. Both microcontrollers have enough power and pins to achieve the objectives of this project. For the project, I decided to choose the Node MCU ESP-32S, because there are more open sources and instructions (Connect, Register, Virtualize and Program) than for the ESP32-DevKitC V4.

2.1.2 ESP8266 comparison

a) NODEMCU Lua IoT ESP8266 WifiController Board v3

NodeMCU is a small-size board, based on the cheap Wi-Fi ESP-12E module, containing a single-chipESP8266 Wi-Fi SoC.NodeMCUas a whole is a set of the most successful open source firmware and development tools, which undoubtedly provides the ability to prototype your IoT in several lines of the Lua script. The ESP-12E chip is considered one of the most advanced and most affordable modules based on the ESP8266 module; This module is becoming increasingly popular thanks to the built-in antenna and the screen of the Wi-Fi circuit board. [7]

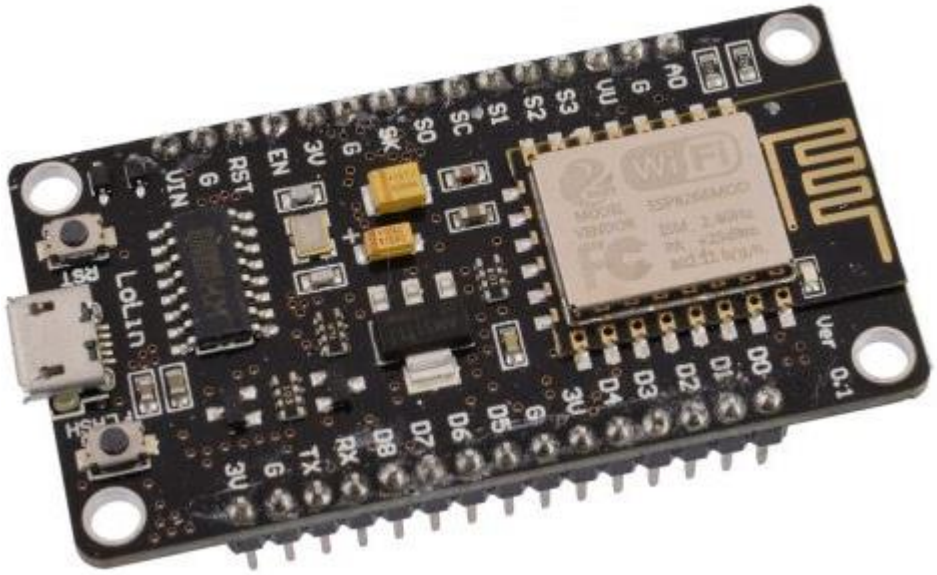


Figure 2.3: NODEMCU Lua IoT ESP8266 Wifi Controller Board v3.[7]

Functional description:

- include open-source, interactive and programmable,
- antenna,
- Wi-Fi: IEEE 802.11 b/g/n,
- digital input/output pins (DIO): 30,
- approximate price is6$.

The Development Kit based on ESP8266, integrates GPIO, PWM, IIC, 1-Wire and ADC all in one board. Power the development in the fastest way combination with NodeMCU Firmware.

b) Witty Cloud / GizWits - ESP8266

The Witty Cloud (also named GizWits) development board is very interesting in the fact that it consists by design of two separate modules, with each module having a USB connector. Once programmed, only the upper part may be used fully functionally as programmed by yourself.

Figure 2.4: Witty Cloud development board.

If you tear those modules apart you end up with - the upper module the upper module

Figure 2.5: Upper Witty Cloud module.

holding on the upper side:

- photo resistor,
- antenna,
- multi-color LED,
- ESP-12 module,
- digital input/output pins (DIO): 12.

the lower module with components on the upper side only

Figure 2.6: Lower Witty Cloud module.

holding on the lower side:

- voltage regulator,
- micro USB port,
- with its only function to accept a power source,
- the pins to plug into a breadboard.

In general, the module contains:

- two buttons (the left one to set the system into flash mode, the right one to reset the system),
- micro USB port accepting a power source as well as data exchange with another device,
- two female header connectors to connect with the second Witty Cloud module if needed.

Conclusion

Specifications and price of these microcontrollers are about the same. Both microcontrollers have enough power and pins to achieve the objectives of this project. The most interesting thing in the design of Witty Cloud / GizWits - ESP8266 module is that it consists of two separate boards: the controller boards itself and the auxiliary one used for firmware and communication with the computer. Firmware included. Moreover, the board comes with a CH40 micro USB module to program it. But on the other hand a lot of small details make this microcontroller less reliable and overly complex. In my opinion the NODEMCU Lua IoT ESP8266 Wifi Controller Board v3 is more comfortable to work with, than Witty Cloud / GizWits - ESP8266.

2.2 Sensor comparison

Weather stations to measure atmospheric parameters use high-precision electronic sensors. The most common temperature and humidity sensors are DHT11 and DHT22/AM2302 which is a modification of the sensor DHT22. They are famous for their reliability and measurement accuracy. Below is a detailed comparison of these sensors. There are a large number of sensors; therefore, comparison of all models is inappropriate and time-consuming; therefore, in this comparison there are only sensors selected by the author based on his subjective opinion.

2.2.1 DHT11

DHT11 is a digital temperature and humidity sensor from the category of combined sensors. DHT11 contains a calibrated digital signal with indicators of temperature and humidity of a sufficiently high reliability and accuracy of measurements. The sensor also has resistance to wet components and an NTC temperature measuring device and is connected to a high-performance 8-bit microcontroller. [8]

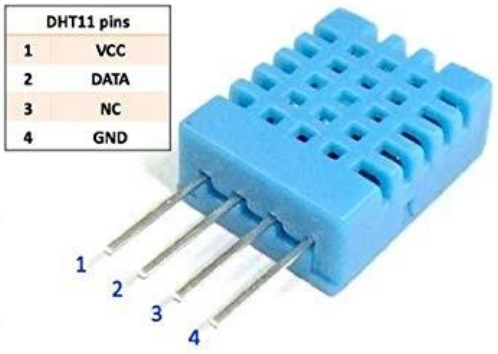


Figure 2.7: DHT11 with pins description (from left to right).[8]

Functional description:

- operating voltage is 3.3– 5.5(V),
- maximum current using during conversion = 0.0025(A),
- perfect for 20-80% humidity readings with 5% accuracy,
- perfect for 0-50°C temperature readings ±2 (°C) accuracy,
- sampling rate is no more than 1 Hz (one per second),

- regular transmission distanceis 10 miters,
- approximate price is5$.

This sensor is quite popular among developers of IoT, however, for the use in weather stations of this project, the accuracy of these devices is not enough to achieve the goals. Also, the operating temperature accuracy of the sensor does not allow this sensor to be used in extremely low temperatures, which makes it impossible to measure in the winter season.

2.2.2 DHT22(AM2302)

This model of sensors uses a special modern technology for collecting and processing a digital signal and moisture detection technology, providing reliability, stability and accuracy of measurements regardless of weather conditions. Sensitive sensor elements are connected to an 8-bit single-chip computer. Most sensors of this model provide temperature compensation and calibration in the calibration chamber, and the calibration factor is stored in the type of program in the OTP memory, since the sensor detects this, it will indicate the coefficient from the memory. [9]

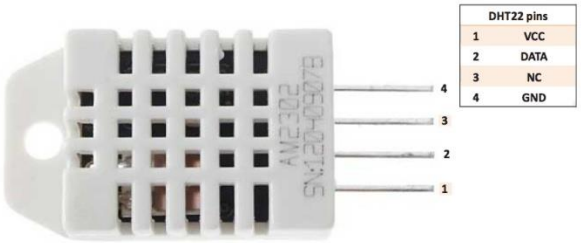


Figure 2.8: DHT22(AM2302) with pins description (from top to down). [9]

Functional description:

- operating voltage is3 – 6 (V),
- max current use during conversion is 0.0025A,
- perfect for 0-100% humidity readings with 2-5% accuracy,
- perfect for -40 to 80°C temperature readings ±0.5(°C) accuracy,
- maximum 0.5 Hz sampling rate (once every 2 seconds),
- regular transmission distance is 20m,

- approximate price is7$.

Conclusion

Having analyzed in detail the technical characteristics of these temperature and humidity measurement sensors, and summing up all the criteria that we can see, we make the following conclusion - the DHT11 temperature and humidity measurement sensor, in comparison with the DHT22 (AM2302) temperature and humidity measurement sensor, is less accurate and less reliable in their measurements. Moreover, the DHT11 sensor works in a smaller range of temperature and humidity data, since this sensor measures temperature in the range from 0 ° C to 50 ° C, which makes it impossible to use it outside the building, which significantly reduces its chances in the struggle for use in my project. Moreover, the accuracy of the relative humidity of the DHT11 sensor is ± 5% relative humidity, which is a noticeable difference compared to the DHT22 (AM2302) ± 2% relative humidity sensor. However, both sensors do not require any additional components, such as a resistor. It is already inside the DHT22 sensor (AM2302) and the DHT11 sensor, but in the framework of this project, for personal safety reasons, as well as to reduce the risk of overload and further failure of the sensor, an additional 10KoM resistor is required. Based on the information above, I conclude that as part of my project it will be more profitable to use a temperature and humidity sensor DHT22 (AM2302) to achieve the goals of maximum accuracy and reliability of my weather stations.

3 GENESIS OF THE MQTT PROTOCOL

"The Internet of Things (IoT) is a network of physical devices, household appliances, vehicles, and many other items equipped with smart electronics, sensors, software, drives, and communication tools that allow these devices to connect, collect, and exchange data with other devices and servers via the Internet.

The Queuing Telemetry Transport (MQTT) protocol was created in 1999 by Andy Stanford (IBM) and Arlen Nipper (Eurotech) for pipeline management in the oil and gas industries. Engineers tried to find an easy and reliable protocol for scale production, however, due to its simplicity, efficiency and low hardware demands, it soon became used in home automation equipment.

Due to the complicated management in the oil industry engineers decided to develop a light protocol with basic options, including account confines for narrow nodes, unreliable connection to the global WAN network and bandwidth limiting with variable delays. This was the main to choose a client/server structure and publishing / subscribing based on the TCP / IP architecture. In 2013 it was certified international OASIS standardization company as an open source protocol suitable for communication between devices." [10]

3.1 Message Queuing Telemetry Transport Protocol

"Engineers research led to the development and implementation of the Message Queuing Telemetric Transport Protocol (MQTT).

Message Queuing Telemetry Transport (MQTT) is an ISO standard publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited." [10]

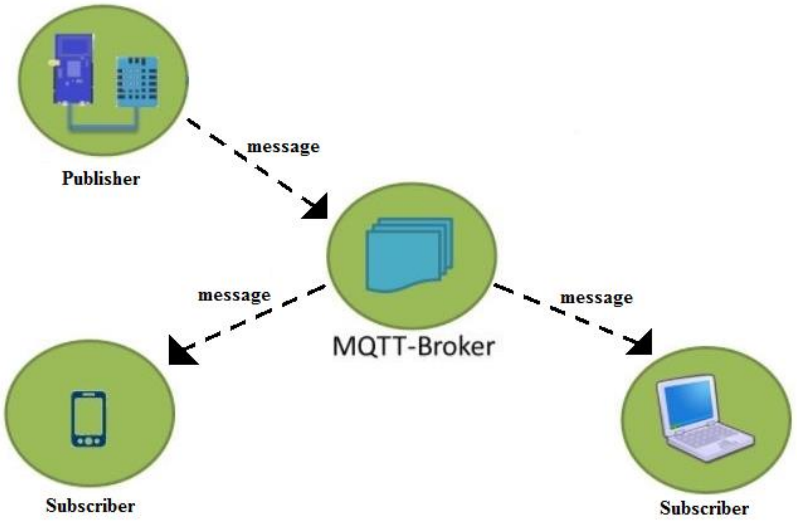


Figure 3.1: MQTT Publish/Subscribe Framework

"An MQTT client can act as a publisher to send data to an MQTT server acting as an MQTT message broker. The MQTT server receives a network connection along withapplication messages, such as Temp / RH data, from publishers. Moreover, it handles the subscription and cancellation process and sends application data to the MQTT clients acting as subscribers. The application at the bottom of the picture is an MQTT client, which is a subscriber to the Temp / RH data created by the publisher or the sensor at the top. This model, in which subscribers want to receive information from the publisher, is known to many." [10]

"The MQTT client can optionally subscribe to all data or specific data from the publisher's data tree. Moreover, the presence of a message broker in MQTT separates the transfer of data between customers acting as publishers and subscribers. In fact, publishers and subscribers do not even know about each other. The advantage of this decoupling is that the MQTT message broker ensures that information can be buffered and cached in case of network failures. It also means that publishers and subscribers do not have to be online at the same time. MQTT control packets pass through TCP transport using port 1883. TCP provides an ordered lossless stream between the MQTT client and the MQTT server." [10]

3.1.1 Connecting clients to the MQTT broker

Individual clients connect via TCP to the MQTT broker, most often through port 1883. In the case of an encrypted TLS connection, port 8883 is used. When connecting, the client sends a CONNECT message, usually with a "clean session" flag, which ensures that all topics that may have been previously preset are unsubscribed. When the client successfully connects, the broker acknowledges the client's connection using a CONACK message. After successfully connecting to the broker, the client can subscribe to certain topics using the SUBSCRIBE message. The broker confirms the successful setting of the topic subscription with a SUBACK message. Likewise, the client can unsubscribe from topics using the UNSUBSCRIBE message. In this case, the broker also sends a UNSUBACK confirmation message to the client. To determine if the client is still active, if it does not generate any messages, it sends PINGREQ messages at regular intervals, which the broker confirms with a PINGACK message.

3.1.2 MQTT Message Types

Fourteen different types of control packets are specified in MQTT version 3.1.1. Each of them has a unique value that is coded into the Message Type field. MQTT message types are summarized in table below.

Table 3.1: MQTT Message Types. [10]

Control packet	Direction of flow	Description
CONNECT	Client to Server	Client request to connect to Server
CONNACK	Server to Client	Connect acknowledgment
PUBLISH	Client to Server or Server to Client	Publish message
PUBACK	Client to Server or Server to Client	Publish acknowledgment
PUBREC	Client to Server or Server to Client	Publish received
PUBREL	Client to Server or Server to Client	Publish release
PUBCOMP	Client to Server or Server to Client	Publish complete
SUBSCRIBE	Client to Server	Client subscribe request
SUBACK	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	Client to Server	Unsubscribe request
UNSUBACK	Server to Client	Unsubscribe acknowledgment
PINGREQ	Client to Server	Ping request
PINGRESP	Server to Client	Ping response
DISCONNECT	Client to Server	Client is disconnecting

3.1.3MQTT Message Format

MQTT is a light protocol because each control packet consists of a 2-byte fixed with optional variable header fields and optional payload.

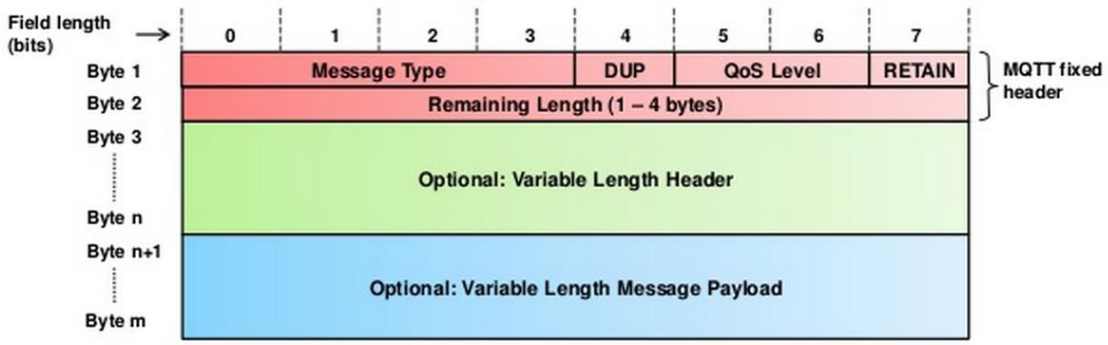


Figure 3.2: MQTT Message Format.[10]

MQTT contains a small-size header of 2 bytes .The first MQTT field in the header is Massage Type, which identifies the kind of MQTT packet within a message.

Conclusion

The protocol is quite reliable due to its simplicity and clear structure. The lightweight and good architecture of the protocol make it a quit fast and useful to work with the memory limited microcontrollers like ESP32 or EPS8266. The main advantage of the MQTT protocol is that for communication and data acquisition it is not necessary to know the IP address of individual clients, but only the IP address of the MQTT broker. Forwarding of messages between individual clients can be solved centrally using a script that connects to the broker as a client. This is more user-friendly than defining to each client what messages to receive from other clients and how to handle them. Especially when clients change. All current data is stored in one place. In general, the protocol is ideal for the implementation of my project. Moreover, many brokers support this protocol, which is a big plus in the implementation of IoT projects of any complexity.

4 MQTT BROKER SELECTION

For the quality of my weather station, you need to find a suitable message broker. The main goal of the broker is to receive and give messages. You can think of it as a post office: when you drop a letter in the box, you can be sure that the postman will deliver it to the addressee sooner or later. In this analogy, the brokers presented in the paragraph below works like a mailbox, a post office, and a postman at the same time. At present, there are many brokers of various parameters and functions, with the help of which it is possible to implement even the most specific projects. However, brokers in this comparison were selected according to the subjective opinion of the author.

4.1 Mosquitto

Mosquito is an open source message broker (licensed with EPL / EDL) that implements the MQTT protocol versions 3.1 and 3.1.1 and supports the proposed MQTT v5, which makes scalability and portability improvements. Mosquitto is lightweight and suitable for use on all devices, from low-power computers to single-board computers to full servers. [11]

The MQTT protocol provides an easy and fast messaging method using the publish/subscribe model. Thanks to this method, the protocol becomes suitable for messaging over the Internet, for example, with low power sensors or mobile devices such as telephones, embedded computers or microcontrollers. [11]

The Mosquitto project also provides the C library for implementing MQTT clients, as well as the very popular MQTT clients mocsquitt__pub and mosquittto_sub. The Mosquittto broker is part of the Eclipse IoT Working group, “industry collaboration companies that invest and promote open source for IoT”. [11]

Advantages:

- easy to setup,
- useful interface,
- verified clients Pub/Sub,

- easy integration with ESP32 and ESP8266 microcontrollers.

Conclusion:

Mosquitto is the most common, really light MQTT broker written in C. For Windows and Mac there are official versions, source codes are also available there. You can also get source codes from GitHub.

4.2 VerneMQ

VerneMQ is primarily an MQTT message / signature broker whose main protocol is the MQTT protocol of the OASIS industry standard. Moreover, VerneMQ was created to accept messaging applications and IoT applications to the next level, offering a unique set of features related to reliability, simplicity, scalability, and high performance. To perform all tasks assigned to the broker, VerneMQ is designed from the ground up to act as a distributed message broker, ensuring continuous operation in the event of network or node failures and simple horizontal scalability. Fundamental technology is a proven technology stack of telecommunications technology that provides a solid foundation for systems that must operate around the clock. It can also effectively use all available resources as a basis for simple vertical scalability.[12]

VerneMQ uses clustering technology without skill. There are no particular nodes, such as wizards or slave devices that need to be considered when inevitable changes in infrastructure or maintenance windows require adding or removing nodes. This makes the cluster easy and safe. [12]

Advantages:

- MQTT protocol support,
- durable subscriptions on topics,
- TLS Support and Certificate based Authentication,
- detailed instructions for installation and use,
- flexible plug in and integration options,
- VerneMQ is 100% open and free to commercial modification and re-use.

Conclusion:

VerneMQ is a relatively new broker MQTT written in the Erlang programming language. The Erlang language is quite popular in the world of brokerage messages, as its soft and distributed real-time capabilities. VerneMQ contains many advantages that areinherent in the best brokers. Full implementation of MQTT 3.1.1, anti-aliasing, TLS 1.2, and authorization with the database. Can be easily extended with Lua scripts, has a useful GitHub development team, and pretty good documentation.

4.3RabbitMQ

RabbitMQ is an open source message broker with a wide range of functions. RabbitMQ was originally designed to implement the Advanced Message Queuing Protocol (AMQP) and was expanded over time using a plug-in architecture to support Message Queuing Telemetric Transport (MQTT), Streaming Text-Oriented Messaging Protocol (STOMP) and other protocols.[13]

The broker's server program is based on the Erlang programming language and is built on the Open Telecom Platform platform for disaster recovery and clustering.

For interaction with the broker, client libraries are available for all leading programming languages. [13]

Advantages:

- supports multiple messaging protocols,
- pluggable authentication & authorization,
- supports TLS,
- lightweight and easy to deploy in public and private clouds,
- multiple exchange type.

Conclusion:

RabbitMQ is a very common message broker designed on the platform “Erlang” that has support for MQTT among other protocols through a plugin. RabbitMQ includes TLS support, clustering is good, authorization cannot be done using a database directly because of that reason you should create an HTTP REST wrapper over your database

and use it as an authorization backend. The disadvantage of RabbitMQ is that the MQTT support itself. RabbitMQ supports the AMQP protocol natively, the MQTT implementation is missing some important features such as Quality of Service (level 2). QoS 2 guarantees that a message is received exactly once. This is important in some cases, for example when commands are sent from the IoT platform to the devices. Lost or duplicate commands can make some problems, which will have a bad effect on these scenarios, so QoS2is a must.

4.4 CloudMQTT

CloudMQTT are managed Mosquitto servers in the cloud. CloudMQTT let you focus on the application instead of spending time on scaling the broker or patching the platform.The main advantage of this service is that it is not necessary to configure the Mosquitto software and own the platform on which it would run. This type of MQTT cloud broker was chosen to avoid errors caused by wrong setup of Mosquitto software.

Advantages:

- MQTT protocol support
- TLS Support
- Certificate based Authentication
- 5 available connections

Conclusion

I chose CloudMQTT for my project, as this cloud broker has enough number of clients-brokers connections and has extensive documentation. For my purpose, a free Mosquitto instance called “Cute Cat” was selected, allowing five clients to connect at a rate of 10 Kbit / s. Moreover, there are also many forums and video tutorials dedicated toCloudMQTT, which will greatly help me in the future use of this cloud broker during my project. This cloud broker is characterized by its simplicity and ease of use. After registration, the user selects and creates the Mosquitto object. After opening the Mosquitto object, the data necessary to connect clients to the created MQTT broker is generated and displayed.

For a full cloud broker connection with the client, we need the following values:

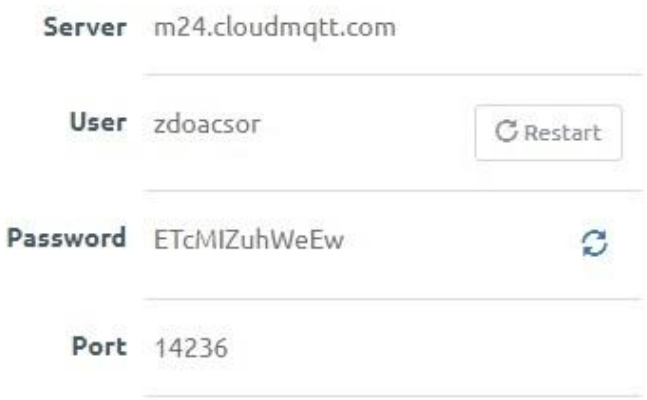


Figure 4.1: CloudMQTT profile information

Server: m24.cloudmqtt.com
User:zdoacsor
Password:ETcMIZuhWeEw
Port: 14236

We will use these values both in the microcontroller's code and in the program code for plotting graphs.

```
const char* Mosquitto_Server = "m24.cloudmqtt.com";
const int Mosquitto_port = 14236;
const char* user_mqtt = "zdoacsor";
const char* pass_mqtt = "ETcMIZuhWeEw";
```

Figure 4.2: CloudMQTT profile information using in Arduino IDE code

CloudMQTT data is exchanged with customers by subscribing to certain topics. In thiscase, my weather stations send data on temperature and relative humidity of air under one general topic: esp. CloudMQTT receives data tagged with an "esp" topic, keeps the information in memory and sends the topic data to users who subscribe to this newsletter. This client supports all text formats of the MQTT transport protocol, including JSON.

Data in JSON format comes to CloudMQTT in the following form:

Figure 4.3: CloudMQTT data receive

5 JAVASCRIPT OBJECT NOTATION

JavaScript Object Notation (JSON) is a data transfer format. As you can see, JSON is a JavaScript based text interchange format, but it is available for use in many languages, including Python, Ruby, PHP and Java.Like other text formats, JSON is easy to read by people. JSON itself uses the .json extension. When it is defined in other file formats, like .html, it appears in quotes as a JSON string or it can be an object assigned to a variable. This format is easy to transfer between the server and the client or browser. [10]

5.1 Syntax and structure

A JSON object is a data format with a key :value, which is usually rendered in curly braces. When you work with JSON, you most likely see JSON objects in the .json file, but they can also exist as a JSON object or string in the context of the program.[10]

This is what a JSON object looks like:

```
{
 "weather_station" : "1",
 "model" : "ESP32",
 "temperature" : Tep,
 "humidity" : Vlh,
 "online" : true
}
```

Figure 5.1: JSON object

Although this is a short example, and JSON could be much longer, it shows that this format is basically set with two curly braces that look like { }, and data with key values are between them. Most of the data used in JSON is in JSON objects. [10]

Pairs of key values have a colon between themselves, such as here the “key”: “value”. Each pair of values is separated by a colon, so the middle of JSON looks like this: “key”: “value”, “key”: “value”, “key”: “value”. In our example above, the first pair of key values is “weather_station”: “1”.

JSON keys on the left side of the colon. They need to be wrapped in parentheses, as with a “key” it can be any string. In each object, the keys must be unique. Such key strings may contain spaces, as in “weather station”, but such an approach may

complicate access to them during the development process, so the best option in such cases would be to use underscores, like here “weather_station”.

JSON values are on the right side of the colon. To be precise, they can be one of six data types: string, number, object, array, boolean, or null value. At a broader level, values can also consist of complex data types, such as a JSON object or an array. Each data type that is passed as a value in JSON will support its own syntax, so the strings will be in quotes, but the numbers will not. Although in the .json files we usually see the format of several lines, JSON can also be written in one continuous line:

```
{ "weather_station" : "1", "model" : "ESP32", "online" : true, }
```

Figure 5.2: JSON object string type

Working with JSON in a multi-line format often makes it more readable, especially when you are trying to handle a large dataset. Since JSON ignores spaces between its elements, you can separate them with the same spaces to make the data more readable:

```
{
  "weather_station" : "1",
  "model" : "ESP32",
  "online" : true
}
```

Figure 5.3: JSON object column type

It is very important to remember that although they are visually similar, but JSON objects do not have the same format as JavaScript objects, so although you can use functions inside JavaScript objects, you cannot use them as values in JSON. The most important feature of JSON is that it can easily be transferred between programming languages in a format that almost all languages understand. JavaScript objects can only work directly through the JavaScript language. [10]

Conclusion

JSON is a neutral format for use and it has many implementations for use in many programming languages. JSON is a lightweight format that allows you to easily share, store and work with data. As a format, JSON is experiencing growing API support,

including the Twitter API. I will not create my own .json files, but I will get them from other sources, it becomes very important to think less about JSON structure and more about how to better use it in my work. As planned, the program should connect to the Wi-Fi network, take readings of the sensor about the current temperature and relative air humidity, transform the received data into text JSON format and send them to the broker MQTT.

Below is a part of the code, which shows a visual transformation of data into JSON format and publication to MQTT broker.

Example:

```
const int jsonSize = JSON_OBJECT_SIZE(2);
StaticJsonDocument<jsonSize> jsonResult;
float Tep = Sensor.readTemperature();
float Vlh = Sensor.readHumidity();
jsonResult["temperature"] = Tep;
jsonResult["humidity"] = Vlh;
char toPublic[128];
serializeJson(jsonResult,toPublic);
client.publish("esp", toPublic);
```

Figure 5.4: publisher to MQTT broker

6 MICROCONTROLLERS & SENSOR CONNECTION

The most important point of this project is proper connection implementation. I have to make two connections for microcontrollers with sensors. Microcontroller's documentation (Pin Mapping) and visual connection schemas are presented in the following paragraphs. To ensure compactness and maximum autonomy, I decided to design and create special printed circuit boards (PCB) to replace the universal solder field.

6.1 EasyEDA

To create printed circuit boards, I used the EasyEDA program, which is an analogue of the famous Eagle. The program is easy to use and suitable for beginners.

EasyEDA is a web-based EDA tool suite that enables hardware engineers to design, simulate, share - publicly and privately - and discuss schematics, simulations and printed circuit boards. EasyEDAallows creation and editing of printed circuit board layouts and, optionally, the manufacture of printed circuit boards.Registered users can download Gerber files from the tool free of charge but for a fee, also EasyEDA offers a PCB fabrication service. This service is also able to accept Gerber file inputs from third party tools.In the schematic drawings below, you can see the connection diagram for an autonomous weather stations based on the ESP32 and ESP8266 microcontrollers.

6.1.1 NodeMCU ESP-32S connection with DHT22/AM2302 sensor

The DHT22 / AM2302 digital sensor will be connected to the first ESP32S module. The sensor will be connected to the microcontroller, as shown in Figure 6.2 from the official documentation of the sensor and the microcontroller. The first GND pin (Earth) will be connected to the ground of the NodeMCU ESP-32S module, the second Vcc pin will be connected to a 3.3V power pin, and the last DATA pin will be connected to the module pin GPIO16, which provides data transfer. VIN pin is connected to a power supply unit consisting of 3 AA batteries of 1.5V each, forming a total voltage of 4.5V, which is sufficient and safe for the operation of weather stations as well as long-term autonomy. For the project of the weather station, an ESP32S

controller based on the WROOM-32 processor was selected, as well as a temperature and humidity sensor DHT22 (AM2302) with high measurement accuracy. To avoid possible overload and further failure of the DHT22(AM2302) digital sensor, a 10kΩ resistor was connected between the data pin and the power pin. This is not a prerequisite for the operation of the station, but only the prevention of a possible problem.

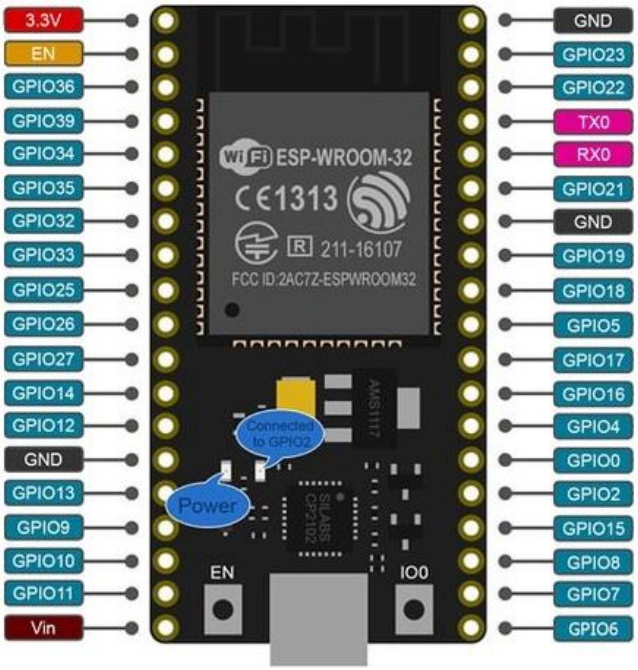


Figure 6.1: NodeMCU ESP-32S pinout.[5]

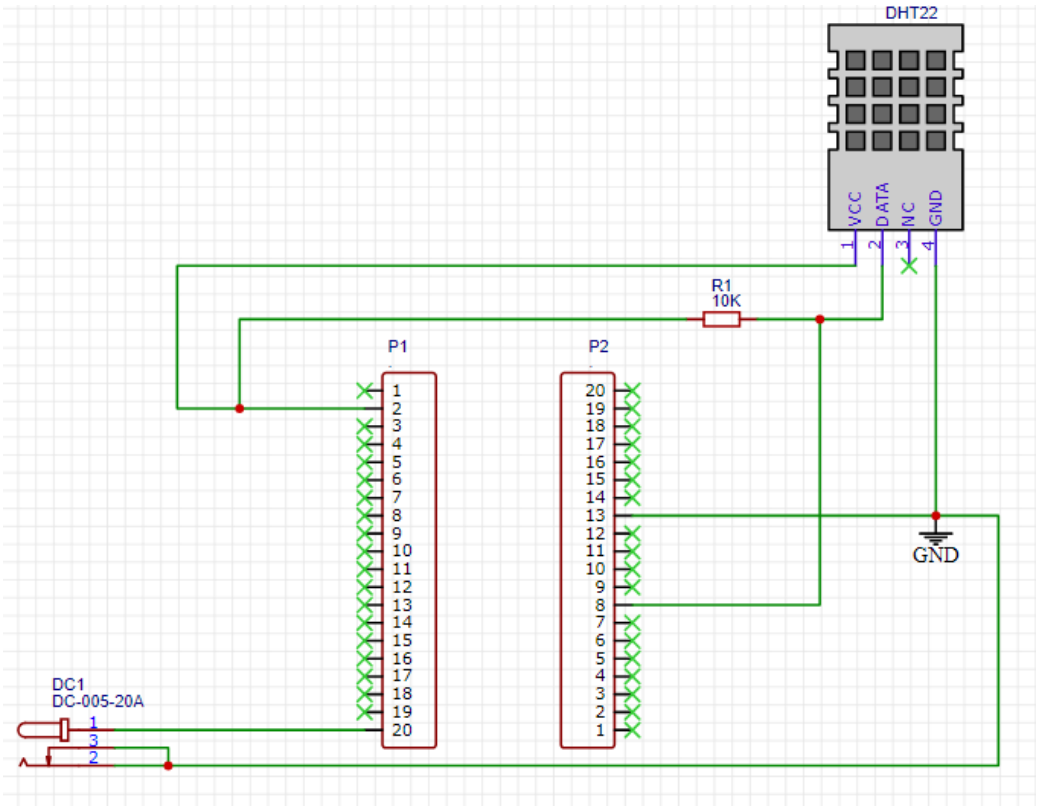


Figure 6.2: NodeMCU ESP-32S connection with DHT22(AM2302) sensor

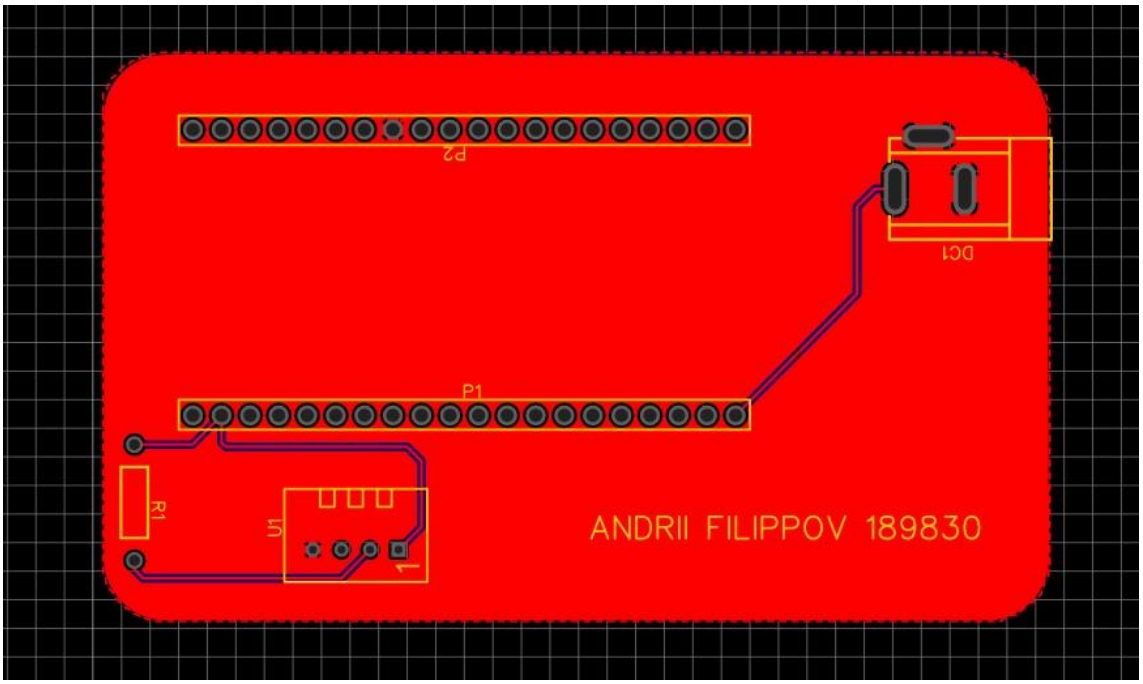


Figure 6.3: NodeMCU ESP-32S converted to PCB format (Top Layer)

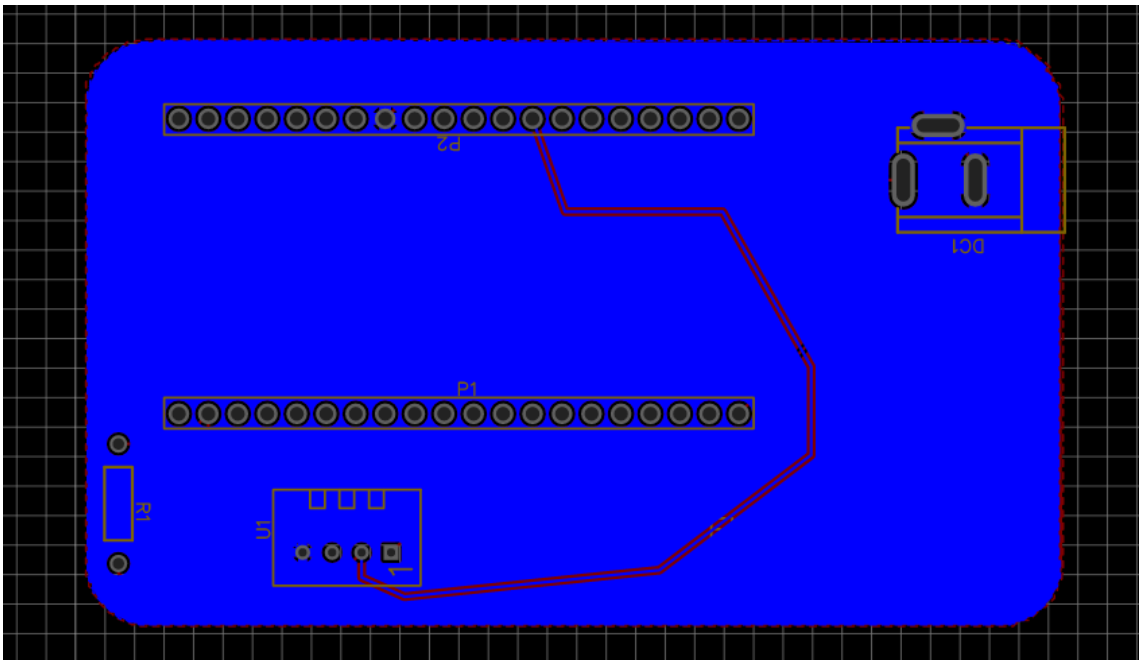


Figure 6.4: NodeMCU ESP-32S converted to PCB format (Bottom Layer)

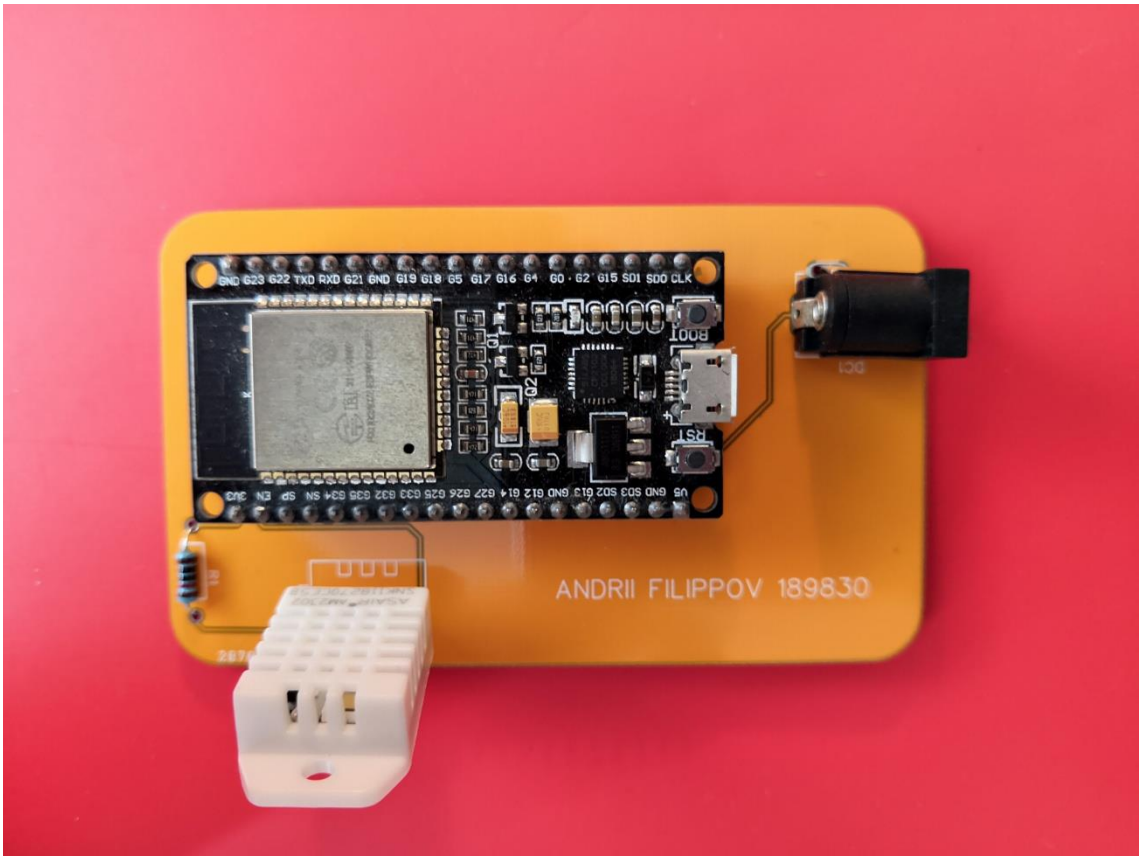


Figure 6.5: NodeMCU ESP-32S complete PCB

6.1.2 NODEMCU Lua IoT ESP8266 Wifi Controller Board v3 connection with DHT22/AM2302 sensor

In the schematic drawings below, you can see the connection diagram for an autonomous weather station based on the NODEMCU Lua IoT ESP8266 Wifi Controller Board v3 microcontroller. The DHT22 / AM2302 digital sensor will be connected to the second ESP8266 module. The sensor will be connected to the microcontroller, as shown in Figure 6.7 from the official documentation of the sensor and the microcontroller. The first GND pin (Earth) will be connected to the ground of the NodeMCU ESP-8266 module, the second Vcc pin will be connected to a 3.3V power pin, and the last DATA pin will be connected to the module pin GPIO4, which provides data transfer. VIN pin is connected to a power supply unit consisting of 3 AA batteries of 1.5V each, forming a total voltage of 4.5V, which is sufficient and safe for the operation of weather stations as well as long-term autonomy. For the project of the weather station, an ESP8266 controller based on theESP-12E processor was selected, as well as a temperature and humidity sensor DHT22 (AM2302) with high measurement accuracy. To avoid possible overload and further failure of the DHT22(AM2302) digital sensor, a 10 kΩ resistor was connected between the data pin and the power pin. This is not a prerequisite for the operation of the station, but only the prevention of a possible problem.

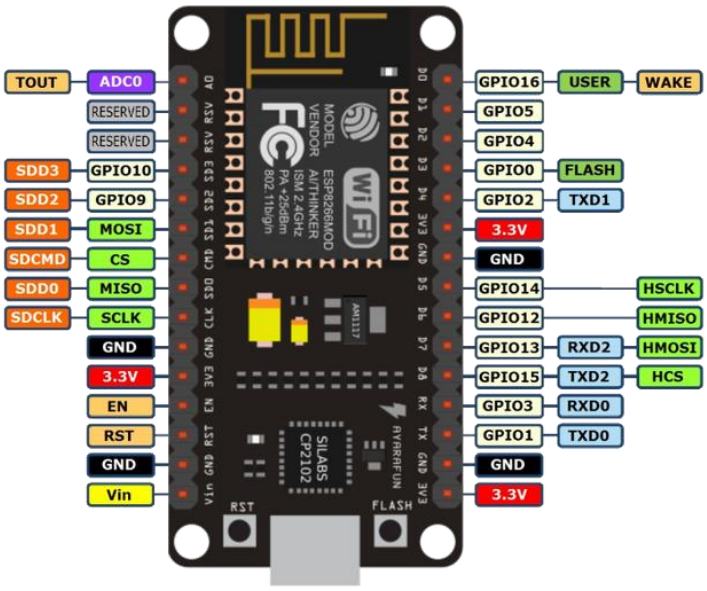


Figure 6.6: NODEMCU Lua ESP8266 v3 pinout.[7]

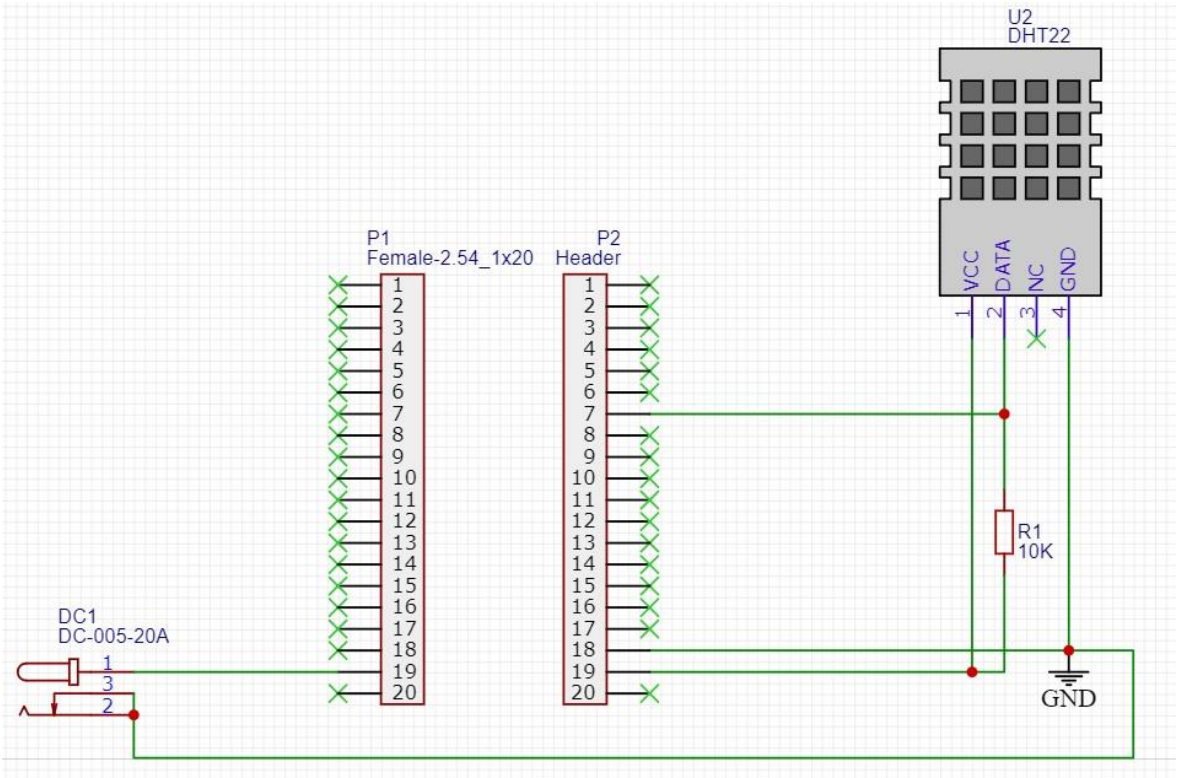


Figure 6.7: NODEMCU Lua ESP8266 v3 connection with DHT22(AM2302) sensor

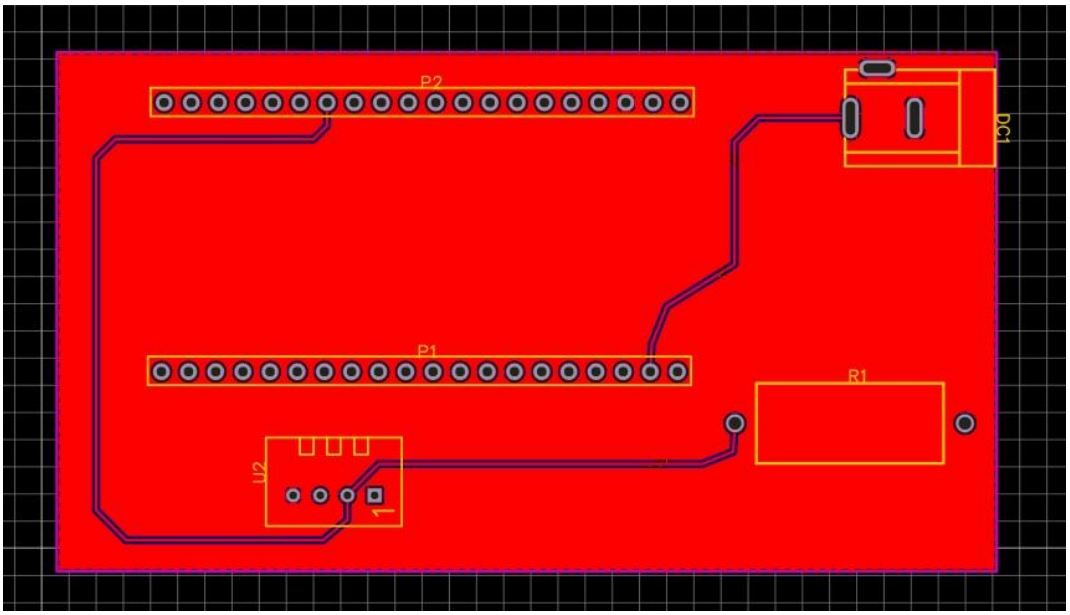


Figure 6.8: NODEMCU Lua ESP8266 v3 converted to PCB format (Top Layer)

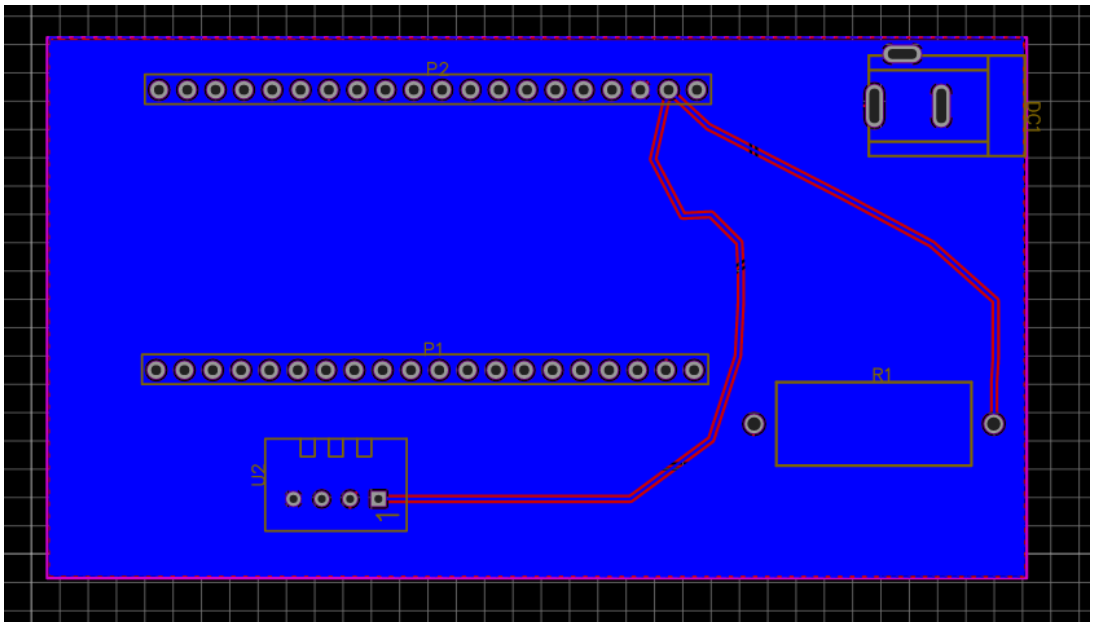


Figure 6.9: NODEMCU Lua ESP8266 v3 converted to PCB format (Bottom Layer)

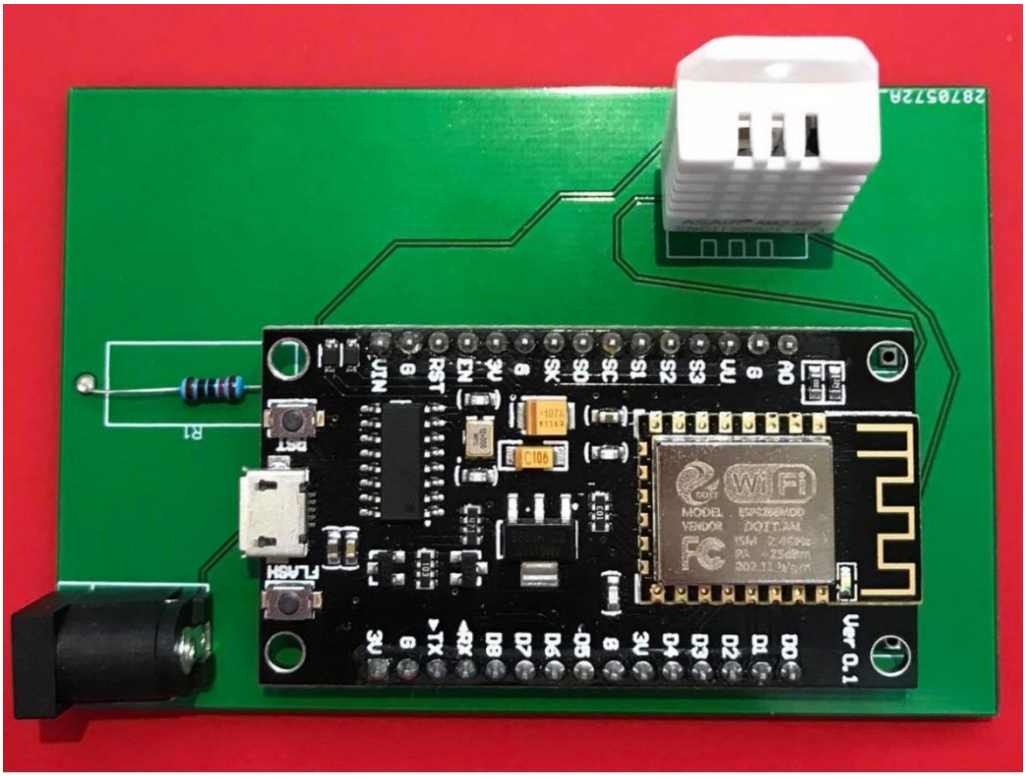


Figure 6.10: NODEMCU Lua ESP8266 v3 complete PCB

7 Arduino IDE

For programming microcontrollers, the Arduino IDE platform (version 1.8.9) was chosen. The choice of this platform was primarily due to the fact that the Arduino IDE is widely distributed among IoT developers, as well as the large amount of documentation and the availability of libraries necessary for effective open access on GitHub. Version 1.8.9 was not chosen by chance. This is the most developed and reliable version.

7.1 Features

Arduino is a programming environment that specializes in programming simple automation and robotics systems, focused on amateur-level programmers. Arduino software consists of a free IDE shell for writing and compiling sketches, and programming microcontrollers and adjacent components. [14]

Use the functions of the Arduino made in the following cases: to create autonomous robotics objects, to work with software on a computer using standard wired and wireless interfaces. Work in the Arduino is fully implemented through the IDE shell, which is freely available. Arduino IDE consists of: Project Manager text editor compiler preprocessor compilation tools The shell is written in Java. Available on Windows, Linux, Mac OS. A set of standard Arduino libraries is used. The Arduino programming language is a classic C ++ with some features that make it even easier for novice programmers to write a work program. [14]

Arduino saves program code in files with the * .ino extension. These files are processed by the Arduino preprocessor. The flexibility of the program allows, if desired, to create and connect standard C ++ files to the project. Arduino code contains two mandatory functions: setup () and loop (). setup () is executed once at startup. In this part of the code, it is customary to enter constant values of functions (login / password, network connection procedure, initialization of sensors and other additional components, connection port, program speed). loop () performs a loop repeatedly an infinite number of times. [14]

In the text of the sketch is not necessary to enter the header files of standard libraries.

These files will automatically be added to the Arduino preprocessor in accordance with the project configuration. Custom libraries must be specified. The mechanism of adding libraries by the Arduino IDE project manager is rather non-standard. As source texts, libraries in standard C ++ are added to the working folder in the IDE directory. But the library name is added to the library list, which is accessible via the IDE menu bar. The user can select the desired libraries, and they are added to the compilation list.

Arduino IDE does not configure the compiler and minimizes the settings of other components, which greatly simplifies the use of the program by beginners and reduces the likelihood of problems. Microcontrollers for working with the Arduino IDE should have a pre-patched bootloader. This bootloader allows the user to load his program into the microcontroller without using standard individual hardware programmers. [14]

The program download algorithm into the microcontroller is carried out in three ways: using a USB cable, using an RS-232 interface, and using Ethernet, depending on the connector connector of the microcontroller used. In some microcontrollers (mainly made in China) for downloading a program to the microcontroller, you may need an additional adapter. Bootloader support is built into the Arduino IDE and runs in one click. [14]

Due to its simplicity and openness of the Arduino IDE, additional tools have been created on the basis of this development environment, which make it even easier to work with the code. A prime example of such tools is the graphical development environment - Minibloq. In essence, this is a graphical code generator with some Arduino IDE functions. The main purpose of Minibloq is assisted in learning programming. This development environment is very common among specialized schools of robotic and computational bias. [14]

7.2 Libraries

The capabilities of the Arduino IDE programming environment can be enhanced through the use of libraries. Libraries extend the functionality of programs and carry additional functions, for example, to work with a wireless Wi-Fi network or work with a particular data format. Standard libraries are installed automatically with the development environment, but it is possible to download or create your own libraries.

To connect the library to the program, select it from the Sketch menu> Import Library.

Standard libraries:

- EEPROM - read and write to "permanent" memory,
- Ethernet - to connect to the Internet through the Arduino Ethernet expansion card,
- Firmata - for interaction with applications on a computer using a standard serial protocol,
- GSM - to connect to the GSM / GRPS network via the GSM expansion card,
- LiquidCrystal- for working with liquid crystal displays (LCD),
- SD - to read and write data to the SD memory card,
- Servo - to control servomotors,
- SPI - for interfacing with peripheral devices via the SPI serial interface,
- SoftwareSerial - to implement serial interfaces on any digital outputs,
- Stepper - to control stepper motors,
- TFT - to display text, images and graphics primitives on the Arduino TFT screen,
- WiFi - to connect to the Internet through the Arduino WiFi expansion card,
- Wire - to work with a two-wire interface that allows you to receive or send data between a network of devices or sensors.

For effective interaction of the microcontroller with the sensor and, in consequence, the server, I used the following Arduino IDE libraries:

- ArduinoJson.h (JSON support library),
- PubSubClient.h (publish / subscribe MQTT support library),
- DHT.h (operations with temperature and humidity sensor DHT22 (AM2302)),
- WIFI.h (standard ESP32 library for working via WIFI connection).

During the installation of libraries, I encountered some difficulties associated with a large number of obsolete inefficient and low-quality libraries to work effectively.

8 Program in C# language

The next important step in my bachelor's thesis was the creation of an application for a graphic illustration of the change in temperature and relative humidity based on the data obtained from the weather station.

According to the requirements of my project, this application must be written in the C # development environment. The program should be a comfortable and understandable interface in which there will be two coordinate planes for building graphs of changes in air temperature and relative humidity in real time.The program also contains buttons for clearing graphs upon completion of the necessary measurements, as well as buttons for saving data received from a broker in CSV (Comma-Separated Values) format - a text format responsible for the presentation of tabular data. Two buttons on each chart, respectively.

The main problem I encountered while writing this project in C # is the transformation of data received in JSON format, as well as the correct setting of the connection to the MQTT broker. The development environment is Microsoft Visual Studio 2015. Since it is this version in my opinion is the most developed, successful and comfortable for programming. In writing the program, I was guided by the logic “brevity - a sign of excellence”, so I decided to use minimalism in the software interface and also in the construction of graphs. Libraries were chosen standard windows forms, as in my opinion it is ideal for working with classic 2D graphics.
In general, during the course of this task, the author improved skills in working with the C# language and, in general, a simple program was created with a comfortable and intuitive interface for users of all levels to work with weather station data.

A fragment of code is presented below; it is possible to see how the program processes MQTT messages for further graphing.

```
private bool PublishArrived(object sender, PublishArrivedArgs e)
{
    string payload = e.Payload;
    AddValues(payload);
    return true;
}

delegate void AddValuesCallback(String payload);

private void AddValues(String payload)
{
    if (this.temperatureChart.InvokeRequired ||this.humidityChart.InvokeRequired)
    {
        AddValuesCallback add = new AddValuesCallback(AddValues);
        this.Invoke(add, new object[] { payload });
    }
    else
    {
        JObject json = JObject.Parse(payload);
    double temperature =Double.Parse(json.SelectToken("temperature").ToString());
    double humidity = Double.Parse(json.SelectToken("humidity").ToString());
    temperatureSeries.Points.AddXY(DateTime.Now.ToString("hh:mm"), temperature);
    temperatureChart.Update();
    humidityeSeries.Points.AddXY(DateTime.Now.ToString("hh:mm"), humidity);
    humidityChart.Update();
    }
 }
```

The figure below shows the program interface at the time of immediate work.

Graphs are updated at intervals of 10 seconds:

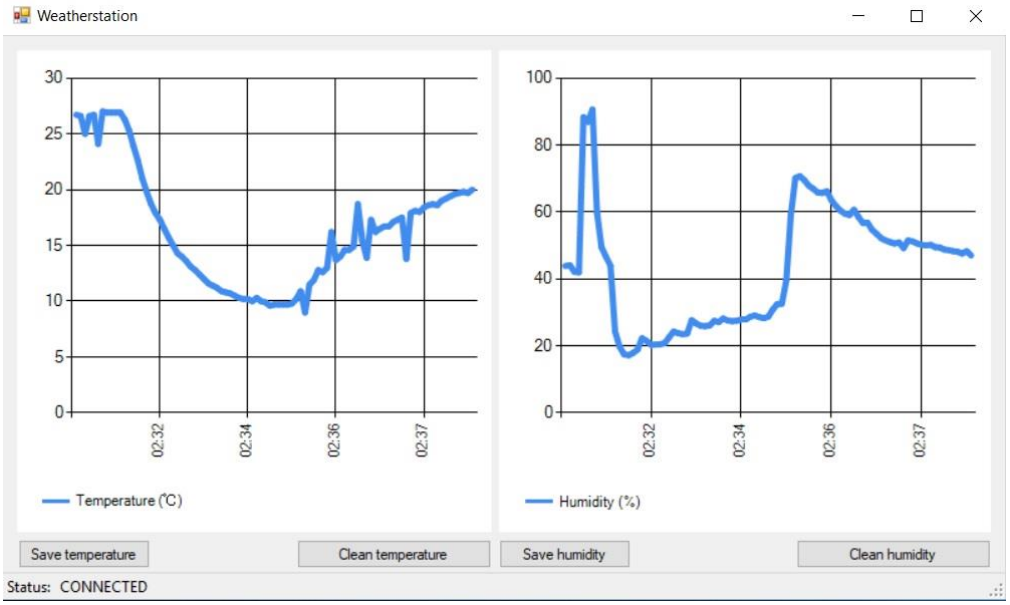


Figure 8.1: Processing received data in the program

After pressing the “Save temperature” or “Save humidity” buttons, the save window opens:

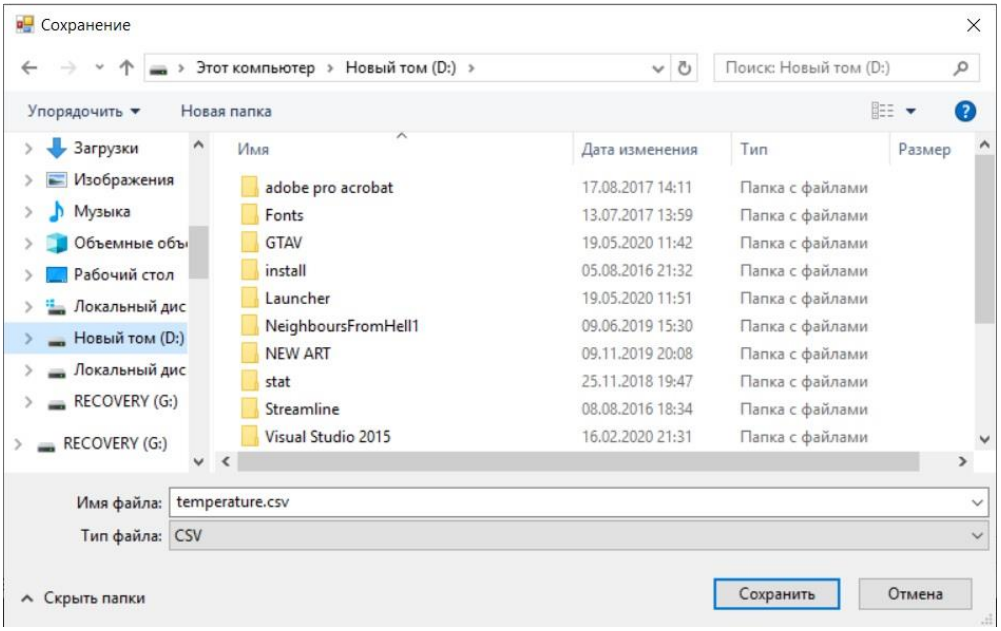
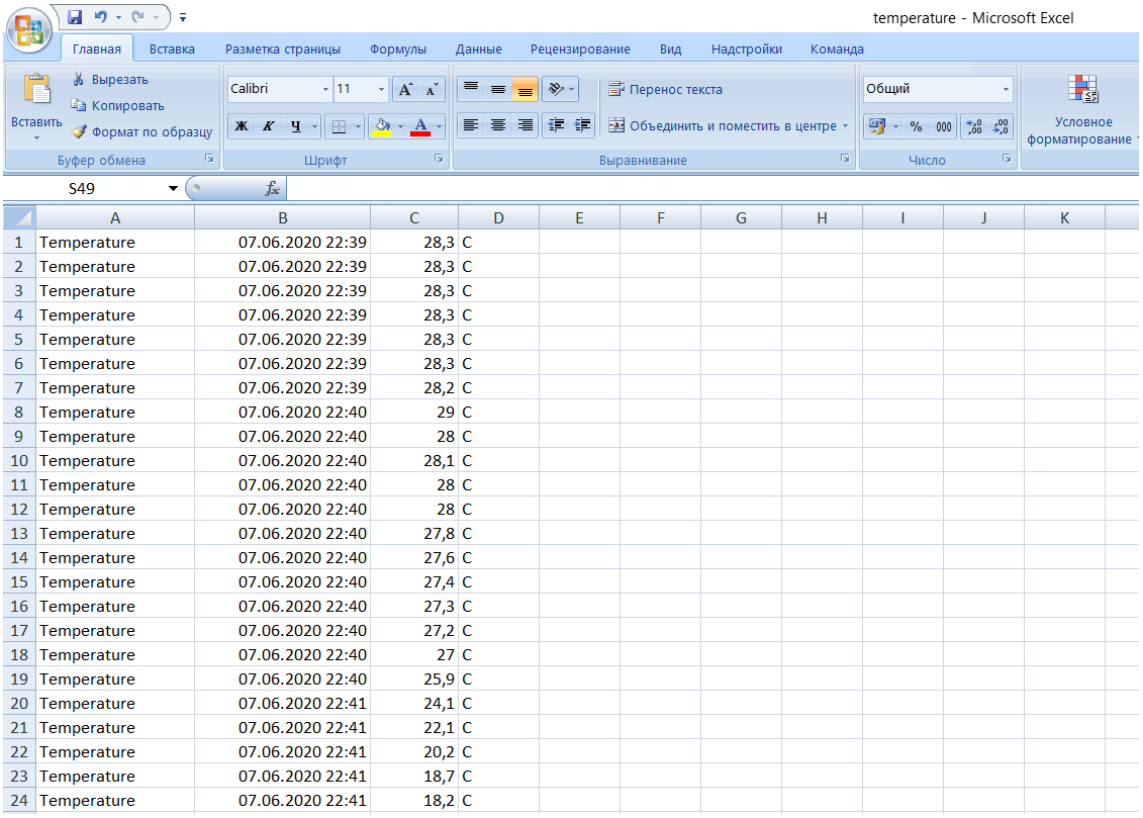


Figure 8.2: Exel save window

This is how the .csv file looks like with saved data:



	A	B	C	D	E	F	G	H	I	J	K
1	Temperature	07.06.2020 22:39	28,3	C							
2	Temperature	07.06.2020 22:39	28,3	C							
3	Temperature	07.06.2020 22:39	28,3	C							
4	Temperature	07.06.2020 22:39	28,3	C							
5	Temperature	07.06.2020 22:39	28,3	C							
6	Temperature	07.06.2020 22:39	28,3	C							
7	Temperature	07.06.2020 22:39	28,2	C							
8	Temperature	07.06.2020 22:40	29	C							
9	Temperature	07.06.2020 22:40	28	C							
10	Temperature	07.06.2020 22:40	28,1	C							
11	Temperature	07.06.2020 22:40	28	C							
12	Temperature	07.06.2020 22:40	28	C							
13	Temperature	07.06.2020 22:40	27,8	C							
14	Temperature	07.06.2020 22:40	27,6	C							
15	Temperature	07.06.2020 22:40	27,4	C							
16	Temperature	07.06.2020 22:40	27,3	C							
17	Temperature	07.06.2020 22:40	27,2	C							
18	Temperature	07.06.2020 22:40	27	C							
19	Temperature	07.06.2020 22:40	25,9	C							
20	Temperature	07.06.2020 22:41	24,1	C							
21	Temperature	07.06.2020 22:41	22,1	C							
22	Temperature	07.06.2020 22:41	20,2	C							
23	Temperature	07.06.2020 22:41	18,7	C							
24	Temperature	07.06.2020 22:41	18,2	C							

Figure 8.3: the .csv file with saved data

CONCLUSIONS

The main goal of the Bachelor thesis was a detailed study of the basic properties and functions of the MQTT network protocol, as well as the possibility of using the protocol for practical purposes for collecting, processing and transmitting meteorological data.
The next part of the study was to analyze the functionality of the JSON text format and the further possibility of implementing the format for practical purposes for transmitting meteorological data. Further research consists in a detailed comparison of the leading ESP8266 and ESP32 microcontrollers and digital sensors for the implementation of my project to create autonomous weather stations. In the practical part of the work, a practical scheme of connection and interaction all the components of the project was developed. The final stage of research is devoted to the implementation of a program for processing meteorological data in the C # programming language.

Certain MQTT brokers for transferring data from IoT devices have been described and compared to make the right choice of a broker whose functionality will allow to realize the objectives of this project.

The theoretical rationale for the selection of sensory data and microcontrollers is given in the Chapter 2 of my project. Chapter 3 is devoted to a detailed theoretical analysis of the basic properties and functions of the MQTT network protocol. The use of this network protocol in my project is due primarily to the main theme of the project, and also because of the high efficiency and prevalence of this protocol in the market of weather stations and IoT devices. The final product of my thesis is a fully autonomous, customizable wireless model data exchange: weather station –broker – program, based on the MQTT network protocol using thematic subscriptions.

During the implementation of this thesis, I improved my understanding of transport protocols, as well as their formats. I learned to use them in programming languages, convert and translate. This project also helped me improve my knowledge of microcontrollers and digital sensors, as well as the IoT industry in general. I have repeatedly improved programming skills and logic. In general, this project helped me to better understand the nature of the MQTT protocol, to get acquainted with its main advantages in practice. This protocol has established itself as a very reliable and easy to

implement, which proved its indispensability in IoT development. The following benefits of this work I would like to highlight my acquaintance with microcontrollers, as well as the practical application of my programming knowledge to create weather stations based on microcontrollers. It was a pleasant experience with many challenges. During work on the project, the author came up with a couple of possible business ideas, which is evidence that this area of development is very promising, primarily in the commercial plan. In general, the author is pleased with the work done.

Work on this bachelor's thesis was a valuable and useful experience for me as a student in the telecommunication area. I started work on this project last year and thanks to this project I improved my knowledge in the field of telecommunications and, in general, expanded the range of my skills to further perfect my professionalism.

The topic is quite new but rapidly developing, therefore this direction is very promising and this project can serve in future as a good basis for writing a master's thesis.

REFERENCES

[1] Department of Atmospheric Sciences (DAS). Meteorology: Clouds and precipitation: Relative humidity. *University of Illinois at Urbana–Champaign* [online]. Urbana and Champaign, USA: Department of Atmospheric Sciences (DAS), 1999 [cit. 2020-04-21]. Available from: http://ww2010.atmos.uiuc.edu/(Gh)/guides/mtr/cld/dvlp/rh.rxml

[2] World Meteorological Organization. *Guide to Meteorological Instruments and Methods of Observation: Measurement of temperature* [online]. Secretariat of the World Meteorological Organization. Geneva, Switzerland: World Meteorological Organization (WMO), 2008 [cit. 2020-04-22]. ISBN 978-92-63-100085. Available from: https://www.weather.gov/media/epz/mesonet/CWOP-WMO8.pdf

[3] Wylie R.G., LalasT. *Measurement of Temperature and Humidity:Specification, Construction, Properties and Use of the WMO Reference Psychrometer*. Secretariat of the World Meteorological Organization, Geneva, Switzerland: World Meteorological Organization (WMO), 1992 [cit. 2020-04-22]. Technical Note No. 194 (WMO-No. 759).

[4] Espressif Systems (Shanghai) Co., Ltd. ESP32-DevKitC V4 Getting Started Guide. *Espressif* [online]. Shanghai, China: Espressif Systems (Shanghai) Co., 2016 [cit. 2020-04-21]. Available from: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html

[5] Espressif Systems (Shanghai) Co., Ltd. NodeMCU ESP-32S Started Guide. *Zerynth* [online]. Shanghai, China: Espressif Systems (Shanghai) Co., 2018 [cit. 2020-04-21]. Available from: https://docs.zerynth.com/latest/official/board.zerynth.nodemcu_esp32/docs/index.html

[6] Espressif Systems (Shanghai) Co., Ltd. ESP32-WROOM-32 Datasheet. *Espressif* [online]. Shanghai, China: Espressif Systems (Shanghai) Co., 2019 [cit. 2020-04-22]. Available from: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[7] SYNACORP TRADING & SERVICES. NODEMCU Lolin Lua IoT ESP8266 Wifi Controller Board v3 with CH340 Guide. *SYNACORP* [online]. Guangzhou, China: SYNACORP TRADING & SERVICES, 2016 [cit. 2020-04-22]. Available from: http://synacorp.my/v3/en/internet-of-things-iot-/1747-arduino-interface-shield.html

[8] Aosong(Guangzhou) Electronics Co.,Ltd. Temperature and humidity module: DHT11Product Manual. *Akizukidenshi* [online]. Guangzhou, China: Aosong(Guangzhou) Electronics Co., 2017 [cit. 2020-04-21]. Available from: https://akizukidenshi.com/download/ds/aosong/DHT11.pdf

[9] Aosong(Guangzhou) Electronics Co.,Ltd. Temperature and humidity module: AM2302 Product Manual. *Akizukidenshi* [online]. Guangzhou, China: Aosong(Guangzhou) Electronics Co., 2017 [cit. 2020-04-21]. Available from: https://akizukidenshi.com/download/ds/aosong/AM2302.pdf

[10] HANES D., SALGUEIRO G., BARTON R. *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things*. Cisco Systems, Inc. Indianapolis, IN 46240 USA: Cisco Press, 2017. ISBN 1587144565.

[11] Eclipse Mosquitto. *Mosquitto* [online]. New York, USA: Eclipse Foundation, 2001 [cit. 2020-04-21]. Available from: https://mosquitto.org/

[12] VerneMQ. *VerneMQ* [online]. Zurich, Switzerland: VerneMQ, 2007 [cit. 2020-04-21]. Available from: https://vernemq.com/

[13] RabbitMQ. *RabbitMQ* [online]. San Francisco, USA: Pivotal Software, 2007 [cit. 2020-04-21]. Available from: https://www.rabbitmq.com/

[14] Arduino Software. Arduino. *Arduino: Introduction* [online]. Ivrea, Italy: Arduino, 2003 [cit. 2020-04-22]. Available from:https://www.arduino.cc/en/guide/introduction

List of attachments

Attachment 1 - Contents of the enclosed CD

The folder contains the executable program *Weatherstantion.sln* and all associated parts for processing meteorological data. The folder also contains the codes *ESP32S.ino* and *ESP8266.ino* for compilation in weather stations, one code per station model ESP32 and ESP8266.