Ministry of Education and Science of Ukraine
Ternopil Ivan Puluj National Technical University

Faculty Of Computer Information Systems and Software Engineering
(Full name of faculty)
Computer science Department
(Full name of department)

QUALIFYING PAPER

For the degree of

bachelor
(Degree name)

topic: Development of the "Smart Office" project based on the Internet of things (IoT) technologies

Submitted by:	fourth year student, group ICH-42
Specialty:	122 Computer science
	(Code and name of specialty)

		Valani Montu Amrutlal
	(signature)	(Surname and initials)
Supervisor		Holotenko O.S.
	(signature)	(Surname and initials)
Standards verified by		
	(signature)	(Surname and initials)
Head of Department		Bodnarchuk I.O.
	(signature)	(Surname and initials)
Reviewer		
	(signature)	(Surname and initials)

Ternopil
2023

Ministry of Education and Science of Ukraine
Ternopil Ivan Puluj National Technical University

Faculty Of Computer Information Systems and Software Engineering
(Full name of faculty)

Department Computer science Department
(Full name of department)

APPROVED BY
Head of Department
_______________ Bodnarchuk I.O.
(signature) (Surname and initials)
« » 2023 р.

ASSIGNMENT

for QUALIFYING PAPER

for the degree of *bachelor*
(Degree name)

specialty *122 Computer science*
(Code and name of the specialty)

student Valani Montu Amrutlal

1. Paper topic: Development of the "Smart Office" project based on the Internet of things (IoT) technologies

Paper supervisor: Holotenko Oleksandr Serhiyovych,, PhD, Assoc. Prof.
(Surname, name, patronymic, scientific degree, academic rank)

Approved by university order as of «___» ________ 2023 №____.
2. Student's paper submission deadline 10/07/2023
3. Initial data for the paper *methods and means of measuring parameters microclimate and fields of application, development environment for Arduino, majordomo management system, MQTT protocol.*
4. Paper contents (list of issues to be developed) *1. Evaluation and examination of projects utilizing internet of things (IOT) technologies. 2. Design of the ecosystem monitoring of the "smart office" information system. 3. Software implementation of the "smart office" information system. 4. Occupational safety and health.*
5. List of graphic material (with exact number of required drawings, slides) 1. *Relevance of the work. 2. The main purpose of work, the object of study, subject of research. 3. Tasks of the work. 4. System components. 5. Subsystem operation algorithm. 6. The basic electrical scheme 7. Functional scheme. 8. Working with the system. 9. Conclusions.*

6. Advisors of paper chapters.

Chapter	Advisor's surname, initials and position	Signature, date	
		assignment was given by	assignment was received by
Occupational safety and health			

7. Date of receiving the assignment.

TIME SCHEDULE

LN	Paper stages	Paper stages deadlines	Notes
	Analysis of technical task		
	Analysis of characteristics of the object		
	IOT technologies analysis		
	Information resources and services		
	Development of structural elements		
	Development of the subsystem interface		
	Testing the interface and the security		
	Occupational safety and health		
	Graphic materials		
	Preparation to the qualification work presentation		
	Qualification work presentation		

Student ________________ Valani Montu Amrutlal
(signature) (surname and initials)

Paper supervisor ________________ Holotenko O.S.
(signature) (surname and initials)

ABSTRACT

Development of the "Smart Office" project based on the Internet of things (IoT) technologies //Qualifying paper // Valani Montu Amrutlal // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, group ICH-42 //Ternopil, 2023 // p. - 61, fig. - 25, tabl.- 2, code snip. - 1, append. - 2, bibliogr. - 15.

Keywords: Ethernet, physical topology, network equipment, switching, routing, sharing services, system, IoT, smart technologies.

In this diploma thesis developed a "Smart Office" project based on the Internet of things (IoT) technologies. Basic documentation has been prepared: subsystem operation algorithm, the basic electrical scheme, functional scheme, ccomparison of data providers. A programming language and appropriate development tools have been chosen, the structural elements of the subsystem and its web interface in configuration mode have been developed. The created subsystem fully satisfies the requirements and successfully accomplishes all assigned tasks.

TABLE OF CONTENT

INTRODUCTION

Currently, there is rapid advancement in advanced technological systems designed to automate household tasks. These systems encompass a wide range of household appliances, devices, and sensors, collectively known as Internet of Things (IoT) devices, which form the foundation of a Smart Home system. This system enables communication between users and their household appliances, expanding automation capabilities, device monitoring, and remote control functionalities.

Similarly, in order to create a more productive and adaptable work environment tailored to specific activities, companies are adopting IoT-based systems known as "Smart Office" solutions. These systems employ modern technologies to enhance employee productivity and optimize space utilization, resulting in efficient and cost-effective operations.

The significance of this work stems from the increasing demand for automated room management systems that can transform existing devices into a smart infrastructure. Efficient management of office space infrastructure is crucial for creating a comfortable and effective working environment, ultimately impacting overall business efficiency.

The objective of this research is to explore IoT technology and develop an information system for implementing the "Smart Office" concept. The goal is to automate and enhance IoT management in office spaces through the implementation of the "Smart Office" information system. The following tasks need to be addressed to achieve this goal:

Conduct a comprehensive literature review and analysis of engineering equipment used in Smart Office systems, IoT concepts, existing market solutions, and technologies for platform development and implementation.

Develop a system implementation plan encompassing the subject area description, project charter, information system life cycle at each development stage, and calendar planning.

Evaluate the effectiveness of the developed system for implementation and integration of the information system.

Compile requirements and describe the functionality of the information system for IoT technology management.

Design the architecture and specifications for the "Smart Office" information system.

The research will employ empirical (comparison, experimentation), theoretical (search, analysis, modeling), and mathematical (calculation) research methods. It will cover fundamental concepts and technologies relevant to the subject matter.

The scientific novelty of this work lies in its contribution to existing Smart Office and IoT solutions. The proposed concept offers the potential for developing systems for proactive management of intelligent office spaces.

The practical significance lies in the economic efficiency, improved usability for end-users, and the promising applications of IoT technologies.

1 EVALUATION AND EXAMINATION OF PROJECTS UTILIZING INTERNET OF THINGS (IOT) TECHNOLOGIES

1.1 Concept of technologies related to the Internet of things

The concept of the Smart Home system revolutionizes the way we approach household activities by integrating a comprehensive automated control system that enhances operational efficiency and the management reliability of all life support systems. A Smart Home is designed to provide security, resource conservation, and comfort for its users. At its core, it should be capable of recognizing specific situations within the building and responding accordingly, with subsystems controlling the behavior of others based on developed algorithms. The automation of multiple subsystems generates a synergistic effect for the entire system.

A Smart Home represents a concept of human interaction with living spaces, where the operation of engineering systems and electrical appliances is regulated automatically from a central hub, following pre-defined parameters. Such a system offers numerous benefits to users, including comfort, safety, and resource efficiency. It ensures coordinated operation of the heating and air conditioning systems while monitoring factors that affect the need for activating or deactivating these systems. In other words, all engineering systems and electrical devices operate and are monitored automatically in accordance with external and internal conditions.

In developed countries, the concept of the Smart Home aligns with environmental friendliness and sustainability, as it facilitates the rational utilization of limited natural resources while minimizing adverse effects on the environment and people's well-being. The primary objective of a Smart Home is to enhance residents' comfort and simplify everyday life.

The history of Smart Home technology dates back to the last century. The term "Smart Home" was initially introduced in the 1970s at the Intelligent Building Institute in the State of Washington, where groundbreaking projects were developed, enabling the transmission of various types of information through a single wire for controlling different devices.

One of the early smart home projects was the house of American engineer Emil Mathias. In the 1950s, Mathias equipped his home with a variety of devices, which he controlled through a button panel. For instance, he could open the garage door and remotely turn the radio on and off with a simple press of a button. Additionally, the engineer installed automatic alarm systems in the house. Implementing the project required over two kilometers of hidden cables, concealing all wires, motors, sensors, and devices within the walls and floors.

Another notable example of smart home development is Monsanto's House of the Future, an attraction introduced in 1957. The Monsanto company collaborated with the Massachusetts Institute of Technology to create a new type of residential building entirely made of plastic. The lightweight nature of plastic enabled the automation of all systems in the house with ease. The house featured an ultrasonic dishwasher, in-floor air conditioning, a speakerphone, shelves that disappeared into the kitchen ceiling, and a connected, automated sink that adjusted its level based on a person's height.

The popularization of the smart home concept gained traction in the late 1990s, with the release of the film "Smart House" by Disney in 1999, which depicted an automated house managed by a robot maid. Since then, the topic has been increasingly discussed in cinema and the media.

In essence, a Smart Home refers to the automation of daily life by integrating electrical appliances and household devices into a unified ecosystem within the home. Let's consider the exemplary architecture of a Smart Home, as shown in Figure 1. When developing a remote control system, the main components typically include user applications, a web server, a Smart Home server, and a controller for managing sensors and actuators.

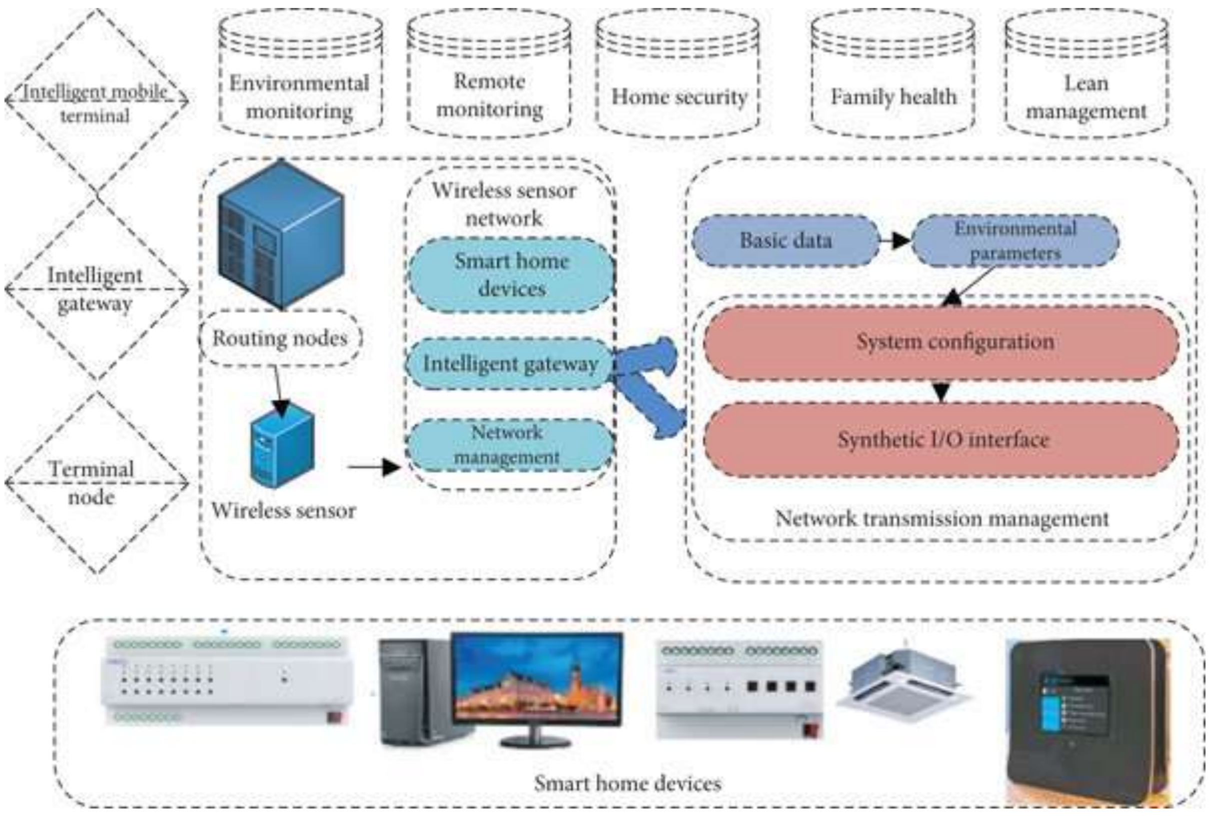


Fig.1 Architecture of the "Smart House"

The Smart Home system encompasses several crucial components that work together seamlessly:

The remote control is facilitated through a mobile application or web browser, enabling users to issue commands to Smart Home devices. This component serves as a conduit for transmitting instructions to the cloud server while also receiving sensor data updates from it.

The web server acts as a data repository, storing information about the status of sensors and devices in a database. Additionally, it acts as an intermediary between remote control devices and the Smart Home server. Its primary function involves transmitting commands from mobile devices to the cloud, where they are processed and subsequently relayed to the home server.

The Smart Home server serves as the central hub for receiving commands from the web server and relaying them to the controllers. It also facilitates the transmission of sensor data from the controllers back to the server.

The controller acts as the linchpin that integrates the various elements of a Smart Home into a cohesive entity. It manages connected sensors, receives commands from the server, and relays information about changes in sensor states back to the server. It plays a pivotal role in providing a web interface, executing scripts, and facilitating command transfer between devices.

Sensors are devices responsible for gathering information about the environment and the state of household appliances, which they subsequently transmit to the controller. For example, a temperature sensor detects the temperature in the house and triggers the activation of a heater or an air conditioner accordingly.

Actuators are executive devices that execute commands directly. This category encompasses a wide range of devices, including smart switches, smart sockets, sirens, climate controllers, and more.

IoT devices, also known as Internet of Things devices, are smart home devices consisting of a controller, a sensor, and a household appliance, such as a refrigerator, thermometer, or blinds.

The Internet of Things (IoT) refers to the vast network of interconnected objects that communicate with each other over the internet, facilitating data exchange and interaction with IoT applications, connected devices, and industrial machinery [30]. Internet-connected devices incorporate built-in sensors to collect and sometimes influence data. IoT applications span a broad spectrum, from smart homes that autonomously adjust heating and lighting to smart factories that monitor industrial machinery, making real-time adjustments to prevent failures.

At its core, the Internet of Things operates on the principle of machine-to-machine communication, enabling electronic devices to interact without human intervention. Unlike "smart" homes, IoT systems utilize TCP/IP protocols to exchange data across the channels of the global internet network. In an article titled "Architecture of a network control complex of a building based on IoT devices", the authors delve into the design task of a cyber-physical system utilized as a service for managing intelligent buildings using IoT technologies.

The Internet of Things comprises a network of physical objects connected to the internet, capable of self-identification and integration with other devices. Thoughtful implementation of IoT can lead to reduced electricity consumption, enhanced safety in homes and cities, and improved indoor comfort. This technology empowers organizations with novel methods to remotely manage and monitor operations, enabling full control over remotely located objects while constantly supplying information to applications and data repositories.

The International Telecommunication Union (ITU) Communication Standards Division has published Recommendation Y.2060, titled "Overview of the Internet of Things" [61]. Within this document, the following definitions provide insights into the scope of the Internet of Things:

Internet of Things: A global infrastructure within the information society that enables the provision of advanced services by integrating (physical and virtual) things based on existing and functionally compatible information and communication technologies.

In the context of the Internet of Things, a "Thing" refers to an entity existing in either the physical world (physical things) or the informational realm (virtual things), possessing the ability to be identified and seamlessly integrated into a communication network.

Regarding the Internet of Things, a "Device" signifies an equipment item equipped with mandatory communication capabilities, along with additional functionalities such as measurement, actuation, data input, storage, and processing.

As depicted in Figure 2, within the realm of Internet of Things technology, the already pervasive information and communication technologies that facilitate communication "anytime" and "anywhere" now encompass a novel dimension - "communication with anything".

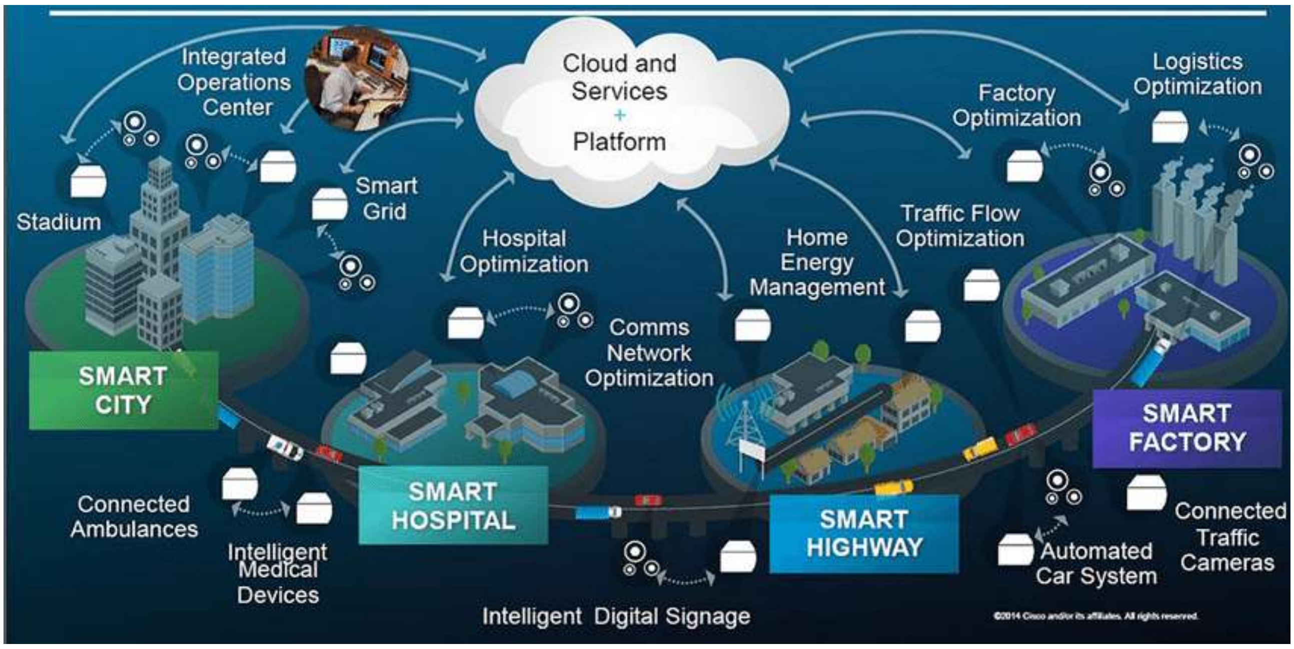

Integrated Operations Center
Cloud and Services + Platform
Factory Optimization
Logistics Optimization
Stadium
Smart Grid
Hospital Optimization
Home Energy Management
Traffic Flow Optimization
Comms Network Optimization
SMART CITY
SMART HOSPITAL
SMART HIGHWAY
SMART FACTORY
Connected Ambulances
Intelligent Medical Devices
Intelligent Digital Signage
Automated Car System
Connected Traffic Cameras
©2014 Cisco and/or its affiliates. All rights reserved.

Fig.2 Communication with anything

In the context of IoT, "Things" refer to objects existing in either the physical world (referred to as physical things) or the digital realm (referred to as virtual things), which can be identified and seamlessly integrated into a communication network. These things possess associated information, which can be either static or dynamic. Physical things exist in the tangible world and can be measured, controlled, and connected. Examples of physical things include the environment, industrial machinery, goods, and electrical equipment. On the other hand, virtual things exist in the digital realm and can be stored, processed, and accessed. Examples of virtual things include multimedia content and application software.

Researchers highlight that governments of various countries are displaying interest in IoT technology and supporting scientists and enterprises engaged in its implementation and development.

The primary challenge with IoT lies in the absence of clear and consistent architectures for building connected automated systems. For instance, a single light switch may employ one level of data encryption, while a remote control may utilize different encryption algorithms. Moreover, wireless protocols can vary, with devices using ZigBee, Bluetooth, or Wi-Fi. This leads to duplication of bridges for connection across these parameters, forming a network that becomes vulnerable to external influences.

Regarding IoT security in the household industry, the situation is relatively less complex. Smart home devices typically access the internet through standard channels, and the information they transmit requires basic protection against viruses and cyber threats. Even if a massive cyberattack occurs, it would not lead to a catastrophe. Additionally, home devices are often produced in large quantities, and excessive protection would significantly increase their cost, discouraging potential buyers.

In contrast, the situation becomes more severe in business and industry, as attackers may attempt to gain access to smart devices within enterprises, posing the risk of significant financial losses and potential industrial disasters. Hence, IoT networks prioritize enhanced security measures.

Various methods are employed for protection, including:

Isolation from the internet, limiting operations to the internal network of the office or factory.

Encryption of industrial data using robust algorithms like AES-256, which requires an astronomical amount of time to break the 256-bit key.

Protection of software and operating systems.

Intrusion detection and prevention systems that notify administrators if any unauthorized attempts to access the system are detected.

Presently, global digitization of business processes is underway, aiming to enhance labor productivity, work quality, and production safety. Digitization represents a modern trend, with significant efforts being devoted to improving and automating workspaces. Data received from various sources is being used for the development of management systems, economies, businesses, and the social sphere, fostering harmonious interaction between these domains.

A smart office refers to an automated control system designed to manage and control various aspects of premises, including lighting, heating, ventilation, water supply, security, and audio/video equipment. The system can be customized and modified based on the owner's preferences. By integrating various subsystems, a smart office ensures their seamless operation and high functionality within the entire complex. This integration not only prevents conflicts during operation but also facilitates harmonious interaction among the subsystems. For instance, the air conditioner will not cool the room when the heating is already active.

Through the integration of the aforementioned technologies, indoor comfort is ensured. A smart office takes charge of managing the parameters of the entire premises complex. It can control lighting, curtains, air conditioning, heated floors, video equipment, and other devices as required.

Currently, the market for smart technologies exhibits the following characteristics:

A significant portion of solutions available in the market are specifically tailored for intelligent office environments. Experts assert that implementing these solutions not only enhances operational efficiency but also proves to be a profitable investment in the long run.

There is an observed increase in the adoption of smart technologies during the construction stage of office buildings in urban areas, highlighting a growing trend towards intellectualization. These solutions are characterized by their complexity, modularity, and the ability to be personally configured.

In general, the smart infrastructure of a building comprises two key components: IT infrastructure and smart engineering systems (refer to Fig. 3). The IT infrastructure encompasses all the information technologies and resources utilized by a particular organization or company. On the other hand, intelligent engineering systems encompass the building or premises systems that are designed to support life, facilitate technological processes, maintain comfort, conserve energy and resources, and ensure security.

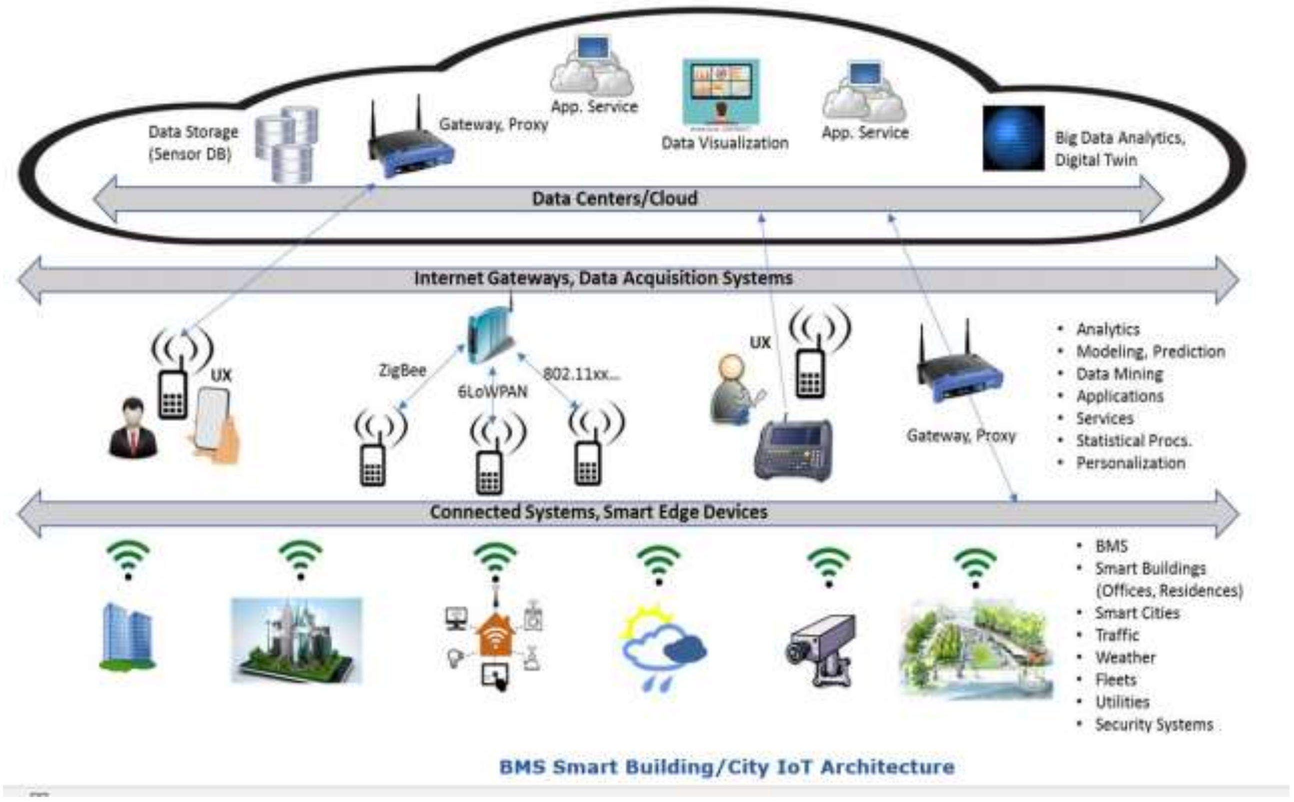


Fig. 3 Smart building architecture.

Smart infrastructure offers clients the convenience of accessing necessary services while adhering to security requirements, eliminating the need for direct contact with service providers.

The implementation of a smart office system brings several key advantages:

Cost reduction: The "smart office" system enables significant cost savings by reducing electricity consumption by up to 50% and optimizing water and heating supply. This, in turn, leads to reduced rental rates and operating expenses.

Increased productivity: Simplified tasks such as finding and booking meeting rooms streamline workflows, allowing employees to dedicate more time to productive work and less time on non-work activities.

Enhanced comfort: The system ensures a comfortable working environment by regulating temperature, humidity, and fresh air circulation both inside and outside the premises.

Improved security: The Smart Office security system encompasses various comfort functions, including security and fire alarms, access control, monitoring of engineering subsystems, and video surveillance.

Optimal utilization of office space: Implementing a smart office solution optimizes the use of workspace, freeing up areas and maximizing their efficiency.

1.2 Standardization of Internet of Things (IoT) systems

The Internet of Things gained significant popularity between 2010 and 2017 with the advent of affordable terminal devices, secure and fast data transmission technologies, and the rapid development of the Internet and communication protocols. These factors led to the emergence of numerous cost-effective and straightforward end devices that could be quickly assembled into ready-made solutions. Alongside simpler solutions, the market saw the introduction of professional systems with complex and extensive logic, involving entire companies and international committees (e.g., the Concept of Industry 4.0 and the Industrial Internet Consortium).

While some researchers argue that a single software architecture standard for IoT applications may not be feasible due to the lack of contextual information for design decisions [60], the Industrial Internet Consortium developed a methodology in 2015 for Industrial IoT systems. This methodology includes the creation of design support tools for specific types of IoT systems. Notably, the publication of established standards governing construction levels, complexity, technical details, and implementation features of such systems in industrial settings. The document has been widely adopted by Western companies involved in the development of industrial IoT systems, demonstrating its practical applicability.

Intellectual buildings possess distinct characteristics outlined by several authors:

The ability for dynamic development of engineering systems, allowing for expansion and modification.

A significant number of sensors to gather operational information about the building's condition.

The software and hardware components should not be restricted to a single manufacturer.

Utilization of typical devices such as controllers, communication buses, input-output modules, and information display systems.

Based on these properties, it is imperative to integrate all security systems and engineering systems within an intelligent building into a unified information system, known as an automated building management system.

Recommendation Y.2060 also provides an IoT reference model that closely resembles the NGN (Next Generation Network) model. It consists of four primary horizontal layers (refer to Figure 4):

- IoT application layer
- Support layer for services and applications
- Network layer
- Device layer

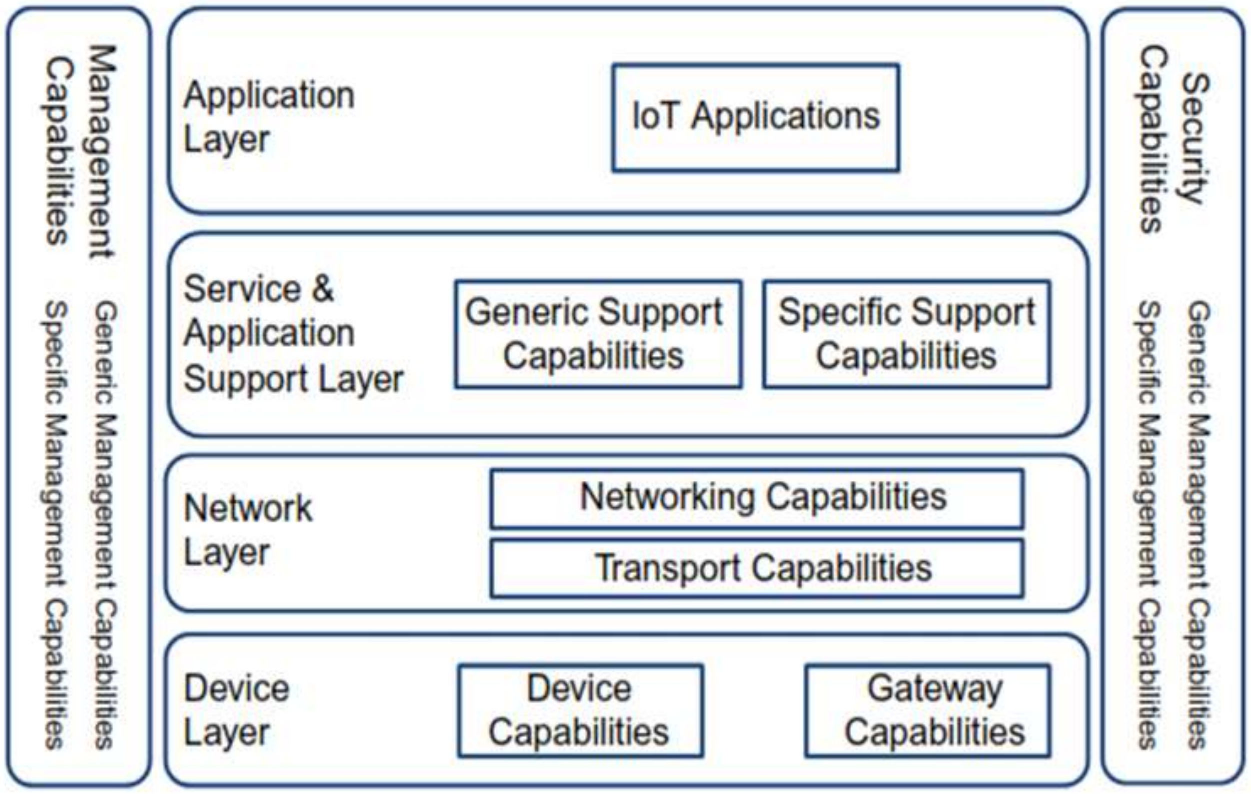

Management Capabilities
Generic Management Capabilities
Specific Management Capabilities
Application Layer
IoT Applications
Service & Application Support Layer
Generic Support Capabilities
Specific Support Capabilities
Network Layer
Networking Capabilities
Transport Capabilities
Device Layer
Device Capabilities
Gateway Capabilities
Security Capabilities
Generic Management Capabilities
Specific Management Capabilities

Fig. 4 IOT reference model

There are two additional vertical levels in the architecture - the management level and the security level, encompassing all four horizontal levels. Recommendation Y.2060 does not provide detailed coverage of the IoT application layer. The level of support for services and programs includes general capabilities for various IoT objects for data processing and storage. The network layer incorporates network functions such as resource management of the access network and transport network, mobility management, authorization, authentication, and settlement functions. Lastly, the device layer represents the capabilities of the device and the gateway. Device functionality includes direct communication with the network, communication through the gateway, sharing via wireless dynamic ad hoc networks, and temporary suspension and resumption of device operations for energy-saving purposes. The gateway supports multiple device interfaces (CAN bus, ZigBee, Bluetooth, WiFi) as well as access networks/transport networks (2G/3G, LTE, DSL). Additionally, the gateway facilitates protocol conversion if the protocols of device and network interfaces differ.

Within the architecture of the information management system, two intellectual information models can be identified. The first model pertains to the level of sensors and executive mechanisms, encompassing sensors, controllers, and a database for storing information processed by the controller. This model enables automatic control through the controller, which retrieves information from the sensors, compares it with threshold values, and issues control commands to adjust the parameters when necessary. Real-time data received from the sensors is stored in a database. The second information model is the logic model of the intelligent building, comprising the interface, business logic, database, and communication elements. This model facilitates user interaction with devices. Web services and SCADA systems are utilized to display and control information from sensors and devices. To transfer this data to the server, protocols are required. Existing protocols based on HTTP are not suitable for the concept of IoT and machine-to-machine interaction. Therefore, a new protocol, MQTT (Message Queue Telemetry Transport), was developed.

MQTT, or Message Queuing Telemetry Transport, is an open data exchange protocol. In this protocol, a broker collects data from multiple nodes and transmits it to the server. It follows a publisher-subscriber model with an intermediate server acting as the broker. The transport used is TCP. The MQTT protocol necessitates the presence of a data broker. All devices send data exclusively to the broker and receive data from it. The broker serves as a program that functions as a TCP server with a dynamic database. The protocol was designed to be open, simple, with minimal resource requirements, and easy to implement. MQTT operates on top of TCP/IP and adopts a client/server model, where each sensor functions as a client connected to a server acting as a broker. The MQTT protocol mandates the presence of a broker to manage data distribution to subscribers. All devices or actuators only send data to the broker and receive data from it as well.

In an MQTT-based network, the following objects are distinguished:

Publisher: An MQTT client that publishes relevant topics to the broker when a specific event occurs.

Broker: An MQTT server that receives information from publishers and transmits it to relevant subscribers. In complex systems, the broker may also perform various operations related to the analysis and processing of received data. Different brokers can connect with each other by subscribing to each other's messages.

Subscriber: An MQTT client that subscribes to the broker, constantly listening for and processing incoming messages on topics of interest.

The diagram illustrating the simple interaction between the subscriber, publisher, and broker is presented in Figure 5.

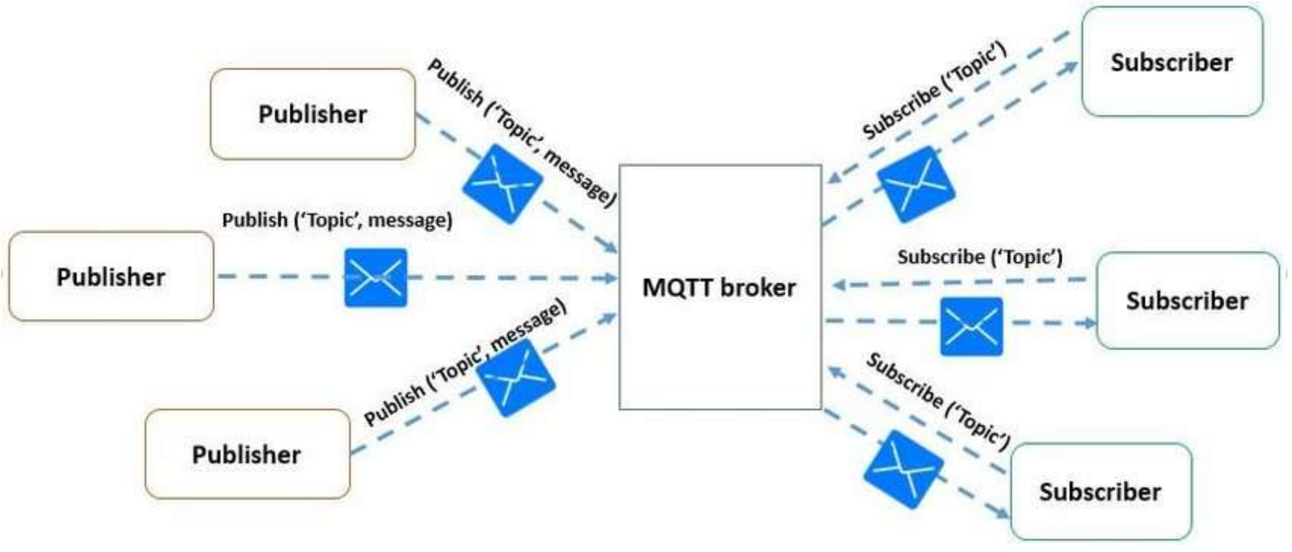


Fig. 5 The main structure of the MQTT broker

Temperature sensors act as "publishers" in the MQTT system, as they can only transmit information about their state and cannot accept commands. On the other hand, the computer acts as a "subscriber" because it receives data from the MQTT broker regarding the state of the sensors.

In analytical practice, time series are commonly encountered. A time series refers to a sequence of observations of the same phenomenon or a parameter of a process over a period of time. Each measurement in a time series corresponds to a specific time or a serial number based on the time scale.

InfluxDB is a clustering database designed specifically for storing time series and metrics. It is written in Go and can be compiled into a single binary without external dependencies. InfluxDB offers users the ability to store data and interact with the database through the command line.

The advantages of InfluxDB include:

- Lack of dependencies
- Capability to work in cluster mode
- Ability to store billions of measurement points
- Efficient data sampling through classification using tags
- Availability of libraries for various programming languages
- SQL-like query language for performing operations on time series

InfluxDB is primarily designed to store DevOps monitoring metrics, sensor data, and data for online time series analytics. Data in InfluxDB is represented as time series, which consist of measured values. A series is a collection of "points" that represent values at specific points in time. The "Points" structure includes fields such as:

- Time: A timestamp that stores time information
- Measurement: A line indicating the series
- Fields: Direct values in a "key-value" format
- Tags: Meta-data about values, also in a "key-value" format

Conceptually, a measurement in InfluxDB can be likened to a table in SQL, where the primary index is always time, and tags and fields serve as columns. Tags are indexed, while fields are not. The advantage is that InfluxDB can handle millions of measurements without requiring predefined schemas. Data is written to InfluxDB using the line protocol.

When analyzing existing systems for IoT, the concept of a "Smart Office" can be divided into two components: technological and conceptual. Technologically, a "Smart Office" is similar to a "Smart Home". Modern advancements allow for integrating the entire engineering infrastructure of a business center into a single management system that controls access, energy supply, heating, water supply, and fire alarm.

For tenants or buyers, the operation of office real estate plays a secondary role. Their main concern is the financial costs associated with maintaining the premises and ways to minimize them. The "Smart Office" system helps reduce electricity costs by 50% and achieve 30-40% savings in water and heat supply. As a result, both rental rates and operating costs are reduced.

The "Smart Office" Estel in Milan, Italy, is a software and hardware complex that enables automatic control of systems and devices. It incorporates a "Smart Office" controller integrated into subscriber devices, allowing users to create a network of wireless sensors controlled through a smartphone, tablet, or web browser (Figure 6).

Fig. 6 The composition of the solution "Smart Office" Estel

Features of the "Smart Office" system include:

- Ensuring premises security: The system includes remote monitoring of the premises through Wi-Fi cameras.
- Remote control and management of system sensors: The installed application receives notifications regarding unauthorized door opening, water leakage, smoke detection, and messages from motion sensors.
- Programmable equipment: All equipment can be programmed to work sequentially or simultaneously in the desired mode with specific time and date settings.
- Resource consumption monitoring: The system includes sensors to track resource consumption for optimal use of supply services.

Additional features of the system are:

Microclimate management: The system allows regulation of temperature and humidity in the room. When the optimal microclimate is reached, the power of heaters or air conditioners is automatically adjusted.

Lighting systems and electrical appliances management: The system automatically turns off lights when the natural lighting in the room reaches an optimal level.

The "Smart Office" system provides an integrated solution that addresses important security tasks. Let's explore the advantages of using the system in organizations in more detail.

Leakage control is a feature that helps in unexpected pipe bursts. When the sensor detects water, it sends a notification to the owner's smartphone and activates a sound or light signal. Future leakage sensors will be capable of automatically shutting off the water supply in case of an emergency. In new buildings, sensors can monitor water or heat meters, upload readings to smartphones, and issue alerts if the consumption exceeds average monthly volumes.

Enhanced security is achieved through opening sensors placed on doors and windows to monitor their status. These sensors are beneficial for office security, as they send notifications to smartphones, activate loud sirens, or alert security agencies if an unauthorized person enters. Opening sensors also record temperature and lighting conditions in the room. Motion sensors notify users if someone is present in the house when they are away, triggering alerts, surveillance cameras, and providing real-time updates. Smoke sensors emit sound signals and communicate with the owner's smartphone.

Outdoor and indoor video surveillance is facilitated by easily installable cameras that contribute to increased security. The camera's video stream can be viewed through a smartphone app or a personal account on a PC. All captured footage is stored in cloud storage.

The central component of the entire "Smart Office" ecosystem is a controller. This device connects all sensors, cameras, and light bulbs in the office, enabling control and coordination among them. The controller allows for the creation of interaction scenarios between devices.

Microsoft Technology Center, with 35 employees worldwide, serves as a testing ground for Microsoft products' implementation scenarios. It conducts training seminars, briefings for partners, as well as conferences and presentations. The center is based on the architecture of IoT solutions utilizing the Azure IoT Suite service announced by Microsoft.

Microsoft Azure is a cloud platform that combines infrastructure-as-a-service (IaaS) solutions (servers, data storage, networks, operating systems) with a range of tools and services that facilitate the development and deployment of cloud applications (platform-as-a-service or PaaS). These solutions enable developers to quickly create, deploy, and manage large-scale applications tailored to the specific needs of organizations. Microsoft Azure services help customers avoid complexities and additional costs associated with replacing existing equipment and managing infrastructure, leading to more effective IT budget management. Customers only pay or the resources they require.

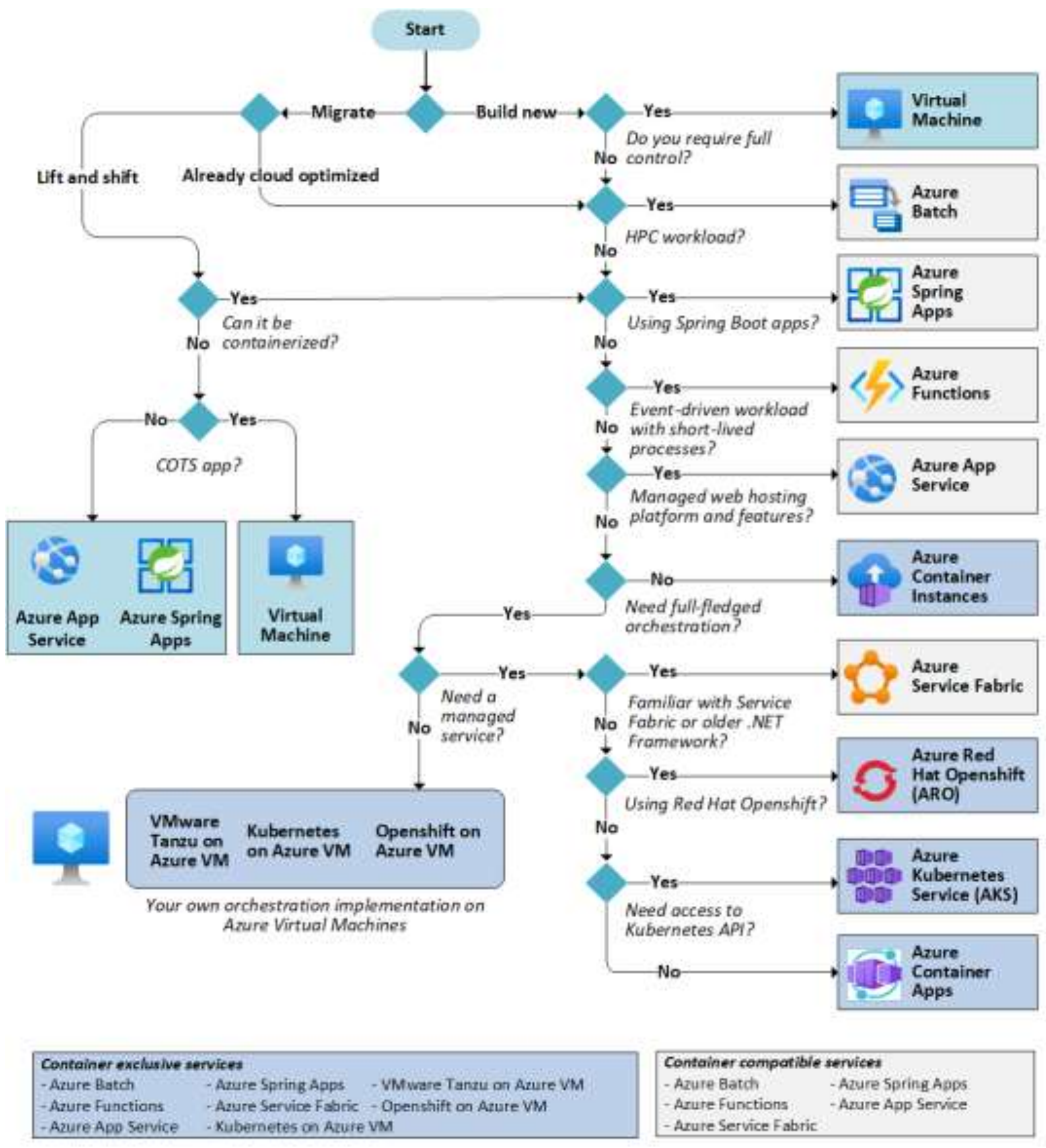


Fig. 7 Microsoft technology center architecture

The infrastructure's operation is monitored using numerous sensors and actuators. The sensor categories serving as information providers include:

Server Display Room sensors: These sensors monitor parameters such as temperature, humidity, power supply, cooling, and video surveillance in the server display room.

Lighting management: Sensors are employed to manage and control the lighting systems.

Audio-video and presentation equipment management: Sensors are utilized to control and manage audio-video and presentation equipment.

Schedule, configuration, and loading of premises management: Sensors play a role in managing the scheduling, configuration, and loading of premises.

Climate management: Sensors are used to monitor and regulate the climate within the infrastructure.

Access management: Sensors contribute to the management of access control within the premises.

Increasing business productivity and unified communications: Sensors aid in enhancing business productivity and facilitating unified communications.

On March 16, 2015, Microsoft Corporation announced the Azure IoT Suite, a cloud-based solution designed to assist corporate users in adapting to the expanding Internet of Things (IoT) landscape. This solution comprises a collection of cloud services integrated with the Azure environment, enabling companies to initiate their own IoT projects. Azure IoT Suite facilitates the connection of diverse electronic products with network capabilities to the Microsoft Azure cloud. It enables the retrieval, management, storage, and analysis of data from these connected devices, thereby enabling important business decisions and process automation.

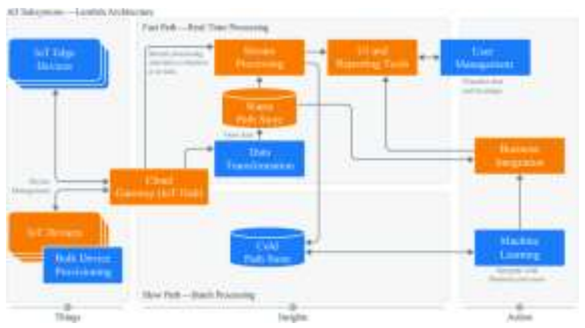


Fig. 8 Azure IoT Reference Architecture

The Microsoft Azure IoT Suite is a comprehensive solution designed for enterprise use, offering preconfigured and extensible solutions that enable rapid deployment. These solutions are suitable for common IoT scenarios such as remote monitoring and predictive maintenance. The preconfigured solutions provided within the Azure IoT Suite include the following components:

Virtual devices: These are preconfigured virtual devices that are essential for starting work within the IoT environment.

Preconfigured Azure services: The suite includes preconfigured Azure services that seamlessly integrate with the IoT solutions.

Control console: Each solution comes with a control console specifically designed for that particular solution.

These preconfigured solutions come with tested and ready-to-use code that can be customized and extended to suit specific IoT scenarios.

At the core of the Azure IoT Suite is the Azure IoT Hub service, which acts as a messaging platform between devices and the cloud. It serves as a gateway to the cloud and other key services within the IoT Suite. The IoT Hub enables real-time message reception from devices, command delivery to devices, and device management.

Azure Stream Analytics is used for operational data analysis within the preconfigured solutions. It handles telemetry input processing, aggregation, and event detection. Stream Analytics is utilized to process informational messages containing metadata or responses to device commands. It helps process device messages and facilitates delivery to other services.

Data storage capabilities are provided by the Azure Document DB service. For telemetry data storage and preparation for analysis, preconfigured solutions utilize Blob storage. Document DB is used to store device metadata and also provides device management functionalities within the solutions.

Data visualization is achieved through Microsoft Power BI web applications. Power BI offers flexibility in creating interactive monitoring dashboards based on data from the Azure IoT Suite.

In addition to Azure IoT Suite, there is another platform called AWS IoT Core, which is a managed cloud platform provided by Amazon Web Services (AWS) for working with IoT devices. AWS IoT Core supports billions of devices and trillions of messages, making it easy to connect devices to the cloud or establish device-to-device connections. AWS IoT Core offers a wide range of choices for operating systems, programming languages, internet application platforms, databases, and other necessary services.

Key features of AWS IoT Core include the ability to filter and transform device data, the preservation of the last state of devices, comprehensive infrastructure security, and integration with various AWS and Amazon services such as AWS Lambda, Amazon Kinesis, Amazon S3, Amazon SageMaker, Amazon DynamoDB, Amazon

CloudWatch, AWS CloudTrail, Amazon QuickSight, and Alexa Voice Service. AWS IoT Core simplifies the development of IoT applications for data collection, processing, analysis, and automation without the need to manage underlying infrastructure.

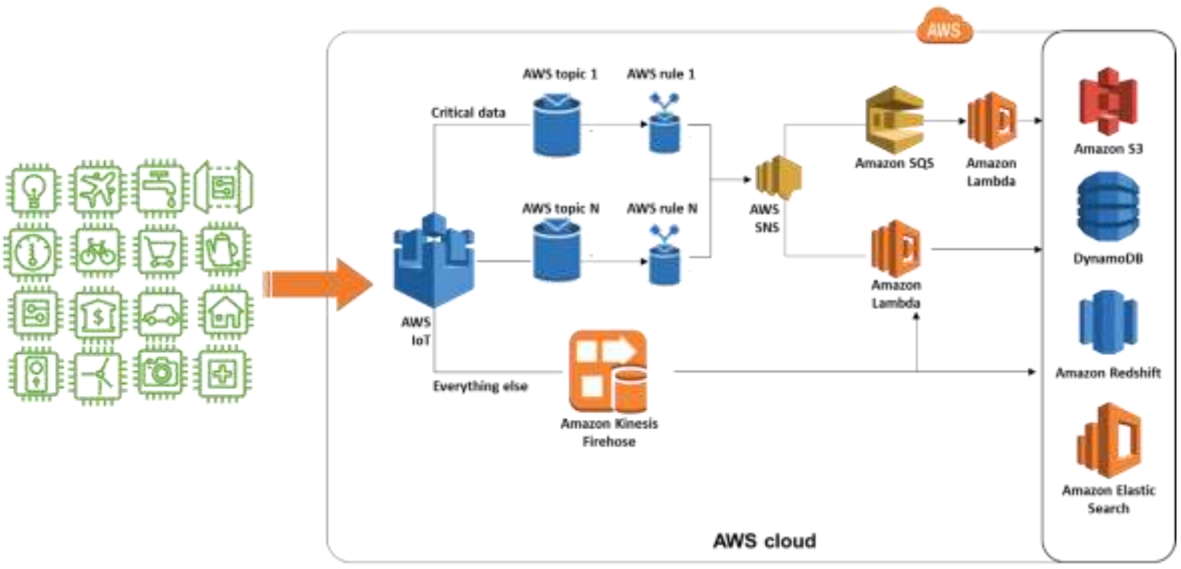


Fig. 9 AWS IoT Core Architecture

AWS IoT Core offers the flexibility to choose the most suitable connection protocol for connecting and enabling IoT devices based on specific application requirements. The supported connection protocols in AWS IoT Core include:

MQTT (Message Queuing Telemetry Transport): This protocol is designed for high-speed telemetry and efficient communication between devices and the cloud.

HTTPS (Hypertext Transfer Protocol Secure): HTTPS provides advanced security features for secure communication between devices and the cloud.

MQTT over WSS (WebSocket's with advanced security): This protocol combines the benefits of MQTT and WebSocket's, allowing for secure and bidirectional communication between devices and the cloud.

LoraWAN: This protocol is ideal for power-efficient, long-range, and low-speed communication in IoT applications that require extended coverage.

The Appliance Gateway serves as the entry point for IoT appliances connecting to AWS IoT Core. It implements the semantics of various protocols, including MQTT, WebSocket's, and HTTP 1.1, to ensure reliable and efficient communication between the appliances and AWS IoT Core. The gateway handles bidirectional communication, enabling appliances to send and receive notifications throughout the day.

The Appliance Gateway is designed to scale automatically and can accommodate a large number of appliances, even exceeding a billion, without the need for additional infrastructure. For customers migrating their data to AWS IoT, the Appliance Gateway provides a secure infrastructure migration capability with minimal impact on the existing architecture and IoT appliances.

2 DESIGN OF THE ECOSYSTEM MONITORING OF THE "SMART OFFICE" INFORMATION SYSTEM

2.1 Selection of element base

To establish a subsystem that can gather and accumulate information within the IT ecosystem of the company, the ESP8266 microcontroller was selected. This choice was made due to its comprehensive functionalities and its ability to remain technologically relevant. The ESP8266 microcontroller, specifically the MK ESP8266, stands out as the optimal option because of its fast operation, ample program memory, support for various popular interfaces, and built-in wireless WiFi module. During the development phase, communication with the MK occurs through a USB to Serial converter, which facilitates code loading into the MK's memory and enables debugging of its performance.

The ESP8266 is highly regarded for its integration capabilities in WiFi applications. It incorporates numerous components that are typically external in competing solutions. This streamlined design with fewer components leads to lower component costs, reduced soldering expenses, a smaller footprint, and decreased printed circuit board expenses.

To ensure the complete functionality of the information gathering and accumulation subsystem within the company's IT ecosystem, it is essential to consider compatible data providers that can be connected to the system.

For enhanced convenience when working with the ESP8266, the NodeMCU debugging board, depicted in Figure 2.1, was utilized. This board primarily consists of the aforementioned MK and incorporates all the necessary circuitry for the reliable operation of the ESP8266.

NodeMCU Lua V3 ESP8266 serves as a developer board based on the ESP8266 chip, which functions as a UART-WiFi module with extremely low power consumption. This board facilitates the convenient development of Internet-enabled devices as it features pre-existing USB connections, a power regulator, and interfaces for data exchange.

NodeMCU was developed shortly after the release of the ESP8266. Espressif Systems initiated the production of the ESP8266 on December 30, 2013.

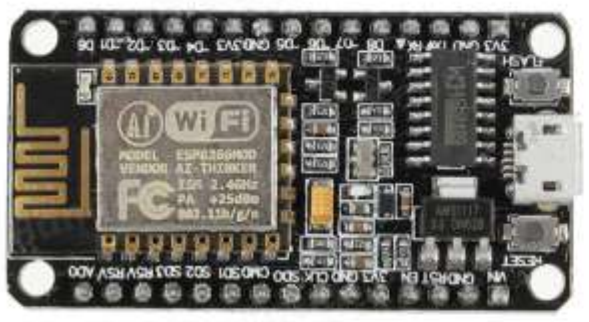


Fig. 10 - NodeMCU development board

The ESP8266, an integrated Wi-Fi SoC (System-on-a-Chip) with the Ten silica Extensa LX106 core, is extensively used in IoT applications. On October 13, 2014, NodeMCU was created, and two months later, it expanded to support the IoT MQTT protocol using Lua to access the MQTT broker.

The ESP8266 microcontroller offers compatibility with popular interfaces such as SPI and I²C. For the purpose of working with these interfaces, the following sensors were selected due to their good characteristics and compatibility:

- DS18B20 temperature sensor
- DHT11/21/22 air humidity and temperature sensor
- BMP180 atmospheric pressure sensor
- BME280 atmospheric pressure sensor
- MH-Z19 carbon dioxide level transmitter
- MQ-135 air quality sensor

2.2 Designing the subsystem structure

When implementing monitoring systems, it is important to provide users with a convenient interface. The ESP8266 enables the creation of a web server, which can offer a user or system administrator an accessible web interface. User interaction with the system begins once the device is successfully assembled into a unified system. To facilitate this, an assembly algorithm is developed, which is presented in Appendix A. This algorithm outlines the sequence and mandatory steps that the system user (or administrator) needs to perform to initiate successful operation with the system.

Before commencing component installation, it is necessary to verify the availability of all required components and establish a workspace dedicated to device assembly. Workspace organization involves checking the availability of necessary tools and acquainting oneself with the assembly algorithm. Once all the tools are ready, the subsystem for collecting and accumulating information flows from the company's IT ecosystem can be assembled:

Ensure that all system components are present.

If all components are available, start assembling the device by placing the ESP8266 microcontroller on the breadboard.

Install the power supply board, which includes a voltage stabilizer capable of selecting a voltage within the range of 3.3V to 5V, according to the mock-up plan.

Connect the appropriate contacts of the microcontroller to the power board and verify its functionality (successful activation is indicated by an illuminated LED).

Connect the data providers to the subsystem by determining the parameters of the ecosystem that need to be collected and accumulated.

To establish communication between the subsystem and a computer, connect them via a USB to Serial converter and use the Arduino IDE development environment to download the firmware into the microcontroller's memory.

If the download is successful, an access point will be created, allowing you to begin working with the system. If not, re-flash the microcontroller.

After connecting to the access point created by the subsystem, using the password defined in the code, you can proceed with the settings.

During the setup process, specify the global network access parameters by entering the name and password to connect to the router, and activate the data transmitters connected to the subsystem.

The final stage involves setting the data export interval to the cloud storage and specifying the API key for accessing the cloud data storage.

If the device assembly algorithm is executed correctly and the system setup is successful, the export of data from the connected sensors will commence. The data will be sent to the selected cloud data storage based on the protocol defined in the code.

2.3 Development of the basic information flow collection subsystem diagram

Once the appropriate components have been chosen to fulfill the specified requirements, it is necessary to construct a schematic diagram of the subsystem based on the selected element base (as depicted in Figure 2.2). The diagram features the ESP8266 microcontroller (ESP-12E version) denoted as DD1, to which various sensors required for monitoring the company's IT ecosystem are connected. The power supply is provided by a 3.3V source. To enable the input/output lines to which the data transmitters are connected, resistors with a nominal value of 10 kΩ are required. These resistors are connected to each output of the monitoring module and the power line.

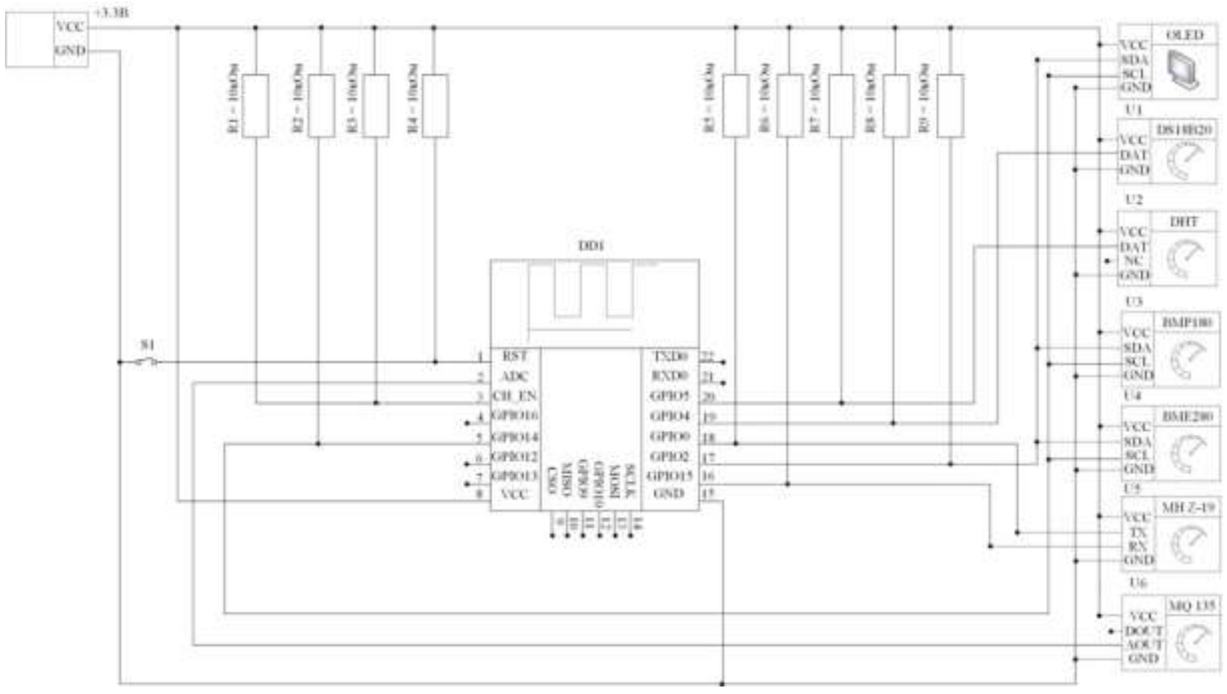


Fig. 11 – Scheme of the electrical principle of the subsystem

The ESP8266 ESP-12E subsystem module provides 10 pins for connecting data sensors and other components. In this implementation, 8 digital outputs are utilized to connect data transmitters. Referring to the schematic diagram, it can be observed that the subsystem currently supports simultaneous operation with various data providers, including:

DS18B20 temperature sensor, labeled as U1 on the diagram.

DHT11/21/22 air humidity and temperature sensors, labeled as U2 on the diagram.

BMP180 temperature and atmospheric pressure sensor, labeled as U3 on the diagram.

BME280 temperature, humidity, and atmospheric pressure sensor, labeled as U4 on the diagram.

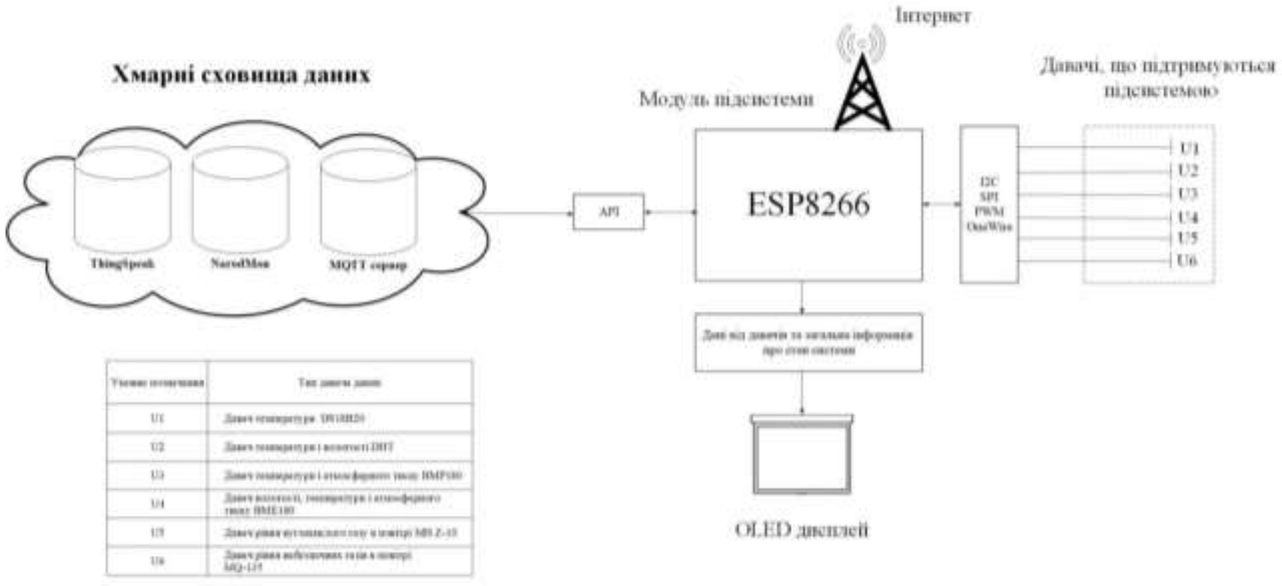
MH-Z19 carbon dioxide level sensor, labeled as U5 on the diagram.

MQ135 air quality sensor, labeled as U6 on the diagram.

Additionally, the circuit includes a reset button for the monitoring module, labeled as S1. This button may be necessary to implement changes after the user modifies the subsystem settings. By connecting the sensors to the subsystem following this schematic diagram, proper system functionality is ensured, safeguarding both the subsystem and the data sensors from damage.

2.4 Designing subsystem behavior

The behavior of the subsystem for collecting and accumulating information flows from the ecosystem depends on the actions of the end user. To illustrate the subsystem's behavior, a functional diagram is the most convenient way to depict its operation and the interaction between individual elements. The functional scheme of the subsystem is presented in Figure 12.



Умовне позначення	Тип давача даних
U1	Давач температури DS18B20
U2	Давач температури і вологості DHT
U3	Давач температури і атмосферного тиску BMP180
U4	Давач вологості, температури і атмосферного тиску BME180
U5	Давач рівня вуглекислого газу в повітрі MH-Z-19
U6	Давач рівня небезпечних газів в повітрі MQ-135

Fig. 12 – Functional diagram of subsystem operation

The functional scheme clearly demonstrates that it accurately represents the intended set of functions for the subsystem. Once connected to the access point created by the subsystem and configured appropriately, it initiates the export of data to the chosen cloud data storage. Users have the option to select ThingSpeak, NarodMon, or export data using the MQTT protocol. Among these options, ThingSpeak is the most popular choice due to its extensive capabilities for data storage and manipulation.

In this particular version of the subsystem for collecting and accumulating information flows from the ecosystem, it enables monitoring of temperature, humidity, atmospheric pressure, and carbon dioxide levels in the air. In the future, there may be opportunities to expand its functionality to include monitoring of water, power consumption of connected devices, and other parameters.

3. SOFTWARE IMPLEMENTATION OF THE "SMART OFFICE" INFORMATION SYSTEM

3.1 Selection of programming language and development tools

To practically implement the monitoring of an IT company's ecosystem, the Arduino IDE was selected as the development environment, and the ESP8266 ESP-12E microcontroller was chosen as the hardware platform. The subsystem's interface is implemented through a web page, designed to be user-friendly and not requiring extensive programming knowledge.

The most popular and freely available development environments for the ESP8266 platform are ESPLoer and Arduino IDE. The main difference between these environments lies in the list of compatible microcontrollers and the programming languages they support. Many developers opt for Arduino IDE due to its user-friendly nature and convenience for writing code, as it utilizes the widely-used open microcontroller programming language, Processing. On the other hand, ESPLoer requires the Lua scripting language and is compatible with microcontrollers that support it.

Throughout the development of the ecosystem monitoring system, various tools and development components were utilized, including:

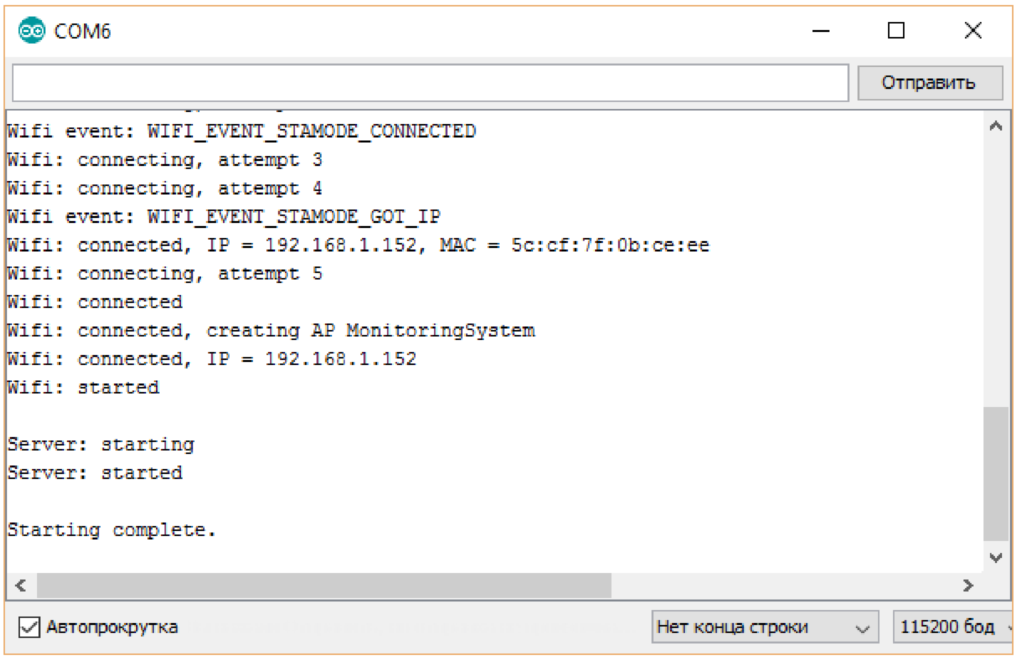
Processing: An open-source programming language based on Java, designed for creating images, animations, and interfaces. It provides a lightweight and efficient toolkit.

Lua: A fast and compact scripting programming language with open-source code written in C. Lua is commonly embedded as a scripting language in various projects. It combines a simple procedural syntax with powerful data description capabilities, utilizing associative arrays and extensible language semantics. Lua employs dynamic typing and translates language constructs into bytecode that runs on a register-based virtual machine with an automatic garbage collector. The interpreter, written in C, can be easily integrated into projects in C and C++ languages. Lua is distributed under the MIT license.

Arduino IDE: A software development environment specifically designed for microcontrollers from the Arduino Uno, Leonardo, Mega, and other families. It can be used to create autonomous interactive objects or to connect with software running on a computer, such as Adobe Flash, Processing, Max/MSP, Pure Data, and SuperCollider.

The ESP8266 chip-based board features Tensilica's L106 Diamond series 32-bit processor. Some boards include a +5V or +3.3V linear voltage regulator. The chip supports interfaces such as SPI, I²C, I²S, and UART, and it incorporates a 10-bit ADC.

At a conceptual level, all boards can be programmed via USB, thanks to the FTDI FT232R USB-to-Serial converter chip. In the NodeMCU platform version, the board can be directly programmed by connecting it to a computer via USB, eliminating the need for external USB-to-Serial converters. By default, the ESP8266 microcontroller board does not have an onboard USB-to-Serial converter, so a separate converter board must be connected to program it. The Arduino IDE development environment is utilized for programming these boards. For debugging purposes, the COM port monitor shown in Figure 13 is employed, allowing the microcontroller to be connected and facilitating the observation of service messages to troubleshoot and resolve any issues if necessary.



```
Wifi event: WIFI_EVENT_STAMODE_CONNECTED
Wifi: connecting, attempt 3
Wifi: connecting, attempt 4
Wifi event: WIFI_EVENT_STAMODE_GOT_IP
Wifi: connected, IP = 192.168.1.152, MAC = 5c:cf:7f:0b:ce:ee
Wifi: connecting, attempt 5
Wifi: connected
Wifi: connected, creating AP MonitoringSystem
Wifi: connected, IP = 192.168.1.152
Wifi: started

Server: starting
Server: started

Starting complete.
```

Fig. 13 - COM port monitor

Boards based on the ESP8266 microcontroller offer the advantage of utilizing a significant number of I/O pins for external circuits. The ESP8266 microcontroller is renowned for its high integration, as it incorporates multiple elements on a single chip, eliminating the need for additional components on the board and reducing costs compared to other systems.

The development of modern ecosystem monitoring subsystems entails the creation of various components and modules that are essential for fulfilling their intended functions, tailored to meet the needs of the target audience. Selecting the appropriate components and modules is a crucial stage in the product creation process. Without functional components and modules, the subsystem would be unable to provide the necessary information to its users.

3.2 Development of structural elements

The primary element in ecosystem monitoring is the user interface, which becomes accessible after connecting to the system. It serves as the platform for all system manipulations and settings.

The subsystem can be divided into the following structural modules:

Wireless network initialization module.

Web server startup module (refer to listing 3.2).

Data transmitter initialization modules.

Display module for data visualization.

Data export module to cloud storage.

The wireless network initialization module functions by creating an access point through which the end user or system administrator can configure the system. This module employs the "ESP8266WebServer" library, which provides the necessary functions for web server operation. By creating a wireless access point, users can connect to it and perform the required actions. The resulting output is then sent via the HTTP protocol to the web browser, generating the web page. The entire process is displayed on the connected OLED display, allowing users to monitor the subsystem's status at any time.

The web server creation module is responsible for initializing the web interface after successful user connection to the access point.

Listing 3.2 is a fragment of the web server creation module

```
void initWebServer()
{
  Serial.println("Server: starting");
  WebServer.on("/", webRoot);
  WebServer.on("/display", webDisplay);
  WebServer.on("/cloudSetup", cloudSetup);
  WebServer.on("/wifiNoScanSetup", wifiNoScanSetup);
  WebServer.on("/wifiSetup", wifiSetup);
  WebServer.on("/sensors", webSensors);
  WebServer.on("/reboot", webReboot);
  WebServer.on("/styles.css", webStyles);
  WebServer.on("/main.js", webJs);
  WebServer.on("/main_onload.js", webJs2);
  WebServer.onNotFound(handleNotFound);
  WebServer.begin();
  Serial.println("Server: started");
}
```

The module is responsible for creating hypertext pages on the web server, forming the user interface. These pages include various settings pages such as system network settings (wifiNoScanSetup and wifiSetup), sensor settings (sensors), data export settings (cloudSetup), display settings (display), a reboot system link, a stylesheet defining the site's design, and JavaScript scripts.

A separate module was developed for initializing the data transmitters based on the user's configuration. This module handles requests to the data provider libraries and activates them according to the user's settings. Once successfully initialized, the system starts displaying data from the data providers. The data export module receives data from the sensors, obtained through another module, and sends them to a remote web server in the appropriate format required by the chosen cloud service. The subsystem supports various data export methods, including exporting data to ThingSpeak, the "NarodMon" service, and using the MQTT protocol.

The data display module is responsible for presenting all stages of the subsystem's operation. This includes initializing the web server, creating an access point for configuration, and displaying readings from the data sensors on the connected and configured OLED display through the user interface.

The development of all these modules ensures the necessary functionality of the subsystem and creates a user-friendly interface. The complete code listing of the subsystem can be found in Appendix B, while the operational algorithm of the IT ecosystem monitoring subsystem is provided on the "Working Algorithm" poster.

3.3 Development of the subsystem interface

A user-friendly interface is essential for the seamless operation of any subsystem, as it determines whether people will utilize it regularly.

After defining the functionality of the "Subsystem for monitoring the company's IT ecosystem," interface layouts for its pages were developed. The main page contains general information that allows users to assess if the system has been properly configured and view data from activated sensors.

The main page includes the following elements:

- Web interface menu
- System identifier
- IP address and MAC address of the monitoring system
- Available memory

- Readings from temperature, relative humidity, atmospheric pressure, and CO2 sensors
- Data from the analog data transmitter

The "Settings" page is designed for performing basic system configurations necessary for its correct operation. It includes fields for specifying:

- Access point name
- Password for the access point created by the system
- Deep sleep mode switch
- SSID (wireless network name)
- Password for connecting to the wireless network
- Static IP addressing mode status
- Main gateway
- Network mask

This page is crucial for setting up the system, as it contains key parameters that users need to specify for the stable and reliable operation of the subsystem. The final step for the user is to enable or disable the data transmitters connected to the monitoring system on the "Data Transmitters" page, as shown in Figure 14.

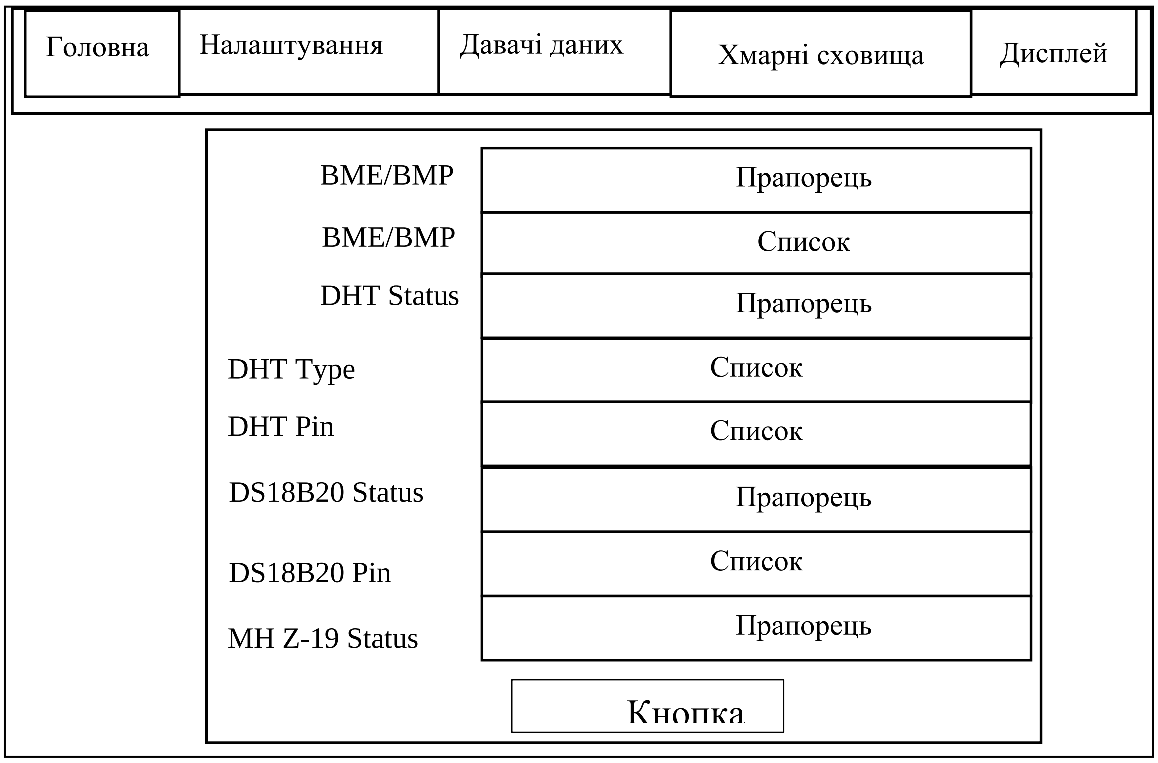


Figure 14 – Layout of the "Data Providers" page

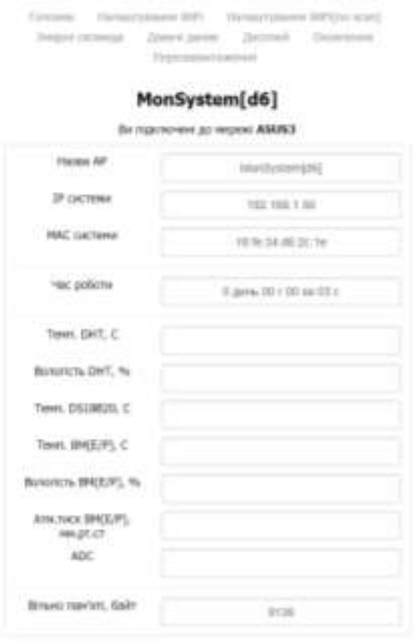
The "Data Transmitters" page provides functionality for managing the connected transmitters within the subsystem.

On the "Cloud Storage" page, users can configure the export of data from the subsystem to various cloud services or the MQTT server of the broker. This involves specifying settings for the MQTT server, password, and corresponding topic. The page includes the following fields:

- NarodMon switch
- ThingSpeak switch
- MQTT switch
- API key for ThingSpeak
- Data export interval
- MQTT server
- MQTT user
- MQTT password
- MQTT topic

Users can navigate between different functionalities of the website by selecting the appropriate items in the menu. The subsystem's interface is designed to be user-friendly and accessible, even for non-specialists. The color scheme of the website is neutral, devoid of bright colors, and ensures proper display on any device.

Figure 15 illustrates the main page of the subsystem's web interface.



MonSystem[d6]

Ви підключені до мережі **ASUS3**

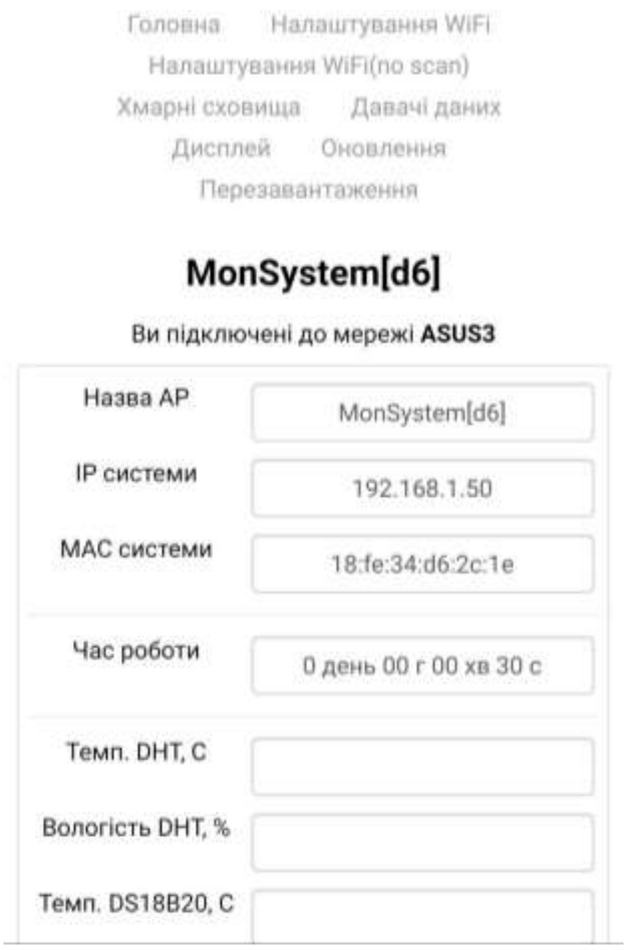
Назва AP	MonSystem[d6]
IP системи	[illegible]
MAC системи	[illegible]
Час роботи	[illegible]
Темп. DHT, C	
Вологість DHT, %	
Темп. DS18B20, C	
Темп. BM(E/P), C	
Вологість BM(E/P), %	
Атм.тиск BM(E/P), мм.рт.ст	
ADC	
Вільна пам'ять, байт	[illegible]

Fig. 15 – Main page of the subsystem interface

To ensure the web interface's versatility in configuring the subsystem and viewing sensor data from any device, including personal computers, laptops, smartphones, and tablets, an adaptive design approach was implemented. This design approach enables the website to automatically adjust the placement and sizes of components based on the device type and screen size.

Implementing an adaptive design enhances the accessibility of the web interface across various devices. It ensures that users can comfortably interact with the interface regardless of the device they are using. The adaptive design allows for optimal utilization of screen space, providing a seamless user experience.

Figure 16 illustrates the adaptability of the design, showcasing how the interface adjusts to the screen's expansion.



MonSystem[d6]

Ви підключені до мережі **ASUS3**

Назва AP	MonSystem[d6]
IP системи	192.168.1.50
MAC системи	18:fe:34:d6:2c:1e
Час роботи	0 день 00 г 00 хв 30 с
Темп. DHT, C	
Вологість DHT, %	
Темп. DS18B20, C	

Fig. 16 - Responsiveness of web interface design

The inclusion of responsive design support eliminates the necessity of creating a separate version of the web interface specifically for mobile devices. This ensures that the web interface can seamlessly adapt to different screen sizes and device types, providing an optimal user experience across various devices.

3.4 Functional testing of the subsystem

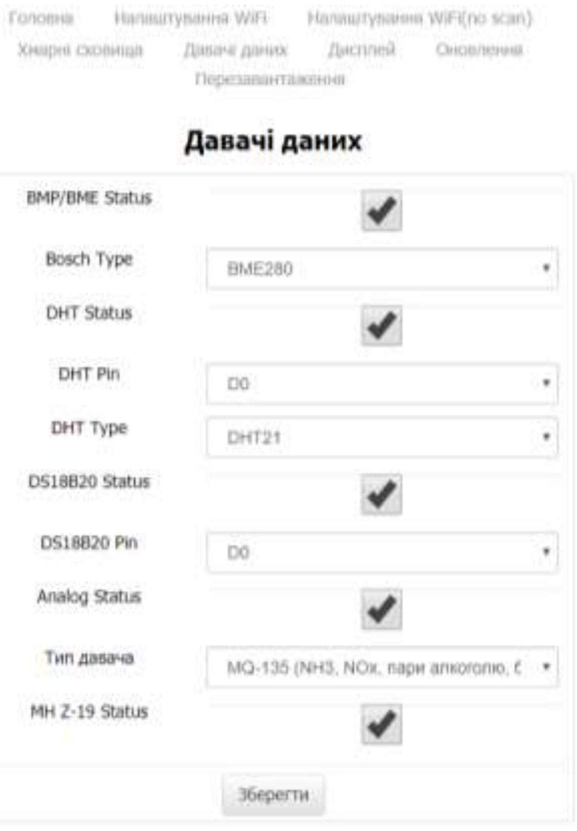
Functional testing focuses on assessing the system's external behavior, including its functions, features, and interaction with other systems. These tests can be conducted at different levels, such as component/module testing, integration testing, system testing, and acceptance testing.

Functional testing aims to verify the overall functionality of the system's interface. This includes:

- Checking the operation of mandatory site functions.

- Testing the performance of user interface forms.
- Verifying the functionality of hyperlinks.

Testing the system's main functions should commence with a verification of its ability to create an access point upon power-on. This access point serves as the user's gateway to the system. To ensure security and prevent unauthorized changes to system settings, the access point is protected by a password. The next stage of testing involves confirming that the system successfully receives data from enabled and connected data transmitters, as depicted in Figure 17. Enabling or disabling data sensors can be achieved by checking the corresponding box next to the sensor's name and selecting the sensor type or the microcontroller output to which it is connected.

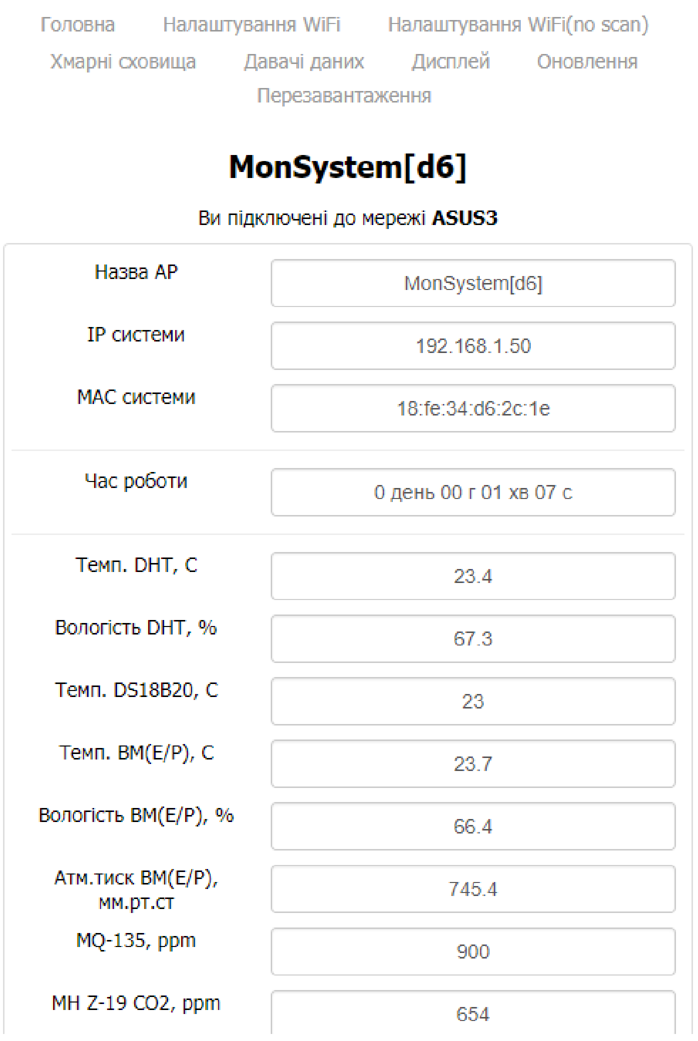


Давачі даних

BMP/BME Status	✓
Bosch Type	BME280
DHT Status	✓
DHT Pin	D0
DHT Type	DHT21
DS18B20 Status	✓
DS18B20 Pin	D0
Analog Status	✓
Тип давача	MQ-135 (NH3, NOx, пари алкоголю, €
MH Z-19 Status	✓

Fig. 17 – Management of data providers

After turning on the sensors, the subsystem must be rebooted so that it will initialize the new sensors on the next startup. Now, on the main page of the subsystem shown in Figure 18, you can observe the data received from the sensors that have been turned on.



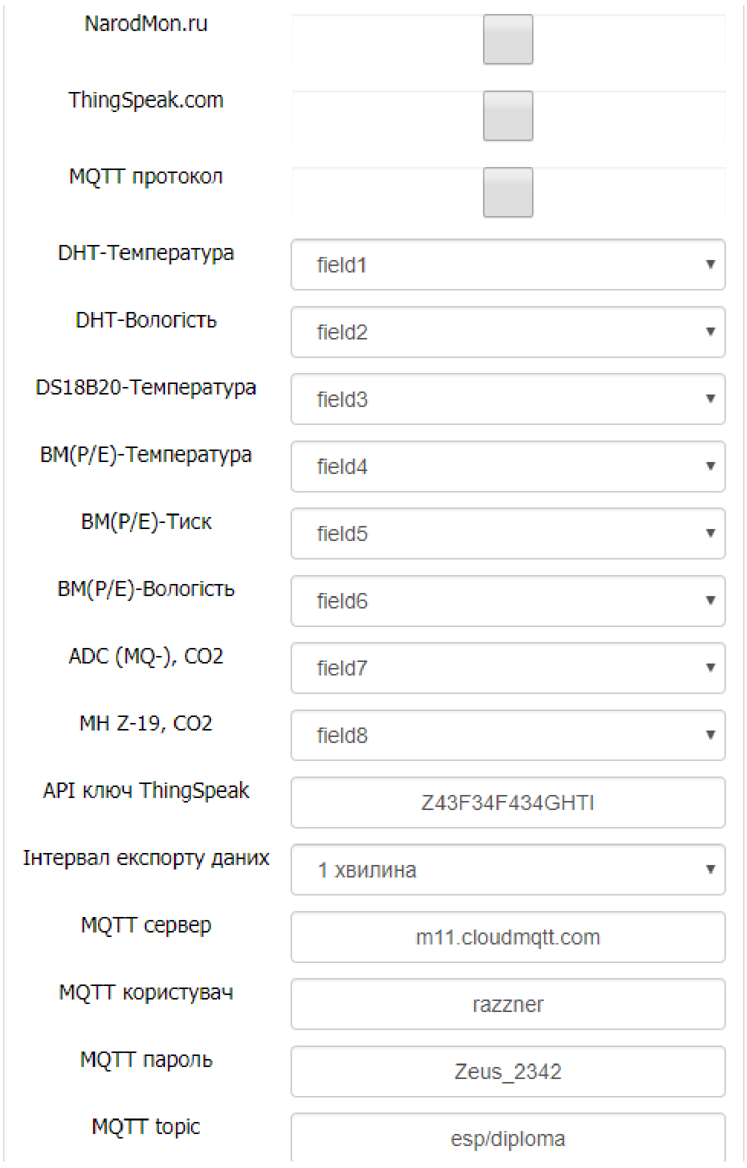
MonSystem[d6]

Ви підключені до мережі **ASUS3**

Назва AP	MonSystem[d6]
IP системи	192.168.1.50
MAC системи	18:fe:34:d6:2c:1e
Час роботи	0 день 00 г 01 хв 07 с
Темп. DHT, C	23.4
Вологість DHT, %	67.3
Темп. DS18B20, C	23
Темп. BM(E/P), C	23.7
Вологість BM(E/P), %	66.4
Атм.тиск BM(E/P), мм.рт.ст	745.4
MQ-135, ppm	900
MH Z-19 CO2, ppm	654

Fig. 18 – Received data from enabled data transmitters

The subsystem's key functionality includes the ability to export data to different cloud storage platforms or any MQTT server associated with a broker. To enable these functions, users are required to obtain an authorization key from the cloud storage provider's website and enter it in the designated settings section. Alternatively, when using the MQTT protocol, users need to provide the web server address, username, password, and topic on the server, as illustrated in Figure 19. Additionally, users are required to specify the desired data export interval.



NarodMon.ru	☐
ThingSpeak.com	☐
MQTT протокол	☐
DHT-Температура	field1 ▾
DHT-Вологість	field2 ▾
DS18B20-Температура	field3 ▾
BM(P/E)-Температура	field4 ▾
BM(P/E)-Тиск	field5 ▾
BM(P/E)-Вологість	field6 ▾
ADC (MQ-), CO2	field7 ▾
MH Z-19, CO2	field8 ▾
API ключ ThingSpeak	Z43F34F434GHTI
Інтервал експорту даних	1 хвилина ▾
MQTT сервер	m11.cloudmqtt.com
MQTT користувач	razzner
MQTT пароль	Zeus_2342
MQTT topic	esp/diploma

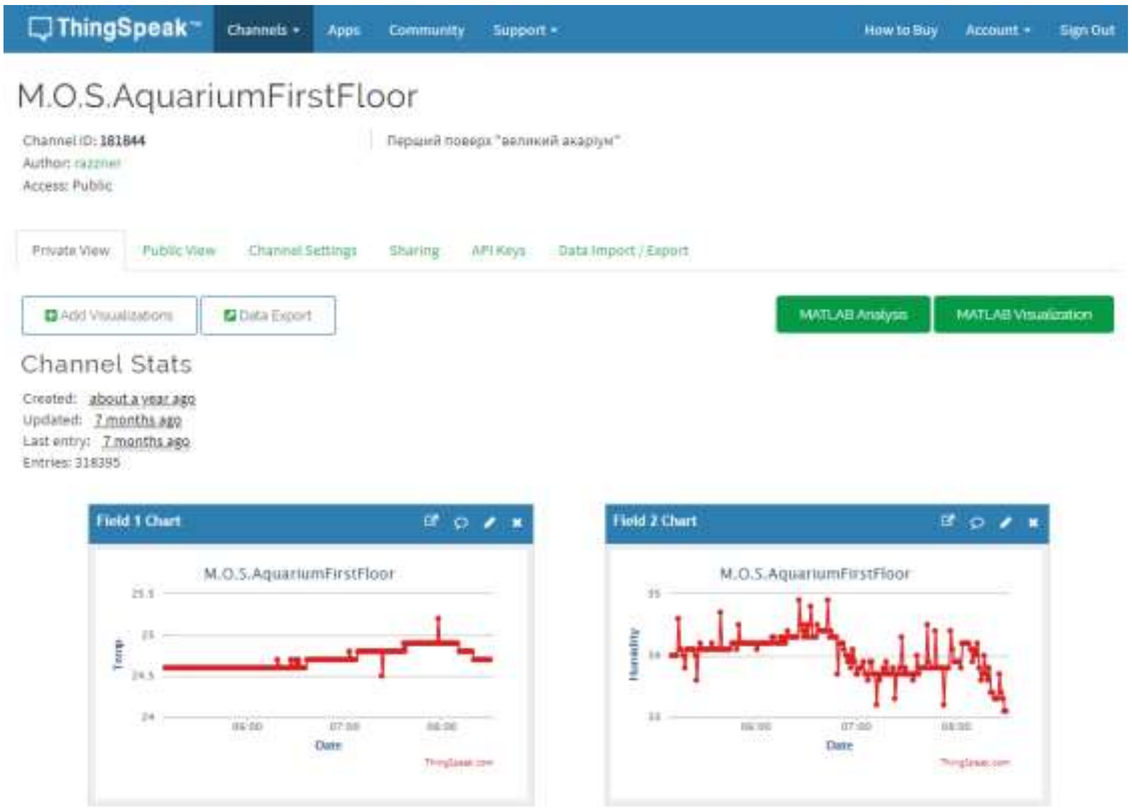
Fig. 19 - Data export settings

Upon completing the aforementioned settings, the subsystem requires specific data to establish a connection to the global network. Users can configure these settings, such as the name of the wireless network (SSID) and its password.

The subsystem also supports static IP addressing mode, which can be enabled by selecting the checkbox in the "Static IP Mode" field. When enabling this mode, users must provide the static IP address, primary gateway, and netmask. After entering these details, the subsystem needs to be rebooted. Upon the next power-on, the subsystem will establish a connection to the access point, taking into account the static IP addressing.

Once access to the subsystem settings is gained, users have the ability to control the creation of an access point if a successful connection to the global network is established. They can also modify the name and password of the access point created by the subsystem.

The final step in testing the system's operation is to view the exported data on one of the available cloud data repositories, such as ThingSpeak. Users need to log into their profile and access the data view of their channel, as illustrated in Figure 20.



M.O.S.AquariumFirstFloor

Channel ID: 181844
Author: razzner
Access: Public

Перший поверх "великий акаріум"

Channel Stats

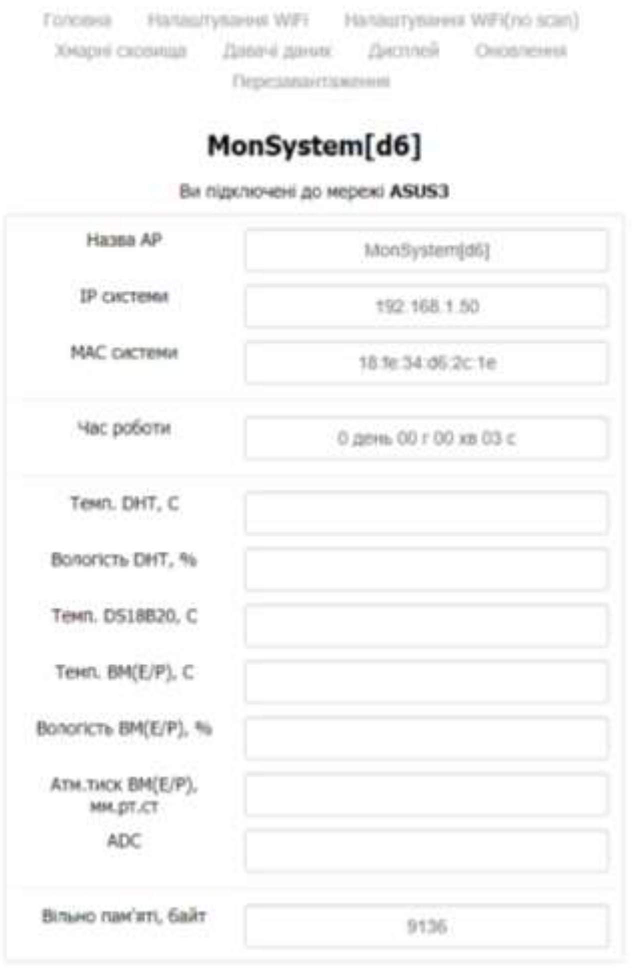
Created: about a year ago
Updated: 7 months ago
Last entry: 7 months ago
Entries: 318395

Fig. 20 - Viewing data on the ThingSpeak cloud storage site

Successful appearance of the subsystem's data in the cloud storage interface confirms a successful configuration, indicating that the subsystem is ready for operation.

3.5 Testing the interface template layout

The web interface of the monitoring system should be accurately displayed on any device, ensuring cross-browser compatibility and adaptability. Popular web browsers, including Google Chrome, Safari, Mozilla Firefox, Internet Explorer, and Opera, each follow general guidelines for rendering page markup. However, they may process the code differently based on their respective rendering engines. Testing the adaptability of the web interface was conducted using developer tools, specifically those integrated into the Google Chrome browser. Figure 21 illustrates the display of the system's web interface on a typical laptop screen.



MonSystem[d6]

Ви підключені до мережі **ASUS3**

Назва AP	MonSystem[d6]
IP системи	192.168.1.50
MAC системи	18:fe:34:d6:2c:1e
Час роботи	0 день 00 г 00 хв 03 с
Темп. DHT, C	
Вологість DHT, %	
Темп. DS18B20, C	
Темп. BM(E/P), C	
Вологість BM(E/P), %	
Атм.тиск BM(E/P), мм.рт.ст	
ADC	
Вільно пам'яті, байт	9136

Fig. 21 – The view of the web interface on the screen of a typical laptop

The template's adaptability was tested by emulating the screen of an Apple iPhone, as depicted in Figure 22. When viewed on a mobile phone screen, it is evident that the website menu has been reorganized to accommodate the narrower screen width. The menu now spans across two rows to ensure optimal visibility and usability.

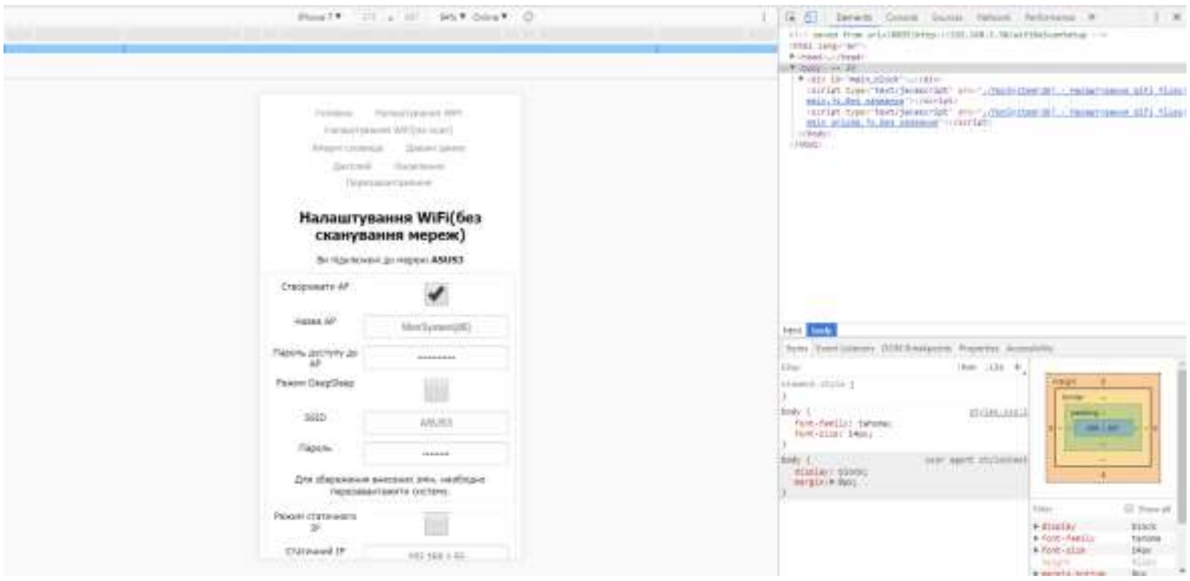


Fig. 22 – iPhone template adaptability testing

The adaptability testing can conclude at this point, as it has considered the display of the template on various popular screen resolutions, and it has been observed to be correctly displayed across all scenarios.

3.6 Security Testing

Security testing is a crucial strategy employed to ensure the system's security and analyze potential risks associated with protecting applications from hacker attacks, viruses, and unauthorized access to sensitive data. The security strategy is based on three fundamental principles: confidentiality, integrity, and availability.

In the subsystem, access security to the settings is ensured through password authentication when connecting to the access point, which is created upon system startup and allows access to the web interface. The access point utilizes the WPA2 encryption algorithm, which is a secure protocol designed to replace the outdated WEP protocol. It addresses the vulnerabilities of WEP, such as encryption key reuse, by implementing TKIP (Temporary Key Integrity Protocol).

The subsystem's interface provides the option to change the access point's password and name. This can be done through the corresponding fields in the system settings, as depicted in Figure 23.

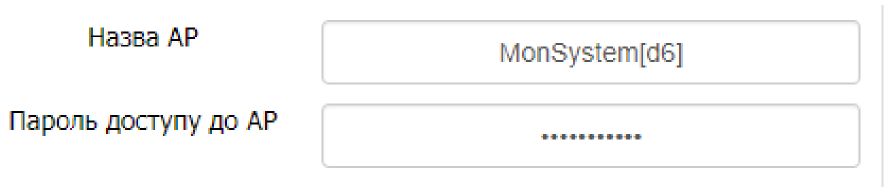

Назва AP
MonSystem[d6]
Пароль доступу до AP
•••••••••••

Fig. 23 – Changing the password and name of the access point

To access the subsystem, users are required to input a valid password. Upon successful authentication, they will gain access to the web interface of the subsystem. Based on the test results, it can be concluded that the subsystem effectively fulfills its tasks and offers a user-friendly web interface for configuration.

3.7 Utilizing IT Ecosystem Monitoring for the Company

In recent years, there has been a significant rise in the popularity of various systems capable of connecting to the Internet of Things (IoT). The number of such systems continues to grow annually. Experts estimate that by 2021, the global IoT market value will reach $7.1 trillion, with approximately 25 billion connected devices. This estimate does not even include devices like tablets, smartphones, and laptops.

The designed monitoring system aims to collect information flows within the company's IT infrastructure. The placement of sensors for monitoring is illustrated in Figure 24. By installing the required data providers in appropriate locations and configuring them for use within the subsystem, the IT office's ecosystem can be effectively monitored.

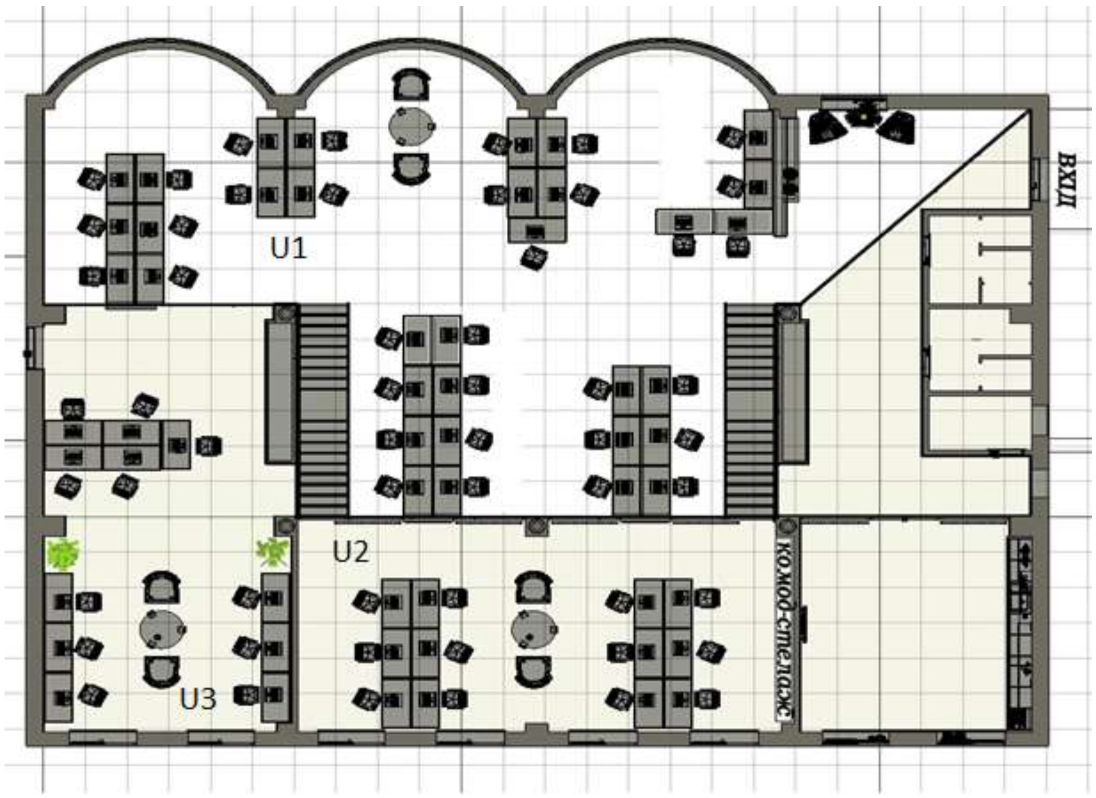


Fig. 24 – Scheme of the first tier of the 4th floor of the Smart office of IT company

The first tier of the subsystem consists of three modules, each equipped with different data providers. Module U1 is positioned in the open-space area, gathering data on temperature and humidity. Module U2 is situated in a separate room exposed to direct sunlight and equipped with an air conditioner. It collects data on carbon dioxide levels, temperature, and humidity, enabling assessment and analysis of the working conditions for IT employees in terms of air quality. Module U3 is strategically placed in a corner section, aiming to detect any cold spots within the office.

The second tier, illustrated in Figure 25, also comprises multiple modules equipped with different data providers. Module U3 is located in the office of the IT manager, monitoring temperature, humidity, atmospheric pressure, and carbon dioxide levels. Additionally, module U4 is situated in a corner section, allowing for the identification of potential heat losses within the room.

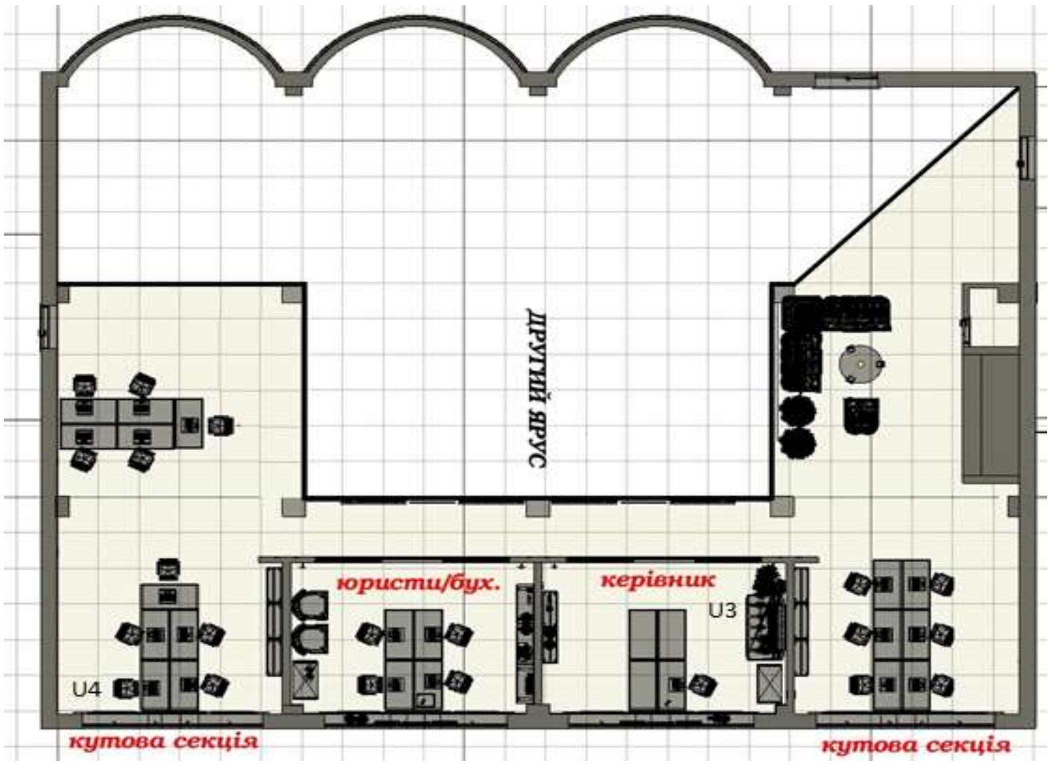


Figure 25 – Scheme of the second tier of the 4th floor of the Smart office of IT company

The installation of an ecosystem monitoring subsystem within the company's IT infrastructure enables remote monitoring of various parameters. By accessing the collected data, users can analyze the state of the IT office's ecosystem and take necessary actions to normalize it whenever required.

4. OCCUPATIONAL SAFETY AND HEALTH

Occupational safety and health issues are considered for the design and development phase of climate data analysis and visualization system.

Occupational safety is a system of legal, socio-economic, organizational and technical, sanitary and hygienic and treatment and prevention measures and tools aimed at preserving human life, health and ability to work. Working conditions at the workplace, safety of technological processes, machines, mechanisms, equipment and other means of production, condition of collective and individual protection means used by the employee, as well as sanitary and living conditions must meet the requirements of the law. An employee has the right to refuse the assigned work if a work situation has arisen that is dangerous to his life or health or to the people around him, or to the work environment or the environment. He must immediately notify his immediate supervisor or employer. The existence of such a situation is confirmed, if necessary, by labor protection specialists of the enterprise with the participation of a representative of the trade union of which he is a member or a person authorized by employees on labor protection (if the trade union was not established), as well as an insurance expert [12]. The task of labor protection is to minimize injuries and illnesses of the employee while ensuring comfort with maximum productivity. The main objectives of labor protection are the formation of specialists with the necessary knowledge and practical skills on legal and organizational issues of labor protection, industrial sanitation, safety, fire safety.

4.1. General characteristics of the room and workplace

The development of the analysis and visualization system is performed in a room located on the fourth floor of an eight-storey building with general and local lighting. The room has one-sided lighting, the windows are oriented to the east, the windows have shutters. White ceiling with a reflection coefficient of 0.7, light brick walls with a reflection coefficient of 0.5. There are 4 people working in the room, in accordance with this we obtain input data for the analysis of potentially dangerous and harmful production factors, which are given in table. 4.1.

Table 4.1

Incoming data

Room parameters	**Value**
Length x width x height	6.6 x 6.1 x 2.7 m
Area	40.26m²
Volume	108,70 m³
Workplace number	**Specifics of work**
I workplace	Front-end programmer (web application client development specialist)
II workplace	Back-end programmer (specialist in the development of the server part of web applications and database design)
III workplace	Business analyst (also acts as a product manager)
IV workplace	UI-UX web designer
Technical means (quantity)	**Name and characteristics**
Monitor (4 pcs.)	HP 22Xi / 21.5 "/ 1920x1080px / IPS
Computer (4 pcs.)	HP ProBook 440 G6, 14 "IPS screen (1920x1080) Full HD, Intel Core i7-8565U (1.8 - 4.6 GHz) / RAM 16 GB / SSD 256 GB
Floor cooler (1 piece)	CRYSTAL YLR3-5V208
Air conditioner (1 piece)	DEKKER DSH105R / G / 26m2 / 2,65kW- 2.9 kW / 25x74.5x19.5 cm / 9 kg
General purpose luminaries (3 pcs.)	The lamp raster built-in 4x18W
Local lamps (4 pcs.)	Delux Decor TF-05/1 x 40W

According to NPAOP 0.00-7.15-18, the area S 'allocated for one workplace with a personal computer must be at least 6 m2 and the volume - at least 20 m3. There are 4 workplaces in the room, which fully meets the required standards.

We calculate the actual values of these indicators by dividing the volume of the room and the total area by the number of employees.

Therefore, based on the results obtained in terms of area and volume, the room meets the standards.

Table 4.2

Workplace characteristics

№	The name of the parameter	Value	
		in fact	Normative
1.	Height of a working surface, mm	780	680 – 800
2.	Width of a working surface, mm	1500	not less than 600
3.	Depth of a working surface, mm	750	not less than 600
4.	Height of space for legs, mm	750	not less than 600
5.	Width of space for legs, mm	800	not less than 500
6.	Depth of space for legs, mm	750	not less than 450
7.	Seat surface height, mm	480	400 – 500
8.	Seat width, mm	500	not less than 400
9.	Seat depth, mm	500	not less than 400
10.	Height of a basic surface of a back, mm	550	not less than 300
11.	Width of a surface of a back, mm	470	Not less than 380
12.	Length of armrests, mm	300	not less than 250
13.	Width of armrests, mm	60	50 – 70
14.	Distance from eyes to the screen, mm	650	600 – 700

It is possible to draw a conclusion that the sizes of a workplace of the programmer correspond to the established norms, proceeding from the set parameters.

4.2 Analysis of potentially dangerous and harmful production factors in the workplace

When creating a system of analysis and visualization, the work is performed sitting without physical effort, so it belongs to the category of light Ia.

Premises for work must be equipped with heating, air conditioning or supply and exhaust ventilation in accordance with DBN B.2.5-67: 2013. Normalized parameters of the microclimate, ionic composition of air, content of harmful substances meet the requirements of LTO 3.3.6.042-99, GN 2152-80, GOST 12.1.005-88, DSTU GOST 12.0.230: 2008 and DSTU GOST 12.4.041: 2006. Ventilation is understood as a set of measures and means designed to ensure meteorological conditions and cleanliness of the air environment that meet hygienic and technical requirements at permanent places and service areas. The main task of ventilation is to remove polluted, humid or heated air from the room and supply clean fresh air.

The sources of noise in the room are the fan of the system unit, laptop and air conditioner. The sound generated by the fan and air conditioner can be classified as constant.

According to DBN B.2.5-28: 2018 the work belongs to the category of visual works. The use of natural, artificial and mixed lighting is envisaged.

The computer is a single-phase consumer of electricity powered by 220V AC from a network with grounded neutral. IBM PC refers to electrical installations up to 1000V closed version; all conductive parts are in the casings. According to the method of protecting a person from electric shock, computers and peripherals must meet 1 class of protection.

Technical methods of protection against electric shock is reduced to the use of current of safe voltage, protection in case of accidental touching current-carrying parts and against excessive currents, protection in case of voltage transfer to non-current-carrying metal parts of the installation.

Safe voltage is obtained from the high voltage grid (110-120 V) by means of step-down transformers.

Protection against contact with live parts of the installation is achieved by means of insulation, fencing off the use of blocking safety devices and inaccessibility of the location of the installations.

Switchboards are placed in closed metal casings-boxes.

Safety alarm is used in the form of posters and inscriptions. The best light alarms are double, which in the presence of voltage lights a red light, and in its absence - green.

Protection against excessive currents - short circuits and overload currents, which can cause insulation to ignite, is provided by fuses and circuit breakers, and protection against voltage transfer to live parts by means of protective earthing and protective disconnection.

Fire prevention is achieved by eliminating the formation of sources of ignition and combustible environment.

Fires of the following classes are possible in this room: A - combustion of solids, E - combustion of live electrical installations.

CONCLUSIONS

The main goal of this work was the development of a Smart office using IoT.

An analysis of the subject area has been conducted, and the objectives of the coursework have been established. The requirements for the subsystem responsible for collecting and aggregating information within the ecosystem have been outlined. The necessary hardware components have been selected, and the subsystem's structure has been designed, including the development of a schematic diagram to ensure its reliable performance and successful task execution. The subsystem's behavior in response to user actions has been defined.

A programming language and appropriate development tools have been chosen to meet the project's requirements and provide the necessary functionality. The structural elements of the subsystem and its web interface in configuration mode have been developed. Functional testing has been performed to ensure proper operation across various devices, and the layout of the interface template has been evaluated. Security testing has also been conducted to ensure the subsystem's protection.

Furthermore, potential applications of the subsystem for collecting and accumulating information within the company's IT premises have been described. The created subsystem fully satisfies the requirements and successfully accomplishes all assigned tasks. The obtained results have been analyzed, and the developed subsystem has been tested for monitoring the company's IT ecosystem using IoT technologies. The implemented subsystem enables the collection of diverse data types and their storage in various cloud storage platforms.

REFERENCES

1. Asghar M., Mohammadzadeh N., «Design and simulation of energy efficiency in node based on MQTT protocol in Internet of Things» // International Conference on Green Computing and Internet of Things. – 2015. – C. 1413-1417.

2. Bass A., Bauer M., Fiedler M., Kramp T., van Kranenburg R., Lange S., Meissner S. Enabling Things to Talk. Springer-Verlag GmbH, 2013. – P. 325.

3. Gubbi J., Marusicet S., Buyya R., Palaniswami M. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems. – 2013. – №. 7. – C. 1645–1660. [сайт]. - URL: 10.1016/j.future.2013.01.010.

4. Heather Flanagan, "Digital Preservation Considerations for the RFC Series," January 2015, Internet Draft, work in progress, draft-flanagan-rfc-preservation-03.

5. Internet of Things Global Standards Initiative: [WEB]. – URL: http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx

6. Internet of Things Global Standards Initiative: [WEB]. – URL: http://www.itu.int/en/ITU-T/gsi/iot/ Pages/default.aspx.

7. Kang D., Park M. Lin S.W., Martin R.A., Miller B.W., Durand J. et al. Industrial internet reference architecture technical report. IIC, 2015.

8. Lobaccaro G, Carlucci S, Lofstrom E. A review of systems and technologies for smart homes and smart grids. Energies, 2016. – 348 c.

9. Shih C., Chou J., Designing CPS/IoT applications for smart buildings and cities / C. Shih, J. Chou // IET Cyber-Physical Systems: Theory & Applications. – 2016. – № 1. – C. 3-12.

10. Wortmann F., Flüchter K. Internet of things. Business & Inform. Syst. Eng, 2015. – № 3. – C. 221–224 .

11. Xia F., Yang L.T., Wang L., Vinel A. Internet of things. Int. J. of Commun. Syst. – 2012. – Vol. 25. – № 9. – C. 1101–1109.

12. Yih-Fang Huang; Werner, S.; Jing Huang; Kashyap, N.; Gupta, V., "State Estimation in Electric Power Grids: Meeting New Challenges Presented by the

Requirements of the Future Grid," Signal Processing Magazine, IEEE , vol.29, no.5, pp.33,43, Sept. 2012.

13. Tomoiagă, B.; Chindriş, M.; Sumper, A.; Sudria-Andreu, A.; Villafafila-Robles, R. Pareto

Optimal Reconfiguration of Power Distribution Systems Using a Genetic Algorithm Based on NSGA-II. Energies 2013, 6, 1439-1455.

14. F.R. Yu, P. Zhang, W. Xiao, and P. Choudhury, "Communication Systems for Grid

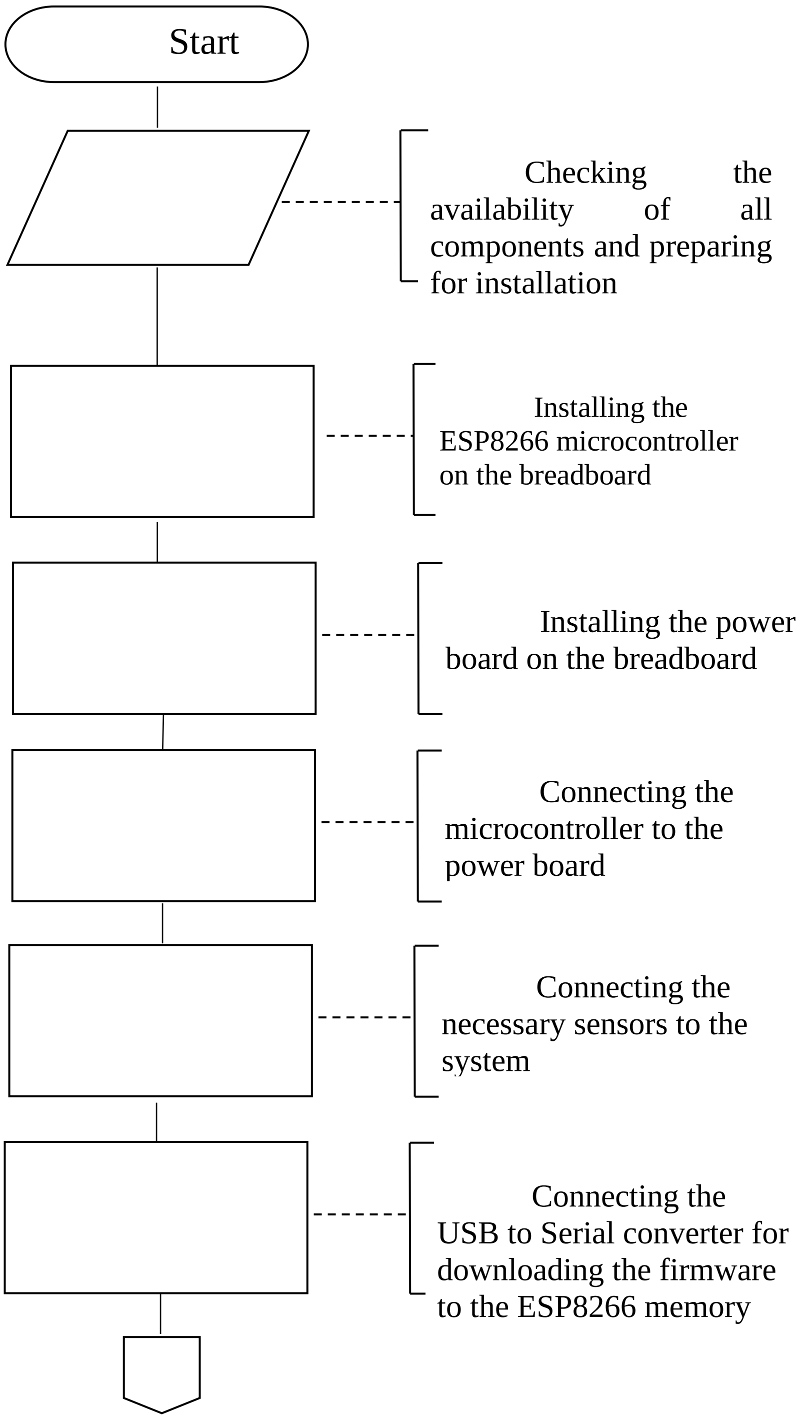
Integration of Renewable Energy Resources," IEEE Network, vol. 25, no. 5, pp. 22-29, Sept.

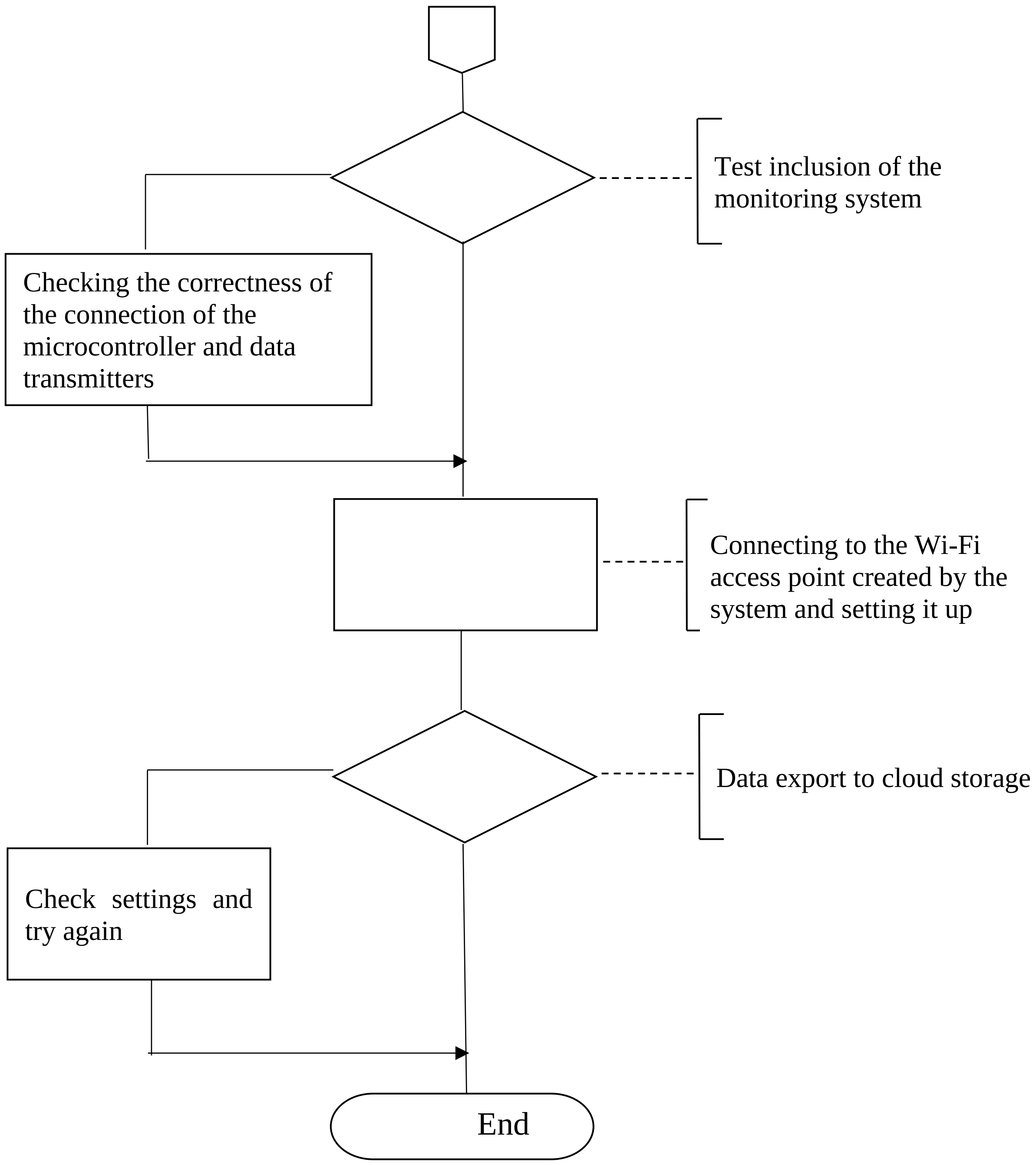
2011.

15. “Values and Principles”. Principles. Internet Society, 2015. http://www.internetsociety.org/who-we-are/mission/values-and-principles.

APPENDIX A

Algorithm for compiling the monitoring system


Start
Checking the availability of all components and preparing for installation
Installing the ESP8266 microcontroller on the breadboard
Installing the power board on the breadboard
Connecting the microcontroller to the power board
Connecting the necessary sensors to the system
Connecting the USB to Serial converter for downloading the firmware to the ESP8266 memory


Test inclusion of the monitoring system
Checking the correctness of the connection of the microcontroller and data transmitters
Connecting to the Wi-Fi access point created by the system and setting it up
Data export to cloud storage
Check settings and try again
End

Subsystem code fragment

```
#include <ESP8266WiFi.h>
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <WiFiClient.h>
#include <ESP8266mDNS.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266HTTPUpdateServer.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include <DHT.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <DallasTemperature.h>
#include <LiquidCrystal_I2C.h>
#include "SSD1306.h"
#include "OLEDDisplayUi.h"
#include <SoftwareSerial.h>
//#include <PubSubClient.h>
#include "MQ135.h"
#include "JsonConfig.h"
#include "WebCommon.h"
#include "Common.h"
extern "C" {
#include <user_interface.h>
}
ESP8266WebServer WebServer(80);
ESP8266HTTPUpdateServer httpUpdater;
const int maxConnectAttempts = 20;
JsonConfig config;
#define MAX_WIFI_COUNT 50
WiFiData wiFiDatas[MAX_WIFI_COUNT];

SensorData data1;
// Data wire is plugged into port 2 on the Arduino
#define TEMPERATURE_PRECISION 12 // resolution
SensorData data2;
// arrays to hold device address
DeviceAddress insideThermometer;
#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BMP085 bmp;
Adafruit_BME280 bme; // I2C

SensorData data3;
SensorData data4;
#define MH_Z19_RX D6
#define MH_Z19_TX D5
SoftwareSerial MH_Z19Serial(D6, D5); // define MH-Z19
```

```
     SensorData data5;

     SSD1306  display(0x3c, D2, D1);
     OLEDDisplayUi ui     ( &display );

     #define AIO_SERVER      "io.adafruit.com"
     #define AIO_SERVERPORT  1883                   // 8883 for
MQTTS
     #define AIO_USERNAME    "razzner"
     #define AIO_KEY         "408905b770614823984b0ac73344b7ca"

     // Create an ESP8266 WiFiClient class to connect to the MQTT
server.
     WiFiClient mqtt_client;
     // Setup the MQTT client class by passing in the WiFi client
and MQTT server and login details.
     Adafruit_MQTT_Client mqtt(&mqtt_client, AIO_SERVER,
AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

     Adafruit_MQTT_Publish reboot_log =
Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/home-err-log");
     Adafruit_MQTT_Publish sensor_err_log =
Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/sensor-error-
log");

     const char* host = "api.thingspeak.com";
     String mac = getMacString();

     void wifiSetup()
     {
       Serial.println("\r\nServer: request SETUP");

       bool config_changed = false;
       String payload = WebServer.arg("ap_name");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.ap_name,
sizeof(config.ap_name));
         config_changed = true;
       }
       payload = WebServer.arg("ap_toogle");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.ap_toogle,
sizeof(config.ap_toogle));
         config_changed = true;
       }
       payload = WebServer.arg("ap_pwd");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.ap_pwd, sizeof(config.ap_pwd));
         config_changed = true;
       }
       payload = WebServer.arg("sta_ssid");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.sta_ssid,
sizeof(config.sta_ssid));
         config_changed = true;
       }
       payload = WebServer.arg("sta_pwd");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.sta_pwd,
sizeof(config.sta_pwd));
         config_changed = true;
       }
       //    payload = WebServer.arg("hidden_toogle");
       //    if (payload.length() > 0)
       //    {
       //        payload.toCharArray(config.hidden_toogle,
sizeof(config.hidden_toogle));
       //        config_changed = true;
       //    }
       payload = WebServer.arg("deepsleep_toogle");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.deepsleep_toogle,
sizeof(config.deepsleep_toogle));
         config_changed = true;
       }
       payload = WebServer.arg("static_ip_toogle");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.static_ip_toogle,
sizeof(config.static_ip_toogle));
         config_changed = true;
       }
       payload = WebServer.arg("static_ip");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.static_ip,
sizeof(config.static_ip));
         config_changed = true;
       }
       payload = WebServer.arg("static_gateway");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.static_gateway,
sizeof(config.static_gateway));
         config_changed = true;
       }
       payload = WebServer.arg("static_subnet");
       if (payload.length() > 0)
       {
         payload.toCharArray(config.static_subnet,
sizeof(config.static_subnet));
         config_changed = true;
       }
       String ssid_list, ssid_h_name;
       int numSsid = WiFi.scanNetworks();
       // print the name for each network found:
       for (int thisNet = 0; thisNet < numSsid; thisNet++) {
         String ssid_str = WiFi.SSID(thisNet) + " (" +
WiFi.RSSI(thisNet) + " dBm)",
                ssid_name =  WiFi.SSID(thisNet),
                rssi = String(WiFi.RSSI(thisNet));
         ssid_list += renderParameterList(ssid_name, ssid_str,
rssi);
       }
       if (WiFi.status() == WL_CONNECTED)
       {
         ssid_h_name = "<p>Ви підключені до мережі <b
id='ssid_name'>" + String(config.sta_ssid) + "</b></p>";
       }
       else {
         ssid_h_name = "";
       }
       String data =
         renderTitle(config.ap_name, "Налаштування WiFi") +
FPSTR(stylesInclude) + FPSTR(headEnd) + FPSTR(bodyStart) +
renderMenu() +
         "<h2>Налаштування WiFi</h2>" +
         ssid_h_name +
         "<div class='container'>" +
         renderParameterRow("Створювати AP", "ap_toogle",
config.ap_toogle) +
         renderParameterRow("Назва AP", "ap_name", config.ap_name)
+
         renderParameterRow("Пароль доступу до AP", "ap_pwd",
config.ap_pwd, false, true) +
         //        renderParameterRow("Прихований SSID",
"hidden_toogle", config.hidden_toogle) +
         renderParameterRow("Режим DeepSleep", "deepsleep_toogle",
config.deepsleep_toogle) +
         "<hr/>" +
         "<div class='input-group'><label class='input_label'
for='sta_ssid'>SSID</label><select id='sta_ssid' class='form-
control'>" +
         ssid_list +
         "<select></div>" +
         renderParameterRow("Пароль", "sta_pwd", config.sta_pwd,
false, true) +
         "<p>Для збереження внесених змін, необхідно
перезавантажити систему.</p><hr/>" +
         renderParameterRow("Режим статичного IP ",
"static_ip_toogle", config.static_ip_toogle) +
         renderParameterRow("Статичний IP", "static_ip",
config.static_ip) +
         renderParameterRow("Основний шлюз", "static_gateway",
config.static_gateway) +
         renderParameterRow("Маска мережі", "static_subnet",
config.static_subnet) +
         "<hr/>" +
         "<a class='btn btn-default marginTop0' role='button'
onclick='saveFormData(\"/wifiSetup\");'>Зберегти</a>" +
         "</div>" +
         "</div>" +
         FPSTR(scripts) +
         FPSTR(bodyEnd);

       WebServer.send(200, "text/html", data);

       if (config_changed)
       {
         config.saveConfig();
       }

       Serial.println("Server: request SETUP sent");
     }
     void webStyles()
     {
       Serial.println("\r\nServer: request STYLES");

       String stylesText = String("") + FPSTR(styles);
       WebServer.send(200, "text/css", stylesText);

       Serial.println("Server: request STYLES sent");
     }
     void webJs()
     {
       Serial.println("\r\nServer: request Scripts");

       String Scripts = String("") + FPSTR(scripts_content);
       WebServer.send(200, "text/css", Scripts);

       Serial.println("Server: request Scripts sent");
     }

     void initWebServer()
     {
       Serial.println("Server: starting");
       WebServer.on("/", webRoot);
       WebServer.on("/display", webDisplay);
       WebServer.on("/cloudSetup", cloudSetup);
       WebServer.on("/wifiNoScanSetup", wifiNoScanSetup);
       WebServer.on("/wifiSetup", wifiSetup);
       WebServer.on("/sensors", webSensors);
       WebServer.on("/reboot", webReboot);
       WebServer.on("/styles.css", webStyles);
       //  WebServer.on("/some.js",webJs3);
       WebServer.on("/main.js", webJs);
       WebServer.on("/main_onload.js", webJs2);
       WebServer.onNotFound(handleNotFound);
       WebServer.begin();
       Serial.println("Server: started");
     }

     int connectWiFi()
     {
       Serial.println("Wifi: connecting");
       int connectAttempts = 0;

       while (connectAttempts < maxConnectAttempts)
       {
         Serial.printf("Wifi: connecting, attempt %d\r\n",
connectAttempts);
         if (WiFi.status() == WL_CONNECTED)
         {
           if (atoi(config.display_toogle) == 1) {
             display.clear();
             display.setTextAlignment(TEXT_ALIGN_CENTER);
             display.setFont(ArialMT_Plain_10);
             display.drawString(64, 10, "WiFi: CONNECTED");
             display.drawString(64, 20, "SSID:" +
String(config.sta_ssid));
             display.drawString(64, 30, "with IP:" +
getIpString(WiFi.localIP()));
             display.display();
           }
           Serial.println("Wifi: connected");
           renderWiFiStatus("On", 255, 255, 255);
           return 1;
         }

         delay(500);
         connectAttempts++;
         if (atoi(config.display_toogle) == 1) {
           display.clear();
           display.setTextAlignment(TEXT_ALIGN_CENTER);
           display.setFont(ArialMT_Plain_10);
           display.drawString(64, 10, "WiFi: Connecting...");
           display.drawString(64, 20, "SSID: " +
String(config.sta_ssid));
           display.drawString(64, 30, "Attempt " +
String(connectAttempts));
           display.display();
         }
         yield();
       }

       Serial.println("Wifi: timeout");
       renderWiFiStatus("Timeout", 255, 0, 0);
       return 0;
     }

     void handleWiFiEvent(WiFiEvent_t event)
     {
       switch (event)
       {
         case WIFI_EVENT_STAMODE_CONNECTED:
           Serial.println("Wifi event:
WIFI_EVENT_STAMODE_CONNECTED");
           renderWiFiStatus("Linking", 255, 255, 0);
```

```
          break;
        case WIFI_EVENT_STAMODE_DISCONNECTED:
          Serial.println("Wifi event:
WIFI_EVENT_STAMODE_DISCONNECTED");
          renderWiFiStatus("Off", 255, 0, 0);
    //      sta_disc_flag = 1;
          break;
        case WIFI_EVENT_STAMODE_AUTHMODE_CHANGE:
          Serial.println("Wifi event:
WIFI_EVENT_STAMODE_AUTHMODE_CHANGE");
          break;
        case WIFI_EVENT_STAMODE_GOT_IP:
          Serial.println("Wifi event: WIFI_EVENT_STAMODE_GOT_IP");
          Serial.print("Wifi: connected, IP = ");
          Serial.print(WiFi.localIP());
          Serial.print(", MAC = ");
          Serial.print(mac);
          Serial.println();
          renderWiFiStatus("On", 255, 255, 255);
    //      sta_disc_flag = 0;
          break;
        case WIFI_EVENT_STAMODE_DHCP_TIMEOUT:
          Serial.println("Wifi event:
WIFI_EVENT_STAMODE_DHCP_TIMEOUT");
          break;
        case WIFI_EVENT_SOFTAPMODE_STACONNECTED:
          Serial.println("Wifi event:
WIFI_EVENT_SOFTAPMODE_STACONNECTED");
          renderAPStatus("Connected", 255, 255, 255);
          break;
        case WIFI_EVENT_SOFTAPMODE_STADISCONNECTED:
          Serial.println("Wifi event:
WIFI_EVENT_SOFTAPMODE_STADISCONNECTED");
          renderAPStatus("Off", 255, 255, 255);
          break;
        case WIFI_EVENT_SOFTAPMODE_PROBEREQRECVED:
          break;
        case WIFI_EVENT_MAX:
          Serial.println("Wifi event: WIFI_EVENT_MAX");
          break;
      }
    }

    void initWiFi()
    {
      Serial.println("Wifi: starting");

      //  renderWiFiStatus("Off", 255, 0, 0);
      //  renderServerStatus("-", 255, 255, 255);
      //  renderAPStatus("Off", 255, 255, 255);

      delay(1000);
      WiFi.mode(WIFI_STA);
      WiFi.onEvent(handleWiFiEvent);
      WiFi.begin(config.sta_ssid, config.sta_pwd);
```

```
       if (atoi(config.static_ip_toogle) == 1)
       {
         Serial.println("Wifi: use static IP");
         IPAddress staticIP = stringToIp(config.static_ip);
         IPAddress staticGateway =
stringToIp(config.static_gateway);
         IPAddress staticSubnet = stringToIp(config.static_subnet);
         WiFi.config(staticIP, staticGateway, staticSubnet);
       }
       else
       {
         Serial.println("Wifi: using DHCP");
       }

       Serial.println(String("Wifi: connect to '") +
config.sta_ssid + "' with password '" + config.sta_pwd + "'");

       connectWiFi();

       if (WiFi.status() == WL_CONNECTED)
       {
         if (atoi(config.ap_toogle) == 1) {
           Serial.println(String("Wifi: connected, creating AP ") +
config.ap_name);
           Serial.println(String("with password:  ") +
config.ap_pwd);
           WiFi.mode(WIFI_AP_STA);
           //          if(atoi(config.hidden_toogle) == 1) {
           //          Serial.println("AP options: hidden ");
           //          WiFi.softAP(config.ap_name, config.ap_pwd,
5, 1);
           //          }
           //          else {
           WiFi.softAP(config.ap_name, config.ap_pwd, 5);
           //          }
           Serial.print("Wifi: connected, IP = ");
           Serial.print(WiFi.localIP());
           Serial.println();
         }
         startup_oled_time = millis();
       }
       else
       {
         if (atoi(config.display_toogle) == 1) {
           display.clear();
           display.setTextAlignment(TEXT_ALIGN_CENTER);
           display.setFont(ArialMT_Plain_10);
           display.drawString(64, 10, "WiFi: not connected");
           display.drawString(64, 20, "created access point");
           display.drawString(64, 30, "SSID: " +
String(config.ap_name));
           display.drawString(64, 40, "Pass: " +
String(config.ap_pwd));
           display.display();
           startup_oled_time = millis();
         }
         Serial.println(String("Wifi: not connected, creating AP ")
+ config.ap_name);
         Serial.println(String("with password:  ") +
config.ap_pwd);
         WiFi.mode(WIFI_AP);
         WiFi.softAP(config.ap_name, config.ap_pwd, 5);
       }

       Serial.println("Wifi: started\r\n");

       initWebServer();

       if (!MDNS.begin(config.ap_name))
       {
         while (1)
         {
           delay(1000);
           yield();
         }
       }
       MDNS.addService("http", "tcp", 80);
     }

     void initSensors()
     {
       if (atoi(config.sensor_dht_on) == 1)
       {
         uint8_t pin = (uint8_t)atoi(config.dht_pin);
         uint8_t type = (uint8_t)atoi(config.dht_type);
         DHT dht(pin, type);
         dht.begin();
       }
       if (atoi(config.sensor_ds18b20_on) == 1)
       {
         uint8_t pin = (uint8_t)atoi(config.ds_pin);
         OneWire oneWire(pin);
         // Pass our oneWire reference to Dallas Temperature.
         DallasTemperature sensors(&oneWire);
         sensors.begin();
         sensors.getAddress(insideThermometer, 0);
         sensors.setResolution(insideThermometer,
TEMPERATURE_PRECISION);
       }

       if (atoi(config.sensor_bosch_on) == 1)
       {
     //    uint8_t bosch_adr = strtol(config.sensor_bosch_adr,
NULL, 16);
         String bosch_type(config.sensor_bosch_type);
         if (bosch_type == "bme280") {
           if (bme.begin()) {
             bme280initialized = true;
           }
         }
```

```
          else {
            if (bmp.begin()) {
              bmp180initialized = true;
            }
          }
        }
      }

      // Function to connect and reconnect as necessary to the MQTT
server.
      // Should be called in the loop function and it will take care
if connecting.
      void MQTT_connect() {
        int8_t ret;

        // Stop if already connected.
        if (mqtt.connected()) {
          return;
        }

        Serial.print("Connecting to MQTT... ");

        uint8_t retries = 3;
        while ((ret = mqtt.connect()) != 0) { // connect will return
0 for connected
          Serial.println(mqtt.connectErrorString(ret));
          Serial.println("Retrying MQTT connection in 2
seconds...");
          mqtt.disconnect();
          delay(2000);  // wait 1 seconds
          retries--;
          if (retries == 0) {
            // basically die and wait for WDT to reset me
            return;
          }
        }
        Serial.println("MQTT Connected!");
      }

      SensorData getdhtData()
      {
        uint8_t pin = (uint8_t)atoi(config.dht_pin);
        uint8_t type = (uint8_t)atoi(config.dht_type);
        DHT dht(pin, type);

        float h = dht.readHumidity();
        float t = dht.readTemperature();

        // Check if any reads failed
        if (isnan(h) || isnan(t)) {
          Serial.println("Failed to read from DHT sensor!");
          if (WiFi.status() == WL_CONNECTED) {
            MQTT_connect();
            String err = "\nMAC:" + mac + "\nDHT Err: Failed to read
from sensor, get NaN";
```

```
          const char *mycharp = err.c_str();
          sensor_err_log.publish(mycharp);
        }

        dht_err_flag = true;
      }
      else {
        dht_err_flag = false;
      }
      SensorData data;
      if(dht_err_flag == false) {
        data.humidity = h;
        data.temp = t;
        data.pressure = 0;
        data.adc = 0;
      }
      else {
        data.humidity = 0;
        data.temp = 0;
        data.pressure = 0;
        data.adc = 0;
      }
      return data;
    }

    SensorData getDS18B20Data()
    {
      uint8_t pin = (uint8_t)atoi(config.ds_pin);
      OneWire oneWire(pin);
      // Pass our oneWire reference to Dallas Temperature.
      DallasTemperature sensors(&oneWire);
      sensors.requestTemperatures();
      float t = sensors.getTempCByIndex(0);
      SensorData data;
      data.humidity = 0;
      data.temp = t;
      data.pressure = 0;
      data.adc = 0;
      return data;
    }

    SensorData getMHZ19Data() {
      SensorData data;

      byte cmd[9] = {0xFF, 0x01, 0x86, 0x00, 0x00, 0x00, 0x00,
0x00, 0x79};
      // command to ask for data
      char response[9]; // for answer

      MH_Z19Serial.write(cmd, 9); //request PPM CO2
      MH_Z19Serial.readBytes(response, 9);

      if (response[0] != 0xFF)
      {
          Serial.println("Wrong starting byte from Z19 co2 
sensor!");
        }

        if (response[1] != 0x86)
        {
          Serial.println("Wrong command from Z19 co2 sensor!");
        }

        int responseHigh = (int) response[2];
        int responseLow = (int) response[3];
        int ppm = (256 * responseHigh) + responseLow;

        if(ppm < 100 || ppm > 15000) {
          Serial.println("Too much ppm value. Reset ESP");
          ESP.restart();
        }
        data.co2 = ppm;
        return data;
      }

      SensorData getAnalogSensorData()
      {
        float a = analogRead(A0);
        SensorData data;
        String str(config.adc_type);

        if (str == "mq135") {
          #define RZERO 324
          MQ135 gasSensor = MQ135(A0);
          //      float rzero = gasSensor.getCorrectedRZero(19,34);
          //      delay(3000);
          float ppm = gasSensor.getCorrectedPPM(19, 34);
          delay(1000);
          data.adc = ppm;
        }
        else
        {
          data.adc = a;
        }
        data.humidity = 0;
        data.temp = 0;
        data.pressure = 0;
        return data;
      }

      SensorData getBoschData()
      {

        SensorData data;
        String bosch_type(config.sensor_bosch_type);

        if (bosch_type == "bme280") {
          if (bme280initialized) {
            data.temp = bme.readTemperature();
            data.pressure = bme.readPressure() * 0.007500637554192;
            data.humidity = bme.readHumidity();
            data.adc = 0;
            return data;
          }
        }

        else if (bosch_type == "bmp180") {
          if (bmp180initialized)
          {
            data.temp = bmp.readTemperature();
            data.pressure = bmp.readPressure() * 0.007500637554192;
            data.humidity = 0;
            data.adc = 0;
          }
          return data;
        }
      }

      void renderRowValue(String value, int row, int r = 0, int g =
  255, int b = 0)
      {
        if (isRebooting)
        {
          //do nothing if rebooting
          return;
        }

        while (value.length() < 10)
        {
          value += " ";
        }
      }
      void sendSensorsData()
      {
        // Use WiFiClient class to create TCP connections
        WiFiClient client;
        const int httpPort = 80;
        if (!client.connect(host, httpPort)) {
          Serial.println("connection failed");
          return;
        }
        String url = "/update?key=";
        url += config.thing_speak_api_key;

        if (atoi(config.sensor_dht_on) == 1 && dht_err_flag ==
  false) {
          String dht_temp = (String)getTempForJson(data1.temp),
                 dht_humidity =
  (String)getHumidityForJson(data1.humidity);
          url += "&";
          url += config.dht_t_field;
          url += "=";
          url += dht_temp;
          url += "&";
          url += config.dht_h_field;
          url += "=";
          url += dht_humidity;
        }

        if (atoi(config.sensor_ds18b20_on) == 1) {
          String ds18b20_temp = (String)getTempForJson(data2.temp);

          url += "&";
          url += config.ds18b20_field;
          url += "=";
          url += ds18b20_temp;
        }

        if (atoi(config.sensor_bosch_on) == 1) {
          String bosch_temp = (String)getTempForJson(data3.temp),
                 bosch_pressure =
(String)getPressureForJson(data3.pressure);

          url += "&";
          url += config.bosch_t_field;
          url += "=";
          url += bosch_temp;

          url += "&";
          url += config.bosch_p_field;
          url += "=";
          url += bosch_pressure;

          String bosch_type(config.sensor_bosch_type);
          if (bosch_type == "bme280") {
            String bosch_hum =
(String)getHumidityForJson(data3.humidity);
            url += "&";
            url += config.bosch_h_field;
            url += "=";
            url += bosch_hum;
          }
        }

        if (atoi(config.sensor_analog_on) == 1) {
          String adc_val = (String)data4.adc;
          url += "&";
          url += config.adc_field;
          url += "=";
          url += adc_val;
        }

        if (atoi(config.sensor_mhz19_on) == 1) {
          String co2_val = (String)data5.co2;
          url += "&";
          url += config.mh_z19_field;
          url += "=";
          url += co2_val;
        }
```

```
//
//  if (atoi(config.sensor_mhz14a_on) == 1) {
//    String co2_val1 = (String)data6.co2;
//    url += "&";
//    url += config.mh_z14a_field;
//    url += "=";
//    url += co2_val1;
//  }


  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  delay(10);

  // Read all the lines of the reply from server and print
them to Serial
  while (client.available()) {
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }
}

void sendNarodmon()
{
  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 8283;
  String fw_ver(config.firmware_ver);

  if (!client.connect("narodmon.ru", httpPort)) {
    Serial.println("connection failed");
    return;
  }

  client.print("#");
  client.print(WiFi.macAddress()); // MAC ESP8266
  client.print("#");
  client.print("ESP8266 [MonSystem v."); // Назва пристрою
  client.print(fw_ver);
  client.print(" ]");
  client.println();

  if (atoi(config.sensor_dht_on) == 1 && dht_err_flag ==
false) {
    String dht_temp = (String)getTempForJson(data1.temp),
           dht_humidity =
(String)getHumidityForJson(data1.humidity);

    client.print("#T1#");
    client.print(dht_temp);
```

```
        client.print("#DHT Temp#");
        client.println();
        client.print("#H1#");
        client.print(dht_humidity);
        client.print("#DHT Humidity#");
        client.println();
      }

      if (atoi(config.sensor_ds18b20_on) == 1) {
        String ds18b20_temp = (String)getTempForJson(data2.temp);

        client.print("#T2#");
        client.print(ds18b20_temp);
        client.print("#DS18B20 Temp#");
        client.println();
      }

      if (atoi(config.sensor_bosch_on) == 1) {
        String bosch_temp = (String)getTempForJson(data3.temp),
               bosch_pressure =
(String)getPressureForJson(data3.pressure);

        client.print("#T3#");
        client.print(bosch_temp);
        client.print("#BMP/E Temp#");
        client.println();
        client.print("#P1#");
        client.print(bosch_pressure);
        client.print("#BMP/E Pressure#");
        client.println();

        String bosch_type(config.sensor_bosch_type);
        if (bosch_type == "bme280") {
          String bosch_hum =
(String)getHumidityForJson(data3.humidity);
          client.print("#H1#");
          client.print(bosch_hum);
          client.print("#BME Humidity#");
          client.println();
        }
      }

      if (atoi(config.sensor_analog_on) == 1) {
        String str(config.adc_type);
        String adc_val = (String)data4.adc;

        if (str == "mq135") {
          client.print("#CO2#");
          client.print(adc_val);
          client.print("#MQ135 (CO2)#");
          client.println();
        }
        else {
          client.print("#ADC#");
          client.print(adc_val);
```

```
            client.print("#ADC pin#");
            client.println();
          }
        }

        if (atoi(config.sensor_mhz19_on) == 1) {
          String co2_val = (String)data5.co2;
          client.print("#MH-Z19#");
          client.print(co2_val);
          client.print("#CO2, ppmT#");
          client.println();
        }

      //  if (atoi(config.sensor_mhz14a_on) == 1) {
      //    String co2_val = (String)data6.co2;
      //    client.print("#MH-Z14A#");
      //    client.print(co2_val);
      //    client.print("#CO2, ppmT#");
      //    client.println();
      //  }

        client.println("##");
        Serial.println("\r Sensors data send to narodmod.ru \r");
        // Read all the lines of the reply from server and print
them to Serial
        while (client.available()) {
          String line = client.readStringUntil('\r');
          Serial.print(line);
        }
      }
```