ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 8, No. 1, 2023

IMPLEMENTATION OF THE WEATHER STATION SOFTWARE ON A MICROPROCESSOR PLATFORM USING .NET TECHNOLOGY

Bohdan Marii, Tetyana Pavych, Yaroslav Paramud

Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine
MedBridge Inc, 1633 Westlake Ave, Seattle, WA, 98109, USA
Authors' e-mails*: marii.bohdan1@gmail.com, tpavych98@gmail.com, yaroslav.s.paramud@lpnu.ua*

https://doi.org/10.23939/acps2023.01.057

Submitted on 23.02.2022

***Abstract*: The article presents an implementation of the weather station software on a microprocessor platform using .NET technology. The system consists of a hardware module that collects weather data, a microprocessor platform that processes data, and a software application that visualizes and stores data. The software system is designed using the .NET platform, which provides an environment for software development. The system uses a web interface that allows users to access the weather from anywhere with a web browser. The test results of the system demonstrate collecting, processing, and presenting the weather in real time. By comparing readings from AccuWeather with data collected by Arduino sensors, we ensure the accuracy of measurements. AccuWeather Europe is a source of weather data that can be used to validate weather information collected by Arduino sensors. In the developed systems, neural networks for weather forecasting are also used. The neural networks learn patterns and relationships in historical weather data to predict future weather conditions**

***Index Terms*: weather station; software system; microprocessor platform; .NET technology; web interface.**

I. INTRODUCTION

Weather monitoring and analysis are critical in various industries, such as agriculture, aviation, and meteorology, where decision-making requires accurate weather data. One of the solutions is the software system, which consists of a hardware module that collects weather data, a microprocessor platform that processes data, and a software application that visualizes and stores data.

The use of a network of wireless weather stations to forecast weather in developing countries involves the creation of a network of inexpensive battery-powered meteorological stations in key places throughout the region. These meteorological stations can collect data on temperature, humidity, wind speed, and other meteorological parameters, which are then transmitted on a wireless connection to the central database [1].

The introduction of a weather station for real weather in the room involves adjusting the system that can accurately measure and display different weather conditions in the room, such as temperature, humidity, air pressure, and perhaps even air quality [2].

The assessment of the quality of work of an automatic meteorological station based on a fuzzy AHP is to use a combination of fuzzy logic and analytical hierarchical process (AHP) to evaluate the performance of automatic weather stations.

To begin with, a set of criteria and subsections is established to evaluate the quality of the meteorological station, such as data accuracy, sensor accuracy, and frequency of service [3].

The Arduino board is equipped with a Wi-Fi module the ESP8266, which allows the board to connect to a Wi-Fi network and access the internet. This enables the board to perform a wide range of tasks, such as sending data to a server, retrieving data from online sources, or interfacing with other IoT (Internet of Things) devices.

To ensure the security of authentication and data transmission in wireless access points (AP), two main protocols are currently available – WEP and WPA [4]. In recent years, WPA has become the more widely adopted protocol due to its stronger security features and improved resistance to attacks compared to WEP. So, WPA was used in a developed system.

In summary, Wi-Fi can be a powerful tool for enabling an Arduino board to connect to the internet or a local network, allowing it to access online resources, communicate with other devices, and perform a wide range of IoT-related tasks.

The hardware module of the software system includes sensors that collect various weather data, such as temperature, humidity, pressure, speed, and direction of wind [5]. The microprocessor platform is responsible for processing the data collected by sensors and converting them into a readable format for the program. The software uses data to create graphs, diagrams, and other visualizations that provide a clear and short idea of weather conditions. Fig. 1 shows a brief overview of developed system interactions. The figure represents a complex sensor network that can be used for environmental monitoring and data collection, with the ability to store and transmit data wirelessly for further analysis or visualization. The AM2320 digital temperature and humidity sensor, BMP280 barometric pressure and altitude sensor, ML8511 UV sensor, and GP2Y1010F45 optical dust sensor use the ESP8266 Wi-Fi module for wireless communication, and an SD card module is added for data storage.

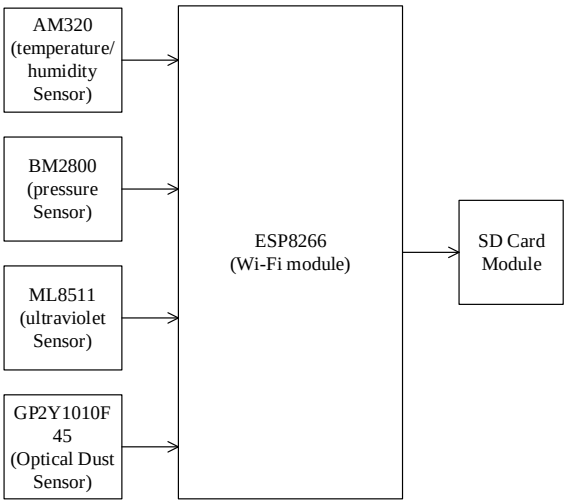


Fig. 1. Block diagram of a system

The .net Framework [6], used in the development of software applications, provides several advantages. Firstly, it allows you to quickly develop a program by reducing the time and cost of development. Secondly, it provides a safe environment for the program, ensuring the integrity of the data. Finally, this makes it easy to integrate other software and systems, improving the overall functionality of the system [7].

To summarize, the software system on a microprocessor platform using .NET and a web interface provides a solution for monitoring and weather analysis. The system's software is designed using the .NET platform and includes various design templates that increase the functionality of the system and fitness for maintenance. The potential application of the system includes rural households, aviation, meteorology, and other industries that rely on accurate real-time weather data.

II. OVERVIEW OF KNOWN METHODS AND TOOLS FOR SOLVING PROBLEMS

Weather systems on microprocessor platforms can be developed and implemented through various programming languages and tools. A microprocessor-based weather station software system is designed to collect data from various weather sensors, process the data, and display it in a user-friendly format. The system is typically composed of hardware components such as sensors, a microcontroller, and software components such as data acquisition, processing, and display software. There are several solutions available for developing a microprocessor-based weather station software system, including:

- Arduino: Arduino is an open-source platform that offers a variety of microcontrollers suitable for weather station applications. It provides a range of libraries and code examples that simplify the development process [8].
- Raspberry Pi: Raspberry Pi is another popular open-source platform for developing microprocessor-based systems. It offers a powerful microcontroller and a range of peripheral interfaces, making it suitable for more complex weather station applications.
- LabVIEW: LabVIEW is a graphical programming environment developed by National Instruments. It provides an intuitive interface for developing data acquisition and processing software, making it suitable for less experienced programmers.
- MATLAB: MATLAB is a high-level programming language and development environment used for data analysis, visualization, and algorithm development. It offers a range of tools and libraries for processing weather data, making it suitable for more complex applications.

Overall, the choice of solution for a microprocessor-based weather station software system depends on factors such as the complexity of the application, the level of programming experience, and the availability of hardware components. Each solution offers its advantages and disadvantages, and the best choice will depend on the specific requirements of the application.

Here is an overview of some known methods and tools for solving meteorological systems on microprocessor platforms using network technologies:

.NET Micro Framework: .NET Micro Framework is an open code platform to create built-in programs using .Net Framework. It provides a set of APIs to develop applications for microcontrollers with limited resources, such as small amounts of RAM and flash memory.

Visual Studio: Microsoft Visual Studio is an integrated development environment (IDE) that can be used to develop programs using .Net Micro Framework. It provides a set of tools for the design, development, and adjustment of programs for microcontroller devices.

C#: C# is a programming language developed by a Microsoft corporation that is widely used to create programs using .net Framework. It is a simple, modern, and typically safe language that provides a high level of abstraction for the development of complex programs.

MQT: MESGE QUEUING Telemetry Transport is a light message exchange protocol that is widely used to create IOT programs. It provides a way to exchange messages between devices, including microprocessor platforms.

Azure Hub: Azure Hub is a cloud platform to control and connect IOT devices.

C/C++: C and C++ are popular programming languages for developing embedded systems. They offer a range of libraries and tools for low-level programming and are often used in conjunction with microcontrollers. They are suitable for more complex weather station applications that require real-time data processing and control.

Wireless Communication: Wireless communication protocols such as Wi-Fi, Bluetooth, and Zigbee can be used to transmit data from the weather station to a central server or display device. This eliminates the need for physical connections and enables remote monitoring and control of the weather station [9].

Cloud Services: Cloud services such as AWS IoT, Microsoft Azure, and Google Cloud Platform provide a

range of tools and services for developing and deploying IoT applications. They can be used to store, process, and analyze weather data in real-time, and provide a scalable and flexible solution for managing large amounts of data.

In summary, developing a microprocessor-based weather station software system requires careful consideration of the hardware and software components. Securing IoT devices involves implementing measures such as encryption, access control, and regular software updates to protect against vulnerabilities and ensure the confidentiality, integrity, and availability of data. Securely connecting the dots using REST API and middleware involves using a standardized interface (REST API) and middleware software to facilitate communication between different IoT devices while implementing security measures such as authentication, authorization, and data encryption to protect against threats [10].

The combination of .NET Micro Framework, Visual Studio, C#, Asp.net, MQT, Azure IoT Hub, and Thing speak provides a powerful set of tools and platforms to create meteorological platforms on microprocessor platforms using network technology [11]. An efficient IoT-based weather station involves using sensors to collect data on weather conditions such as temperature, humidity, and precipitation, transmitting the data wirelessly to a central hub or cloud-based platform and using machine learning algorithms to analyse and predict weather patterns [12].

III. GOAL OF THE WORK

The purpose of this work is to develop and implement a weather software system on a microprocessor platform using .NET technology, with a web interface for visualization and storage of data. The system aims to provide a solution for monitoring and analysis of the weather in various areas, such as agriculture, aviation, and meteorology. One of the purposes is the incorporation of machine learning algorithms to improve the accuracy of weather predictions. This can be accomplished by training a neural network or other machine learning model on historical weather data and then using the model to make more accurate predictions based on real-time sensor data. To measure the effectiveness of this improvement, we will compare the accuracy of the predictions made by the machine learning model to the accuracy of predictions made by other weather systems without machine learning. This could be done by comparing the predicted weather conditions to actual weather conditions over a while and calculating statistical measures such as mean absolute error, standard deviation, and correlation coefficients, the mean absolute error and standard deviation should be at least 0.5 °C lower for a neural network compared to other weather systems without machine learning. The work also aims to demonstrate the benefits of using the .NET platform and software design templates in the development of software systems.

IV. SYSTEM COMPONENTS

Weather software system on a microprocessor platform using .NET technology with a web interface for visualization and storage of data will have the following components:

- Data acquisition component: This component is responsible for collecting weather data from various sources, such as sensors, satellites, and weather stations. The collected data is then stored in a database for further processing and analysis.
- Data processing and analysis component: This component is responsible for processing and analysing the collected weather data to extract meaningful information. It uses advanced algorithms and statistical models to identify patterns, trends, and anomalies in the data.
- Web interface component: This component provides a web-based user interface that allows users to visualize the weather data and perform various operations, such as querying, filtering, and exporting data. The web interface also provides real-time updates and notifications to keep users informed of any changes in weather conditions.
- Storage component: This component is responsible for storing the collected weather data in a secure and scalable manner. It ensures that the data is easily accessible and available for future use.
- Integration component: This component provides integration capabilities with other systems and platforms, such as mobile devices and third-party applications, to extend the system's functionality and reach.

Fig. 2 shows a temperature graph, which is displayed in the visualization interface. A temperature graph is a graphical representation of temperature over time, with the horizontal X-axis representing time in hours and the vertical Y-axis representing temperature in Celsius. The graph displays a continuous line that connects a series of temperature data points plotted against their corresponding time values.

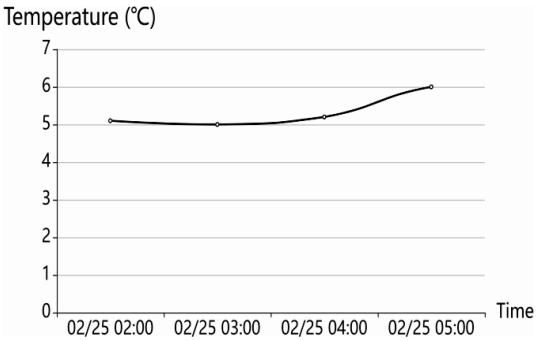


Fig. 2. Temperature graph

The project includes an Arduino-based weather station, an ASP.NET Core MVC data presentation page, and an ASP.NET Core WEB API. The collected data is sent using ESP8266 and written to the SQL Server database via WEB API. The weather station consists of the following parts:

- Arduino Nano.
- Nano Shield.
- AM2320 – Digital Temperature and Humidity Sensor.
- BMP280 – Barometric Pressure and Altitude Sensor.
- ML8511 – UV sensor.

- GP2Y1010F45 – Optical Dust Sensor.
- DS3231 – RTC.
- ESP8266 – Wi-Fi.
- SD card module.
- 18650 Li-ion and Charger (Optional).

The ESP8266 is based on a 32-bit LX106 processor and can be programmed using the Arduino IDE or the Lua scripting language. It has GPIO pins that can be used to control external devices, and it also supports protocols such as SPI, I2C, and UART for communicating with other devices.

One of the most notable features of the ESP8266 is its built-in Wi-Fi capabilities, which allow it to connect to a wireless network and communicate with other devices over the internet. It can function as a client or a server and supports both TCP and UDP protocols.

Due to its low cost and small size, the ESP8266 is often used in DIY projects and small-scale IoT applications, such as home automation, weather monitoring, and smart appliances. It has since been superseded by the ESP32, but it remains a popular choice for many makers and hobbyists.

The system consists of an Arduino Nano microcontroller, which is connected to several sensor modules, including a digital temperature and humidity sensor (AM2320), a barometric pressure and altitude sensor (BMP280), a UV sensor (ML8511), and an optical dust sensor (GP2Y 1010F45). The system also includes a real-time clock module (DS3231), a Wi-Fi module (ESP8266), and an SD card module for data storage.

Also, in the system, Heat Index is calculated. Heat Index is calculated by taking into account both temperature and humidity, as high humidity can make the air feel hotter than it is. Heat Index is often used by meteorologists to issue heat advisories or warnings to help people take appropriate precautions during hot weather. It's important to stay hydrated, seek shade, and limit outdoor activities during times of high heat index to avoid heat-related illnesses. Heat Index values above 33 °C are considered dangerous, and values above 41 °C can be life-threatening.

The formula for calculating the heat index in degrees Celsius:

$$Hi = -8.784695 + 1.61139411 \times T + \\ +2.338549 \times H - 14.611605 \times T \times H - \\ - 0.012308094 \times T^2 - 0.016424828 \times \\ \times H^2 + 0.002211732 \times T^{2 \times H} + \\ +0.00072546 \times T \times H^2 - 0.000003582 \times T^2 \times H^2.$$

Using (1), where Hi is the heat index, T is the temperature in Celsius and H is the relative humidity as a percentage, the heat index was calculated.

The Arduino Nano is the central processing unit of the system, responsible for collecting data from the various sensors, processing the data, and controlling the output of the system. The sensor modules provide environmental data, such as temperature, humidity, pressure, UV intensity, and dust particle concentration, which are used by Arduino to monitor and control the system.

The real-time clock module ensures that the system maintains accurate time and can schedule tasks and events based on the current time. The Wi-Fi module provides connectivity to the internet, allowing the system to upload data to a remote server or receive commands from a remote client.

The SD card module allows the system to store data locally, providing a backup in case of connectivity issues or server downtime. The optional 18650 Li-ion battery and charger provide a portable power source for the system, making it suitable for use in remote or outdoor environments.

Fig. 3 shows an illustration of the System structure. The weather station is an Arduino-based weather station, which is sending weather data using ESP8266. ESP8266 is a low-cost, Wi-Fi-enabled microcontroller that can be used for a variety of IoT (Internet of Things) applications. It was firstly introduced in 2014 by the Chinese company Systems, and it quickly gained popularity due to its small size, low power consumption, and built-in Wi-Fi capabilities.

The Web API (Application Programming Interface) layer and controller level are both important components of a software application.

The API layer is the interface between the application and the outside world. It allows other applications and services to interact with the application by providing a set of functions, protocols, and rules that define how they can communicate with it. APIs can be used for a variety of purposes, such as data sharing, integration with third-party services, and automation of tasks.

In a typical software application, the API layer sits on top of the application's business logic layer, which contains the core functionality of the application. The API layer provides a simplified and standardized way of accessing the application's functionality while abstracting away the complexity of the underlying implementation.

Controllers are often organized into a hierarchy, with higher-level controllers responsible for handling broader requests and lower-level controllers handling more specific requests. For example, a web application might have a top-level controller that handles requests for the entire application, and then lower-level controllers that handle requests for specific features or modules of the application.

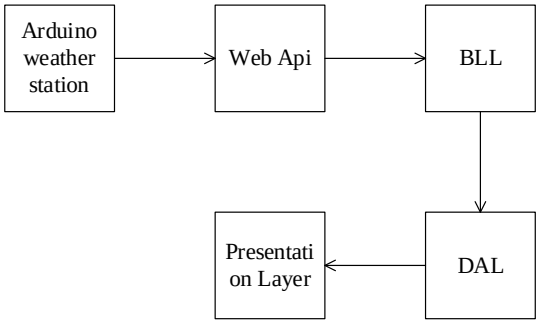


Fig. 3. System structure

Overall, the system architecture is designed to provide a flexible and modular platform for environmental monitoring and control, with the ability to communicate data and receive commands from remote clients.

Arduino Connection

I2C Sensors

- VCC – 5V;
- GND – GND;
- SDA – A4;
- SCL – A5.

ESP8266

- TX – pin 3;
- RX – pin 2;
- CH_PD – pin 5;
- VCC – 3.3V;
- GND – GND.

V. SYSTEM DEVELOPMENT

The code for the weather station will depend on the specific sensors we're using.

Fig. 4 shows a class diagram. The main class is Weather. The class "Weather" represents a weather measurement with several fields:

- Weather Id: a unique identifier for each weather measurement, generated by the database.
- Date Time: the date and time when the measurement was taken.
- Weather Name: a string describing the weather conditions (e. g., sunny, cloudy, rainy).
- Temperature: a double value representing the temperature in degrees Celsius or Fahrenheit.
- Humidity: a double value representing the relative humidity as a percentage.
- Pressure: a double value representing the atmospheric pressure in Pa.
- Dust: a double value representing the concentration of dust particles in the air.
- UV: a double value representing the intensity of ultraviolet radiation.

Fig. 5 shows a table diagram for the database. It is pretty much the same with the same fields as the Weather model in the .Net project, which has also necessary weather information. The table has 8 columns:

- Weather Id: This is a serial data type and serves as the primary key for the table. It is automatically populated with a unique value for each new row added to the table.
- Date Time: This column is of the data type TIMESTAMP and allows the user to store date and time values. It is allowed to be null, meaning it's optional to have a value for each record.
- Weather Name: This column is of the data type VARCHAR (10) and stores a string representing the weather condition. It is allowed to be null.
- Temperature: This column is of the data type double precision and stores the temperature value. The double-precision data type allows the user to specify the total number of digits that can be stored, as well as the number of digits after the decimal point. In this case, the user can store values up to 4 digits, with 2 of them after the decimal point. It is allowed to be null;
- humidity: This column is of the data type double precision and stores the humidity value. It is allowed to be null;

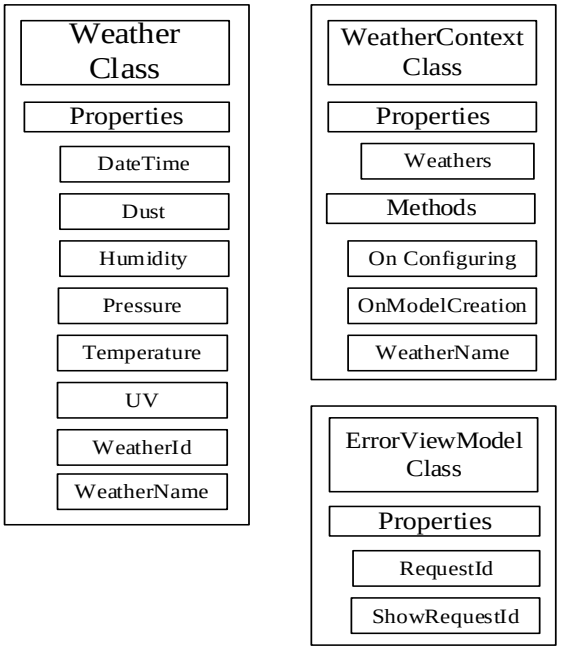


Fig. 4. Class diagram

- pressure: This column is of the data type double precision and stores the pressure value. It is allowed to be null;
- dust: This column is of the data type double precision and stores the dust value. It is allowed to be null;
- UV: This column is of the data type double precision and stores the ultraviolet (UV) radiation from the sun value. It is allowed to be null.

The constraint PK_WEATHER primary key (Weather Id) at the end of the code block specifies that the primary key for this table is the Weather Id column.

In the code, we have a Weather Controller class that derives from the Controller Base class. It is an ASP.NET Core web API controller that handles HTTP requests for weather data.

The Weather Controller class has a constructor that receives a Weather Context object. The Weather Context is used to connect to a database that stores weather data. The class also has two methods, Get and Post.

The Get method handles HTTP GET requests and returns a Weather object with the specified ID. The Post method handles HTTP POST requests and inserts a new Weather object into the database.

An Arduino sketch was written in C++ language that is designed to collect data from multiple sensors and post the data to a remote server over Wi-Fi. A summary of the sensors used:

- SD card: used for data logging.
- UV sensor: measures UV intensity.

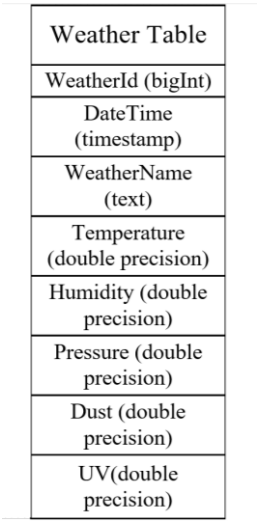


Fig. 5. Weather table diagram

- Dust sensor: measures dust concentration.
- DS3231: a real-time clock (RTC) module that provides an accurate date and time information.
- AM2320: a digital temperature and humidity sensor.
- BMP280: a digital pressure sensor.
- ESP8266: a Wi-Fi module used to connect to a wireless network and post the data to the remote server.

The sketch starts by initializing the sensors and establishing a Wi-Fi connection using the ESP8266 module. The loop function waits for 10 minutes before calling the WriteSdAndPost function, which logs the data to the SD card and posts it to the remote server. The WriteSdAndPost function reads the date and time from the RTC module, as well as the temperature, humidity, pressure, UV intensity, and dust concentration from the other sensors. The data is then formatted into a string and sent to the server using an HTTP POST request.

Some important points to note about the sketch:

The specific pins used for each sensor are listed at the beginning of the sketch.

The sketch uses several libraries, including SoftwareSerial, DS3231, Adafruit_BMP280, and AM2320. These libraries need to be installed before uploading the sketch to the Arduino board.

Also, the neural network is used for weather forecasting. Weather forecasting is a complex and challenging task that involves analysing vast amounts of data from various sources, such as temperature sensors, humidity sensors, and other sensors. Neural networks are a type of artificial intelligence that can learn patterns and relationships in the data and use this information to make predictions. In weather forecasting, neural networks are trained using historical weather data, and the trained network can then be used to predict future weather conditions

The neural network consists of an input layer that takes in various weather features such as temperature, humidity, wind speed, precipitation, and pressure. The number of nodes in the input layer depends on the number of weather features being used as input. Multiple hidden layers are used with various numbers of neurons in each layer. These hidden layers use activation functions such as ReLU or tanh to add non-linearity to the model and to help the model learn complex relationships between the input features. The number of neurons in the hidden layers can be determined through hyperparameter tuning.

The data used to train the neural network needs to be preprocessed before feeding it to the model. Preprocessing steps may include scaling the data, normalizing the data, or filling in missing values in the data.

Feature selection is an important step in weather forecasting because not all weather features may be relevant for predicting a specific weather condition. Selecting the right set of features can improve the accuracy of the neural network.

The neural network has various hyperparameters that need to be tuned to achieve the best performance. Some of the hyperparameters include the number of hidden layers, the number of neurons in each hidden layer, the learning rate of the optimizer, and the activation function used in the hidden layers.

The output layer of the neural network predicts the weather condition such as temperature, rainfall, or pressure. The loss function measures the difference between the predicted weather values and the actual weather values. Mean Squared Error (MSE) is a common loss function used in weather forecasting. The number of nodes in the output layer depends on the number of weather conditions being predicted.

The optimizer is used to minimize the loss function and improve the accuracy of the predictions. Adam or Stochastic Gradient Descent (SGD) is a commonly used optimizer for weather forecasting. The neural network is trained on historical weather data. The data is divided into training and validation sets. The training data is used to adjust the weights and biases of the neural network during the training process.

Regularization techniques such as L1, L2, or dropout are used to prevent overfitting and improve the generalization of the model.

The testing data is used to evaluate the performance of the trained neural network. The testing data is not used during the training process. By using a neural network for weather forecasting, it is possible to improve the accuracy of weather predictions by learning complex relationships between the input features and the weather conditions.

The algorithm of working neural networks is the following. To use neural networks for predicting future weather historical weather data for the area of interest is collected, such as temperature, humidity, wind speed, precipitation, and other relevant variables. The data is prepared for input into the neural network by normalizing and scaling it. Then, the data is split into training and testing sets, and the architecture of the neural network will

be defined. This includes the number and type of layers, activation functions used in each layer, and the loss function used to evaluate the performance of the model.

The neural network is trained using the training data, and the weights and biases in the network would be adjusted to minimize the loss function and improve the accuracy of the predictions. The performance of the model is evaluated using the testing data to identify any overfitting or underfitting of the model and suggest any further improvements that need to be made.

Once the model is trained and evaluated, it can be used to make predictions about future weather patterns. The neural network takes in current weather data as input and outputs predictions for future weather patterns, such as temperature, humidity, wind speed, precipitation, and other relevant variables. Regular updates with new data are necessary to ensure the model remains accurate over time.

One of the key advantages of using neural networks for weather forecasting is their ability to handle non-linear relationships in the data. For example, a neural network can learn the complex relationships between temperature, humidity, and atmospheric pressure to predict the likelihood of rain. Additionally, neural networks can be used to predict more than just the weather conditions themselves.

In addition to traditional neural networks, other advanced neural network architectures such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs) can also be used for weather forecasting. RNNs are used at processing time-series data, such as weather measurements, by incorporating feedback loops that allow the network to learn from its previous predictions. CNN, on the other hand, can extract spatial and temporal features from weather data to identify complex patterns and relationships.

VI. RESULTS

Table 1 shows the Latest Data table of the visualization interface of the application, which provides the latest data on environmental conditions.

Table 1

Latest Weather Data

Time	2023/02/25 07:00
Temperature	28.6 °C
Humidity	75.2 %
Pressure	71872.41 Pa
Dust	0.06 mg/m^3
UV	0.01 mW/cm^2
Heat Index	32.8 °C

The first column indicates the parameter being measured, such as temperature, humidity, pressure, and dust.

The second column provides the value of each parameter in its respective unit of measurement. For example, the temperature is measured in degrees Celsius (°C), humidity is measured in percentage (%), pressure is measured in pascals (Pa), and dust is measured in milligrams per cubic meter (mg/m³). UV: The intensity of ultraviolet radiation at the time of data collection, in milliwatts per square centimetre (mW/cm²). In this case, the intensity was 0.01 mW/cm².

The data from Arduino was validated using AccuWeather resources. By combining the power of IoT technology with an AccuWeather Europe data source, we can ensure that we're getting accurate data for our application. The website provides current and historical weather data, including current conditions, hourly and daily forecasts, and radar and satellite imagery. By comparing our readings from Arduino sensors with the data provided by AccuWeather, we can validate the accuracy of our measurements and gain additional insights into the weather conditions in our area. On February 25 at 07:00 AccuWeather was showing 7.1 °C and 78 humidity and 86000 Pa, which means that the data from Arduino sensors were accurate enough for use.

The tables below Table 2 and Table 3 show performance metrics for two different endpoints (GET and POST respectively) of a .Net Web API that provides weather information. Here's a breakdown of each column:

- Number of Requests: This column indicates the number of requests that have been made to each endpoint.
- Many Successful Responses: This column indicates the number of successful responses returned by each endpoint.
- Dust sensor: measures dust concentration.
- DS3231: a real-time clock (RTC) module that provides an accurate date and time information.
- Percentage of Failed Requests: This column lists the percentage of requests that failed for each endpoint.
- Average Request Time (ms): This column lists the average amount of time it takes for the API to process each request for each endpoint, in milliseconds.
- Median Request Time (ms): This column lists the median amount of time it takes for the API to process each request for each endpoint, in milliseconds.

These performance metrics can be used to analyse the performance of the API and identify areas where improvements can be made. If the average or median request times are too high, it may indicate that the API is overloaded.

Table 2

Performance analysis for API/weather GET endpoint

Number of requests	Number of successful responses	The percentage of failed requests, %	Average time of execution, ms	Median request time, ms
10000	10000	0.0	35	30
15000	15000	0.0	45	41
30000	30000	0.0	51	47
50000	50000	0.0	52	48
70000	70000	0.0	55	52

Table 3

Performance analysis for API/weather POST endpoint

Number of requests	Number of successful responses	The percentage of failed requests, %	Average time of execution, ms	Median request time, ms
10000	10000	0.0	41	35
15000	15000	0.0	56	50
30000	30000	0.0	61	55
50000	50000	0.0	66	57
70000	70000	0.0	71	61

To improve the accuracy of the weather predictions, a machine learning model was incorporated that is trained on historical weather data. To train a neural network for weather forecasting, historical weather data is firstly collected and pre-processed. The data is then split into training, validation, and testing datasets. The training dataset is used to adjust the weights of the network's neurons to minimize the prediction error, while the validation dataset is used to evaluate the network's performance during training. The testing dataset is used to evaluate the network's ability to generalize and predict the weather accurately on new, unseen data. Let's compare it with the rule-based algorithm which simply checks whether the temperature is above or below a certain threshold and whether the humidity is above or below a certain threshold, and then makes a prediction based on those two criteria. Sensor data for a week were collected and make weather predictions using both the rule-based algorithm and the machine learning model.

Table 4

The collected data for six consecutive days

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6
Actual data, °C	7.5	6.9	10.1	12.5	6.8	7.5
Predicted by trained machine learning, °C	7.1	6.7	9.1	11.2	7.5	7.4
Predicted by rule-based, °C	8.5	7.5	8.0	10.5	6.2	6.9

So, Table 4 shows the collected data. The table displays temperature data for six consecutive days. The first row shows the actual temperature values in degrees Celsius for each day which was received and stored by Arduino sensors and confirmed using AccuWeather resource, while the second and third rows show the predicted temperature values using different methods. Note that the predicted temperature values from the machine learning system are closer to the actual temperature values as compared to the rule-based system. This indicates that machine learning algorithms can be effective in making accurate predictions.

The formula for calculating the mean absolute error (MAE):

$$MAE = \frac{\sum_{i=1}^{n} y_i - x_i}{n}. \quad (2)$$

Using (2), where MAE is the mean absolute error, *yi* is the prediction for test sequence number (*i*), *n* is the total number of data points, and *xi* is the actual data for test sequence number (*i*), we can calculate the MAE – a rule-based and trained machine learning algorithm, which is 1.15 °C and 0.61 °C respectively.

And the formula for calculating a standard deviation (SD):

$$SD = \sqrt{\frac{\sum_{i=1}^{n} y_i - x_i \ ^2}{n}}. \quad (3)$$

Using (3), where SD is a standard deviation, *yi* is the prediction for test sequence number (i), *n* is the total number of data points, and *xi* is the actual data for test sequence number (*i*), we can calculate the SD a rule-based and trained machine learning algorithm, which is 2.81 °C and 1.51 °C respectively.

From the calculations, we can see that the machine-learning algorithm is more accurate than the rule-based algorithm, as it has lower values for both MAE and SD.

The formula for calculating how much less are MAE and SD for the machine-learning algorithm:

$$R = y - x. \quad (4)$$

Using (4), where *R* – a result, which represents how much lower MAE and SD are for the machine-learning algorithm than the rule-based algorithm, *y* – the MAE and in the second calculation SD for the rule-based algorithm, and *x* – the MAE and in the second calculation SD for the machine-learning algorithm, for the machine-learning algorithm, the MAE is

$$R_{MAE} = y_{MAE} - x_{MAE} = 1.15\ ℃ - 0.61\ ℃ = 0.51\ ℃.$$

Lower than the rule-based algorithm, and the SD is

$$R_{SD} = y_{SD} - x_{SD} = 2.81\ ℃ - 1.51\ ℃ = 1.3\ ℃.$$

Lower, which corresponds to the purpose of the work.

VII. CONCLUSION

In conclusion, the development and implementation of a weather software system on a microprocessor platform using .NET technology was proposed in this work. The system's web interface provided a solution for monitoring and analyzing weather data in various industries, such as agriculture, aviation, and meteorology. By comparing readings from AccuWeather with data collected by Arduino sensors, we ensured the accuracy of measurements. The accuracy of weather predictions was improved by the incorporation of machine learning algorithms to improve the accuracy of weather predictions. The proposed weather software system has many potential benefits for various industries that rely on weather data. For example, in the agricultural industry, accurate weather predictions can help farmers make decisions about when to plant, harvest, and irrigate crops, which can lead to increased yields and reduced water usage.

Similarly, in the aviation industry, weather conditions can affect flight plans and safety, so having access to up-to-date and accurate weather information is crucial for airline operators and pilots.

The predicted weather conditions were compared to actual weather conditions over a while and also calculated statistical measures such as mean absolute error and standard deviation, which are 0.54 °C and 1.3 °C respectively lower for a neural network compared to other weather systems without machine learning. Overall, the proposed system is a valuable tool for those who need to make informed decisions based on accurate and up-to-date weather information.

REFERENCES

[1] T. M. Bumbary (2017). "Utilizing a network of wireless weather stations to forecast weather in developing countries", *IEEE Integrated STEM Education Conference (ISEC)*, Princeton, NJ, USA, pp. 109–111. DOI: 10.1109/ISECon.2017.7910223.

[2] A. Suryana, F. P. Lismana, R. M. Rachmat, S. D. Putra and M. Artiyasa (2016). "Implementation of Weather Station for The Weather Reality in A Room", *6th International Conference on Computing Engineering and Design (ICCED)*, Sukabumi, Indonesia, pp. 1–6. DOI: 10.1109/ICCED51276.2020.9415799.

[3] J.-M. Li, L. Han, S.-Y. Zhen and L.-T. Yao (2010). "The assessment of automatic weather station operating quality based on fuzzy AHP", *International Conference on Machine Learning and Cybernetics*, Qingdao, China, pp. 1164–1168. DOI: 10.1109/ICMLC.2010.5580916.

[4] Taras Boretskyi (2019). The Methods of Protection and Hacking of Modern Wi-Fi Networks in *Advances in Cyber-Physical Systems*, Vol. 4, No. 1, pp. 1–6. DOI: https://doi.org/10.23939/acps2019.01.001

[5] P. Kapoor and F. A. Barbhuiya (2019). "Cloud Based Weather Station using IoT Devices", TENCON 2019 – 2019 *IEEE Region 10 Conference (TENCON)*, Kochi, India, pp. 2357–2362. DOI: 10.1109/TENCON.2019.8929528.

[6] S.-j. Yang, X.-y. Deng and M.-y. Wang (2012). "Construction of modern education technology web-based course platform based on .NET", *7th International Conference on Computer Science & Education (ICCSE)*, Melbourne, VIC, Australia, 2012, pp. 1702–1705. DOI: 10.1109/ICCSE.2012.6295393.

[7] S. S. Syazlina Mohd Soleh, M. M. Som, M. H. Abd Wahab, A. Mustapha, N. A. Othman and M. Z. Saringat (2018). "Arduino-Based Wireless Motion Detecting System", *IEEE Conference on Open Systems (ICOS)*, Langkawi, Malaysia, pp. 71–75. DOI: 10.1109/ICOS.2018.8632703.

[8] Adepoju, Temilola & Oladele, Matthias & Kasali, Abdulwakil & Fabiyi, Gbenga (2020). Development of a Low-Cost Arduino-Based Weather Station. *FUOYE Journal of Engineering and Technology*. DOI: 5.10.46792/fuoyejet.v5i2.508.

[9] H. Üçgün and Z. K. Kaplan (2017). "Arduino-based weather forecasting station", *International Conference on Computer Science and Engineering (UBMK)*, Antalya, Turkey, pp. 972–977. DOI: 10.1109/UBMK.2017.8093397.

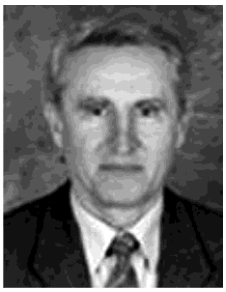
[10] H. Garg and M. Dave (2019). "Securing IoT Devices and SecurelyConnecting the Dots Using REST API and Middleware", *4th International Conference on Internet of Things: Smart Innovation and Usages (IoT–SIU)*, pp. 1–6. DOI: 10.1109/IoT-SIU.2019.8777334.

[11] A. Suryana, F. P. Lismana, R. M. Rachmat, S. D. Putra and M. Artiyasa (2020). "Implementation of Weather Station for The Weather Reality in A Room", 2020 *6th International Conference on Computing Engineering and Design (ICCED)*, Sukabumi, Indonesia, pp. 1–6. DOI: 10.1109/ICCED51276.2020.9415799.

[12] A. S. Bin Shahadat, S. Islam Ayon and M. R. Khatun (2021). "Efficient IoT-based Weather Station", *IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, Bhubaneswar, India, pp. 227–230. DOI: 10.1109/WIECON-ECE52138.2020.9398041.

Bohdan Marii received his B. S. degree in Computer Engineering at Lviv Polytechnic National University, Ukraine, in 2022. His research interests include the architecture, patterns and development of web applications, SQL databases, and programming and technologies for web programming.

Tetyana Pavych Senior Business Analyst at MedBridge Inc. Bachelor's degree in Management Information Systems in 2019 at Northwood University, Midland, MI, USA. Business Career Program graduated in 2021 at Computer Systems Institute, Chicago, IL, USA. Her research is in the CRM management, business intelligence, and data analysis area.



Yaroslav Paramud – Ph. D., Assoc. Prof at Computer Engineering Department of Lviv Polytechnic National University. Scientific interests include processing radar information, research algorithms, and structures of specialized computing devices and systems.