Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему:

Програмне забезпечення мікропроцесорної IoT-системи моніторингу та керування Smart Home з використанням Node-RED середовища

КбПЗ_2026

Виконав здобувач вищої освіти
IV курсу, групи КІ-22ПЗ
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____Босько І.В.
«____» _____ 2026 р.

Керівник проекту
кандидат технічних наук, доцент
_____Кислун О.А.
«____» _____ 2026 р.
Рецензент _____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *бакалавр*
Галузь знань ._12 _____*“*Інформаційні технології*”*_____
Спеціальність *123 “*Комп’ютерна інженерія*”*
Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
“____ ” _____ 2026 р.

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Босько Ірині Вікторівні
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення мікропроцесорної IoT-системи моніторингу та керування Smart Home з використанням Node-RED середовища*
2. Керівник роботи *Кислун Олег Андрійович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від “27” лютого 2026 року №130-02
3. Строк подання роботи до захисту *10.06.2026 р.*
4. Мета та завдання випускної кваліфікаційної роботи. *Метою роботи є розробка мікропроцесорної IoT-системи моніторингу та керування Smart Home*
5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію
6. Висновки.
6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритмів системи моніторингу	*2 аркуші*

7. Дата видачі завдання «27» лютого 2026 р.

КПІ 2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем керування	10.03.2026	
2.	Постановка задачі, оформлення ТЗ	15.03.2026	
3.	Розробка моделі компонента	20.03.2026	
4.	Розробка структур даних	25.03.2026	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2026	
6.	Програмування алгоритмів	10.04.2026	
7.	Оформлення ПЗ	17.05.2026	
8.	Попередній захист роботи	23.05.2026	

КОПЗ_2026

Дата видачі завдання
«27» лютого 2026 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«27» лютого 2026 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Босько І.В. Програмне забезпечення мікропроцесорної IoT-системи моніторингу та керування Smart Home з використанням Node-RED середовища. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2026.

У кваліфікаційній бакалаврській роботі розглянуто розроблення програмного забезпечення мікропроцесорної IoT-системи моніторингу та керування Smart Home із використанням середовища Node-RED. Актуальність теми обумовлена зростаючим поширенням технологій Інтернету речей та необхідністю створення інтелектуальних систем автоматизації житлових приміщень, які забезпечують підвищення комфорту, енергоефективності та безпеки.

У роботі проведено аналіз сучасних IoT-рішень для систем «розумного будинку», досліджено принципи функціонування мікропроцесорних систем моніторингу та керування, а також обґрунтовано вибір програмних і апаратних засобів для реалізації системи. Особливу увагу приділено використанню середовища Node-RED для організації взаємодії між пристроями та реалізації логіки керування.

Розроблено програмне забезпечення, яке забезпечує збір та обробку даних із датчиків, моніторинг параметрів середовища, дистанційне керування виконавчими пристроями та візуалізацію інформації через користувацький інтерфейс.

Отримані результати можуть бути використані для створення сучасних систем домашньої автоматизації та впровадження технологій Інтернету речей у житлових і побутових приміщеннях.

Ключові слова: IoT, Smart Home, Node-RED, мікропроцесорна система, автоматизація, моніторинг, керування, Інтернет речей.

КбПЗ-2026

ABSTRACT

Bosko I. Software for a microprocessor-based IoT monitoring and control system for Smart Home using the Node-RED environment. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

The qualification bachelor's thesis considers the development of software for a microprocessor IoT monitoring and control system Smart Home using the Node-RED environment. The relevance of the topic is due to the growing spread of Internet of Things technologies and the need to create intelligent automation systems for residential premises that provide increased comfort, energy efficiency and security.

The paper analyzes modern IoT solutions for "smart home" systems, investigates the principles of operation of microprocessor monitoring and control systems, and justifies the choice of software and hardware for implementing the system. Particular attention is paid to the use of the Node-RED environment to organize interaction between devices and implement control logic.

Software has been developed that provides data collection and processing from sensors, monitoring of environmental parameters, remote control of actuators and visualization of information through a user interface.

The results obtained can be used to create modern home automation systems and implement Internet of Things technologies in residential and household premises.

Keywords: IoT, Smart Home, Node-RED, microprocessor system, automation, monitoring, control, Internet of Things.

КОПЗ_2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 3
1.1 Призначення системи..... 5
1.2 Область застосування..... 12
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 13
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським рівнем вищої освіти) 13
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 35
2.3 Розгорнута постановка завдання 37
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 38
3.1 Опис функціонування системи 38
3.2 Розробка структурної схеми..... 39
3.3 Розробка функціональної схеми 40
3.4 Розробка діаграми процесів..... 45
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 46
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 46
4.2 Захист розробленого програмного забезпечення..... 67
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 71
6 ОСНОВНІ ВИСНОВКИ..... 77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 79

КПЗ – 2026

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

MQTT	–	Message Queuing Telemetry Transport
МК	–	мікроконтролер (microcontroller)
WiFi	–	Wireless Fidelity
Arduino Uno	–	апаратно-програмна платформа на МК ATmega328
AVR	–	лінійка мікроконтролерів фірми Atmel
LCD	–	рідкокристалічний індикатор/дисплей
RAM (ОЗП)	–	перативний запам’ятовуючий пристрій
ПДУ	–	пульти дистанційного управління
ПЗ	–	програмне забезпечення
ПЛК	–	програмувальні логічні контролери
USART	–	універсальний синхронний/асинхронний приймач та
	–	передавач
I^2C	–	послідовна шина даних для зв’язку інтегральних схем, що використовує дві двонаправлені лінії зв’язку (SDA і SCL).

КбПЗ 2026

ВСТУП

У сучасних умовах технології розумного дому все активніше інтегруються у повсякденне життя людей, забезпечуючи підвищення рівня комфорту, енергоефективності та безпеки. Основою функціонування таких систем є технології Інтернету речей (IoT), які дозволяють об’єднувати різноманітні датчики та пристрої в єдину мережу для обміну інформацією.

Важливим елементом таких систем є використання мікроконтролерів, які виконують функції збору даних із сенсорів, їх обробки та передавання через мережу. Одним із популярних рішень для реалізації IoT-проєктів є мікроконтролер ESP32, який має вбудовані модулі Wi-Fi та Bluetooth, достатню обчислювальну потужність і підтримку широкого спектра периферійних пристроїв.

Для організації обміну даними між пристроями часто застосовується протокол MQTT, який характеризується невеликим обсягом службової інформації, низьким енергоспоживанням та високою ефективністю при роботі з IoT-пристроями. Для візуалізації отриманих даних та створення інтерфейсу користувача зручно використовувати платформу Node-RED, яка дозволяє реалізовувати обробку даних та побудову інформаційних панелей за допомогою графічного редактора.

Актуальність теми зумовлена зростаючою потребою у створенні інтелектуальних систем моніторингу житлового середовища, які забезпечують оперативне отримання інформації про стан приміщення та можуть бути основою для подальшої автоматизації управління.

Серед численних пристроїв, що застосовуються в IoT-системах, важливе місце займають мікроконтролери, які виконують функції збору, обробки та передавання даних. Одними з найпоширеніших є мікроконтролери ESP32, що поєднують достатню обчислювальну продуктивність, підтримку бездротових

КОПЗ_2026

інтерфейсів зв’язку та відносно невисоку вартість.

Дана бакалаврська кваліфікаційна робота присвячена створенню IoT-системи для моніторингу параметрів внутрішнього середовища, зокрема температури, вологості, концентрації газу та наявності руху. Система побудована на базі мікроконтролера ESP32, використовує протокол MQTT для передавання даних та забезпечує їх візуалізацію за допомогою платформи Node-RED.

Особливістю цієї роботи є те, що розроблення та тестування всієї системи виконувалося у віртуальному середовищі Wokwi без застосування реального апаратного обладнання. Це дало можливість ефективно реалізувати, перевірити та продемонструвати працездатність створеної системи.

КбПЗ_2026

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розроблена IoT-система розумного дому призначена для моніторингу основних параметрів внутрішнього середовища житлового приміщення та відображення отриманих даних у зручному інтерфейсі користувача. Система здійснює автоматичний збір інформації з підключених сенсорів, її обробку, передачу через мережу та подальшу візуалізацію у режимі реального часу. Основними параметрами, які контролює система, є температура повітря, відносна вологість, рівень газу (як показник якості повітря) та наявність руху у приміщенні. Отримані дані передаються за допомогою протоколу MQTT на сервер, де вони можуть бути використані для подальшої обробки, зберігання або відображення у вигляді інформаційної панелі. Візуалізація даних здійснюється за допомогою платформи Node-RED, яка дозволяє створити зручний вебінтерфейс для моніторингу показників сенсорів. Це дає можливість користувачу спостерігати за станом внутрішнього середовища у реальному часі, що сприяє підвищенню комфорту та безпеки проживання.

Розумний дім може самостійно керувати освітленням, підтримувати необхідну температуру та рівень вологості, контролювати доступ до приміщень, відстежувати можливі витоки газу чи води, інформувати про аварійні ситуації, а також сприяти раціональному використанню ресурсів.

Основною особливістю таких систем є те, що більшість процесів відбувається без безпосередньої участі людини - управління здійснюється автоматично або дистанційно за допомогою смартфона, планшета чи комп’ютера.

КОПЗ_2026

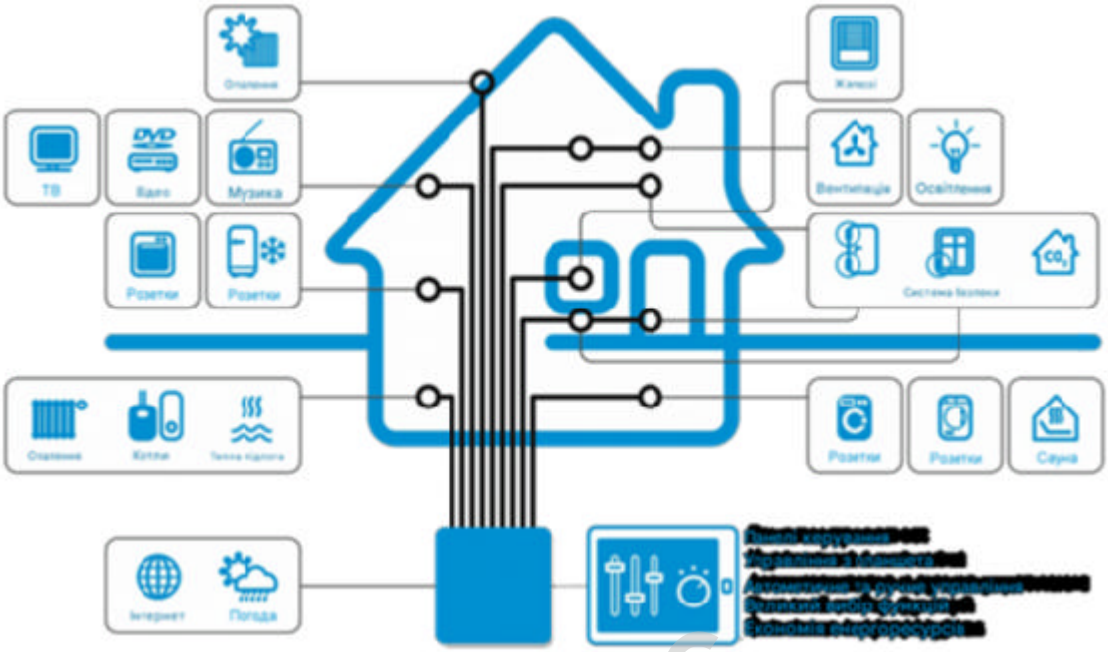


Рисунок 1.1 - Схематичне зображення системи розумного дому

Для створення подібної системи застосовують комплекс датчиків та виконавчих пристроїв, які інтегруються в мережу та взаємодіють між собою під керуванням програмного забезпечення, що виконує централізований аналіз і обробку даних.

Сенсори безперервно відстежують параметри навколишнього середовища, зокрема температуру, вологість або наявність руху. На основі отриманих даних виконавчі пристрої виконують відповідні дії: можуть активувати систему опалення, відкрити жалюзі або надіслати повідомлення користувачеві. Обмін інформацією між елементами системи відбувається через дротові або бездротові протоколи зв'язку, серед яких найбільш поширеними є Wi-Fi, Zigbee, Z-Wave та MQTT.

Для керування системою зазвичай використовуються мобільні застосунки або вебінтерфейси, які дозволяють користувачеві в режимі реального часу контролювати стан пристроїв, переглядати журнал подій та налаштовувати сценарії автоматичної роботи. Наприклад, у разі фіксації руху в нічний час система може автоматично ввімкнути освітлення в коридорі, а при виявленні диму - надіслати сповіщення на смартфон та відключити електроживлення.

КОПЗ_2026

Таким чином, розумний дім є практичним прикладом застосування технологій Інтернету речей у повсякденному житті. Використання таких систем робить житло більш комфортним, підвищує рівень безпеки, сприяє ефективному використанню ресурсів і впровадженню сучасних технологічних рішень.

Архітектура IoT-систем і протоколи

Після ознайомлення із загальною концепцією розумного дому доцільно перейти до розгляду технічних засад, що забезпечують його роботу, зокрема архітектури IoT-систем та комунікаційних протоколів [1]. Саме ці компоненти формують інфраструктуру, яка дозволяє пристроям взаємодіяти між собою, передавати та зберігати інформацію, а також виконувати команди користувача.

Типова система Інтернету речей у розумному домі має багаторівневу архітектуру. На нижньому рівні розташовані сенсори та актуатори — фізичні пристрої, що відповідають за зчитування параметрів навколишнього середовища або виконання певних дій. Сенсори вимірюють такі показники, як температура, вологість, наявність руху, концентрація газу, рівень освітленості та інші параметри. Актуатори, своєю чергою, керують різними приладами: можуть вмикати освітлення, відкривати двері, регулювати систему опалення або активувати сигналізацію. Ці пристрої встановлюються безпосередньо у приміщеннях і забезпечують взаємодію між фізичним середовищем та цифровою системою керування.

Наступним рівнем є мережевий або транспортний рівень, який забезпечує обмін даними між пристроями та контролерами. Для цього використовуються різні типи зв’язку - як дротові (Ethernet), так і бездротові (Wi-Fi, Zigbee, Bluetooth Low Energy). Бездротові технології домінують у домашньому середовищі завдяки простоті встановлення та гнучкості. Зокрема, WiFi дозволяє швидко інтегрувати пристрої до локальної мережі з доступом до Інтернету, що є зручним для передачі даних у реальному часі.

Найважливішим елементом у цій архітектурі є контролер або шлюз - наприклад, мікроконтролер ESP32. Він виконує роль логічного ядра: зчитує дані з

КБПЗ_2026

сенсорів, формує повідомлення, надсилає їх на сервер або хмарну платформу та приймає команди назад. У простих випадках роль сервера може виконувати сам ESP32, але частіше використовується MQTT-сервер - спеціальний сервер, що за маршрутизацію повідомлень між пристроями, панеллю керування та іншими підписниками.

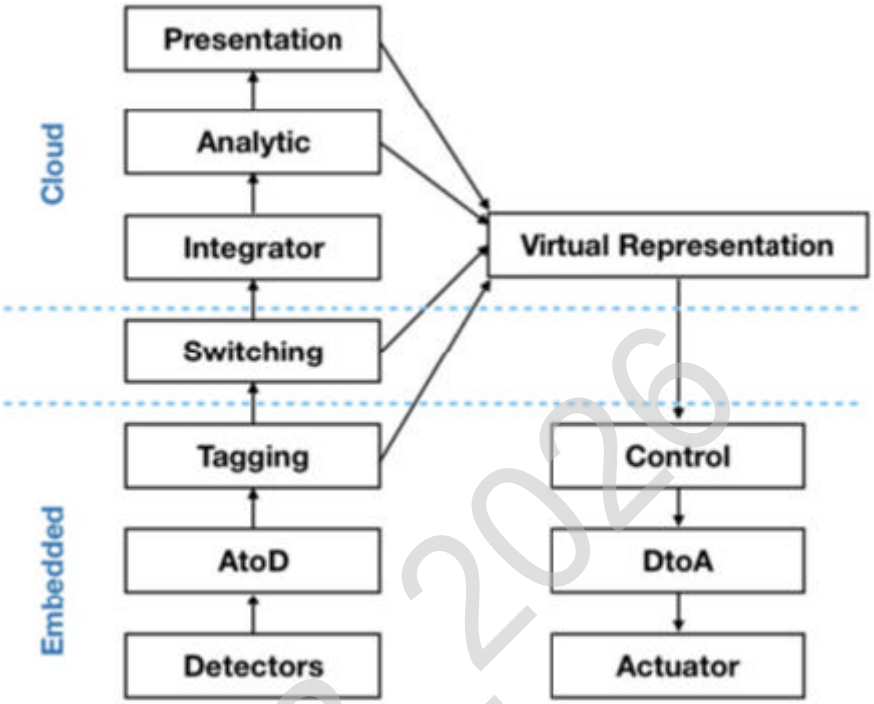


Рисунок 1.2 - Багаторівнева архітектура IoT-системи

Одним із найпоширеніших протоколів обміну даними, що використовується у системах розумного дому, є MQTT. До його основних переваг належать простота реалізації, низьке енергоспоживання, невеликий обсяг передаваних пакетів і висока надійність передачі інформації. Робота MQTT базується на моделі «публікація–підписка», відповідно до якої пристрої публікують повідомлення у визначені топіки, а всі клієнти, підписані на ці топіки, автоматично отримують відповідні дані. Протокол передбачає декілька рівнів гарантії доставки повідомлень (QoS), а також підтримує механізм retained-повідомлень. Завдяки цьому сервер може зберігати останнє отримане значення для кожного топіка та надсилати його новим підписникам під час підключення. У системах розумного дому така функція дає можливість миттєво отримати

КБГЗ_2026

актуальні дані, наприклад, про температуру або стан датчика руху навіть після перезапуску інтерфейсу.

Разом із MQTT у системах Інтернету речей використовується й інший протокол - HTTP, який широко відомий як основа функціонування вебресурсів. У контексті розумного дому HTTP зазвичай застосовується для виконання разових запитів, наприклад під час налаштування пристрою через вебінтерфейс або передавання даних на сервер із використанням REST-API. Незважаючи на свою універсальність і поширеність, HTTP має певні недоліки: відсутність постійного з'єднання, збільшене навантаження на мережу та недостатню ефективність при частій передачі даних від сенсорів у режимі реального часу.

Ще одним протоколом, що активно застосовується у сфері IoT, є CoAP. Він був спеціально розроблений для пристроїв з обмеженими обчислювальними ресурсами. На відміну від HTTP, CoAP працює поверх протоколу UDP, що дозволяє зменшити обсяг мережевого трафіку. Обмін даними у CoAP відбувається за моделлю «запит–відповідь», а також підтримується механізм спостереження за ресурсами (режим observe), який дозволяє отримувати повідомлення про зміну значень сенсорів. Водночас використання UDP робить цей протокол менш стійким у мережах з нестабільним з'єднанням.

Для організації зв'язку між пристроями в межах локальної мережі часто застосовуються бездротові протоколи короткого радіусу дії, зокрема Zigbee та Z-Wave.

Також широко використовується технологія Bluetooth Low Energy (BLE). Протокол Zigbee є відкритим стандартом із низьким енергоспоживанням, що функціонує на основі mesh-мережі. У такій мережевій структурі пристрої здатні передавати сигнал один через одного, що сприяє збільшенню радіусу покриття та підвищенню надійності зв'язку. Протокол Z-Wave має подібні характеристики, проте є закритим стандартом і потребує ліцензування. Обидві технології широко використовуються в комерційних системах автоматизації, зокрема для керування освітленням, жалюзі, дверними замками та іншими елементами інфраструктури

КПЗ_2026

будинку. Технологія Bluetooth Low Energy здебільшого застосовується для взаємодії з мобільними пристроями або у локальних рішеннях, наприклад у смарт-замках, датчиках біля входу чи персональних маяках.

Серед новітніх стандартів, що швидко набувають поширення, варто відзначити протокол Matter. Це відкритий стандарт, створений для забезпечення сумісності між пристроями різних виробників. Його підтримують провідні технологічні компанії, такі як Apple, Google, Amazon та Samsung. Matter функціонує поверх мережевих технологій IP (TCP/IP) і може використовувати Wi-Fi, Ethernet або Thread як транспортний рівень для передавання даних. Основною метою цього стандарту є спрощення інтеграції різних пристроїв у єдину систему, підвищення надійності взаємодії та забезпечення належного рівня безпеки без необхідності використання спеціалізованих шлюзів від окремих виробників.

У цілому архітектура IoT-системи розумного дому являє собою комплексне поєднання апаратних компонентів, комунікаційних протоколів та програмних інтерфейсів, які разом формують єдину функціональну екосистему. Завдяки правильному вибору та ефективній взаємодії цих елементів забезпечується стабільна, безпечна та гнучка робота системи, що дозволяє автоматизувати житловий простір і підвищити комфорт повсякденного життя користувачів.

Огляд платформ візуалізації даних

Важливою складовою IoT-системи розумного дому є можливість наочної візуалізації інформації, що надходить від сенсорів і підключених пристроїв. Для користувача важливо не лише отримувати числові значення параметрів, наприклад температуру в приміщенні 22,7 °C або сигнал про виявлений рух, а й бачити ці дані у зрозумілому та зручному для сприйняття вигляді. Це може бути відображення інформації у формі графіків, індикаторів, текстових повідомлень або у режимі реального часу. Для реалізації таких можливостей застосовуються спеціалізовані інструменти та програмні платформи візуалізації даних.

Для реалізації цих можливостей застосовуються спеціалізовані платформи візуалізації даних. Вони дають змогу створювати інтерфейси моніторингу,

КБПЗ–2026

відображати історію зміни параметрів у вигляді графіків, отримувати повідомлення про події, а також здійснювати керування підключеними пристроями.

Однією з найпоширеніших платформ для візуалізації та обробки даних є Node-RED [2]. Це середовище з відкритим вихідним кодом, яке дозволяє будувати логіку взаємодії між пристроями у вигляді потоків даних (flows). Node-RED має вбудовану підтримку протоколу MQTT, що забезпечує просту інтеграцію з IoT-пристроями та можливість отримання повідомлень у реальному часі. Користувач може у графічному інтерфейсі розміщувати функціональні вузли (nodes), з'єднувати їх між собою та формувати логіку обробки даних без необхідності написання складного програмного коду. Важливою складовою платформи є модуль Dashboard, який дає змогу створювати вебінтерфейс із різними елементами відображення, такими як індикатори, графіки, кнопки та текстові блоки. Завдяки цьому Node-RED добре підходить для організації локального моніторингу системи розумного дому.

Іншою потужною платформою для візуалізації даних є Grafana. Це програмне рішення призначене для створення інтерактивних панелей керування та графічного аналізу даних. Найчастіше Grafana використовується разом із базами даних часових рядів, такими як InfluxDB або Prometheus. За допомогою цієї платформи можна будувати динамічні графіки зміни різних параметрів, наприклад температури, вологості чи концентрації газу, що особливо корисно для аналізу накопичених даних. Grafana більше орієнтована на роботу з історичними даними, ніж на відображення поточного стану системи. Тому в IoT-системах розумного дому її зазвичай використовують разом з іншими інструментами: дані передаються через MQTT до бази даних (наприклад, InfluxDB), після чого відображаються у Grafana у вигляді аналітичних графіків.

Серед платформ, орієнтованих на початківців, можна відзначити Blynk або Tinkercad Circuits. Вони дозволяють швидко створити прості інтерфейси для керування IoT-пристроями, проте мають певні функціональні обмеження, а

КПЗ – 2026

розширені можливості часто доступні лише у платних версіях. Платформа Blynk призначена для створення мобільних застосунків, за допомогою яких можна здійснювати моніторинг і керування IoT-пристроями у режимі реального часу. Вона підтримує такі апаратні платформи, як Arduino, ESP32, Raspberry Pi та інші, що дає змогу створювати мобільні інтерфейси для керування пристроями без необхідності глибоких знань програмування.

Tinkercad Circuits є онлайн-платформою для моделювання та симуляції електронних схем, зокрема проєктів на основі Arduino. За допомогою цього інструмента можна створювати електронні схеми, програмувати їхню роботу та перевіряти функціонування безпосередньо у браузері. Завдяки таким можливостям платформа широко використовується для навчальних цілей, а також для швидкого прототипування електронних систем.

Отже, платформи візуалізації відіграють важливу роль у структурі IoT-систем. Вони забезпечують не лише відображення отриманих даних, але й дають змогу взаємодіяти з підключеними пристроями, формувати сценарії автоматизації, створювати історичні звіти та надавати користувачеві зрозумілий інтерфейс керування незалежно від складності реалізованої системи розумного дому.

Механізм retained-повідомлень та персистентне збереження

У системах обміну повідомленнями важливе місце займає механізм, відомий як retained-повідомлення [3]. Його суть полягає в тому, що брокер повідомлень (сервер або центральний вузол) зберігає останнє опубліковане повідомлення, яке пов’язане з певною темою, каналом або ідентифікатором. Основним призначенням цього механізму є забезпечення нових клієнтів або підписників системи миттєвим доступом до актуальної інформації, тобто до останнього зафіксованого стану або значення.

Це означає, що під час першого підключення до певного інформаційного каналу клієнт одразу отримує останнє збережене повідомлення, не очікуючи наступної публікації. Важливою особливістю є те, що повідомлення зберігається незалежно від того, чи були активні підписники на момент його публікації. Якщо

КПЗ–2026

ж у той самий канал надсилається нове повідомлення з позначкою збереження, воно автоматично замінює попереднє.

Механізм retained-повідомлень має широкий спектр застосувань. Найчастіше його використовують для фіксації поточного стану пристроїв або сервісів, наприклад для визначення, чи перебуває пристрій у мережі, чи він увімкнений або вимкнений. Завдяки цьому нові системи моніторингу або інтерфейси користувача можуть одразу отримати актуальну інформацію про стан системи без необхідності чекати чергового оновлення даних.

Такі механізми дозволяють новим клієнтам або інтерфейсам моніторингу миттєво отримувати необхідну інформацію. Наприклад, retained-повідомлення можуть використовуватися для збереження останніх показників сенсорів, таких як температура, вологість чи рівень освітленості, що дає змогу новим користувачам одразу отримати актуальні дані системи. Крім того, вони можуть застосовуватися для фіксації поточних налаштувань пристроїв або параметрів системи, щоб під час підключення нові клієнти відразу отримували інформацію про актуальну конфігурацію. Також за їх допомогою можна повідомляти про останню доступну версію програмного забезпечення або інші важливі службові дані.

Ще однією важливою концепцією є персистентне збереження даних. Воно означає здатність системи зберігати інформацію таким чином, щоб вона не втрачалася після завершення роботи програми, вимкнення живлення або інших тимчасових збоїв. Для цього дані записуються у нелетку пам’ять, наприклад на жорсткий диск, SSD або флеш-пам’ять, що дозволяє відновити їх після перезапуску системи. Основними характеристиками персистентного збереження є використання енергонезалежної пам’яті, можливість збереження інформації після перезавантаження та підвищена стійкість до збоїв, що сприяє загальній надійності системи.

Між механізмом retained-повідомлень і персистентним збереженням існує тісний взаємозв’язок. Для коректної роботи retained-повідомлень брокер, який відповідає за їх зберігання, повинен використовувати персистентне сховище. У

КОПЗ_2026

протилежному випадку всі збережені повідомлення будуть втрачені після перезапуску сервера, що фактично зводить нанівець переваги цього механізму та робить його використання малоефективним.

Подібні підходи застосовуються і в інших технологіях. Наприклад, деякі сервери обміну повідомленнями або мережеві протоколи також мають механізми збереження останнього стану. У системах на основі WebSocket сервер може зберігати інформацію про клієнта для відновлення контексту після повторного підключення. Бази даних реального часу можуть підтримувати механізм збереження «останнього відомого значення», яке автоматично передається новим підписникам. Аналогічно системи керування конфігурацією зберігають актуальні параметри налаштувань, щоб нові вузли під час підключення одразу отримували необхідну конфігурацію.

Отже, незважаючи на відмінності у реалізації та термінології, основні принципи збереження останнього стану для нових підписників (retained-повідомлення) та забезпечення довготривалого зберігання даних між сеансами роботи системи (персистентне збереження) мають важливе значення для багатьох розподілених інформаційних систем. Особливо це актуально для систем, що працюють у режимі реального часу або потребують швидкого надання актуальної інформації під час підключення нових клієнтів.

Апаратні компоненти: огляд та застосування в IoT-системах

Функціонування систем Інтернету речей забезпечується взаємодією різноманітних апаратних компонентів, які здійснюють збір, обробку та передачу інформації про параметри навколишнього середовища для подальшого аналізу і прийняття керуючих рішень [4]. Одними з основних елементів таких систем є мікроконтролери та мікропроцесори, які виконують роль центральних обчислювальних вузлів пристроїв. Вони відповідають за виконання програмного коду, обробку отриманих даних і керування підключеними модулями та периферійними пристроями.

Мікроконтролери (MCU), зокрема Arduino Uno на базі ATmega328P,

КПЗ 2026

STM32 або Raspberry Pi Pico на базі мікроконтролера RP2040, являють собою енергоефективні інтегровані системи, що поєднують у одному корпусі процесорне ядро, оперативну пам’ять і набір периферійних інтерфейсів. Завдяки цьому вони добре підходять для керування апаратними пристроями та виконання задач у режимі реального часу. На відміну від них, мікропроцесори (MPU), до яких належать, наприклад, плати серії Raspberry Pi або процесори Intel Atom та Celeron, мають значно більшу обчислювальну потужність і можуть працювати під керуванням повноцінних операційних систем. Водночас такі системи зазвичай потребують використання зовнішньої пам’яті та додаткових компонентів.

Не менш важливу роль у IoT-системах відіграють сенсори (датчики), які перетворюють фізичні параметри навколишнього середовища на електричні сигнали, придатні для подальшої обробки мікроконтролером. До цієї групи належать датчики температури та вологості, наприклад DHT11, BME280 або DS18B20, газові сенсори (MQ-135, Sensirion SGP30), датчики руху — ультразвукові модулі HC-SR04, радарні сенсори, а також інерціальні модулі (IMU), що містять акселерометри та гіроскопи. Крім того, використовуються датчики освітленості (LDR, BH1750), датчики атмосферного тиску (BMP180, сенсори серії MPX) та датчики відстані, які можуть працювати на основі інфрачервоного або лазерного випромінювання. Кожен тип датчика має власні технічні характеристики та застосовується залежно від вимірюваного параметра і необхідної точності вимірювання.

Для забезпечення підключення IoT-пристроїв до мережі застосовуються різні комунікаційні модулі. До них належать Wi-Fi модулі (наприклад, ESP-01, NodeMCU), Bluetooth-модулі (HC-05/HC-06, BLE), модулі стільникового зв’язку (2G/3G/LTE/NB-IoT/LTE-M), LoRa та LoRaWAN-модулі (RA-02, SX1276), Zigbee-модулі (CC2530), а також Ethernet-модулі (ENC28J60). Вибір конкретного модуля визначається вимогами до дальності передавання даних, енергоспоживання, швидкості обміну інформацією та умовами використання системи.

Крім основних компонентів, у IoT-системах важливу роль відіграють

КОПЗ-2026

допоміжні елементи. До них належать джерела живлення (акумулятори, блоки живлення, сонячні панелі), модулі зберігання даних (SD-карти, EEPROM, флеш-пам’ять), інтерфейсні пристрої (дисплеї, кнопки, світлодіоди), а також елементи керування навантаженням, такі як реле та транзистори. Окрім цього, для захисту електронних компонентів і забезпечення зручності експлуатації використовуються корпуси, монтажні елементи та інші конструкційні деталі.

Вибір апаратних компонентів для реалізації IoT-системи завжди є результатом компромісу між функціональними вимогами, продуктивністю, енергоефективністю, вартістю та умовами експлуатації. Розуміння можливостей і характеристик різних апаратних рішень дозволяє створювати ефективні та гнучкі системи Інтернету речей для широкого спектра практичних задач.

1.2 Область застосування

Тема проєкту, що передбачає розробку IoT-системи для моніторингу параметрів внутрішнього середовища, зокрема температури, вологості, якості повітря та наявності руху, з подальшою візуалізацією отриманих даних за допомогою платформи Node-RED, є актуальною у зв’язку зі зростаючим поширенням технологій розумного дому. Реалізація такої системи дозволяє дослідити принципи побудови сучасних IoT-рішень і на практиці застосувати технології збору, передавання та обробки даних.

Розробка подібної системи сприяє створенню комфортного, безпечного та енергоефективного житлового середовища. Отримані під час виконання роботи знання та створений прототип можуть слугувати основою для подальшого розширення функціональних можливостей системи, зокрема впровадження автоматичного керування кліматом, систем безпеки, енергоменеджменту та комплексного моніторингу стану приміщення. Це відповідає сучасним тенденціям розвитку технологій розумного дому та цифровізації житлового простору.

КПЗ_2026

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським рівнем вищої освіти)

Мікроконтролер: ESP32

ESP32 - це потужний і багатофункціональний мікроконтролерний модуль, який оснащений двоядерним процесором, має вбудовані модулі бездротового зв'язку Wi-Fi та Bluetooth, велику кількість GPIO-виводів, а також підтримує різноманітні периферійні інтерфейси для підключення зовнішніх пристроїв.

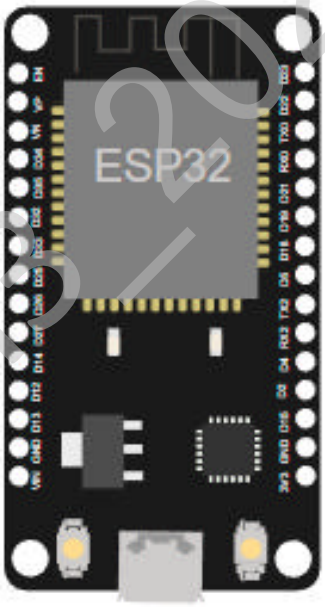


Рисунок 2.1- Мікроконтролер: ESP32

Принцип роботи. ESP32 виконує роль центрального керуючого елемента системи, забезпечуючи збір інформації від підключених сенсорів, її обробку, встановлення мережевого з'єднання та передавання даних через протокол MQTT. Завдяки двоядерній архітектурі мікроконтролер може виконувати кілька завдань одночасно, що сприяє підвищенню ефективності та стабільності роботи системи.

КПЗ 2026

Наявність вбудованих модулів Wi-Fi та Bluetooth забезпечує зручну організацію бездротового обміну даними між пристроями.

Причини вибору ESP32

Інтегрований модуль Wi-Fi

Однією з головних вимог до системи є можливість прямого підключення до локальної мережі та передавання даних на MQTT-сервер без використання додаткових комунікаційних модулів. ESP32 має вбудований Wi-Fi-модуль, що значно спрощує апаратну структуру системи.

Достатня обчислювальна продуктивність і обсяг пам'яті

Мікроконтролер забезпечує ефективну обробку інформації, що надходить від кількох сенсорів, а також підтримує роботу мережевих протоколів і сервісів без істотних затримок.

Гнучкість підключення та велика кількість GPIO-виводів

Наявність значної кількості універсальних входів-виходів дає змогу підключати різноманітні сенсори та виконавчі пристрої, а також забезпечує можливість подальшого розширення функціональності системи.

Підтримка Arduino IDE та широкої екосистеми бібліотек

ESP32 сумісний із середовищем Arduino IDE, що значно спрощує процес розробки програмного забезпечення. Наявність великої кількості готових бібліотек дозволяє швидко реалізувати роботу з датчиками, мережевими протоколами та іншими компонентами системи.

Енергоефективність та підтримка режимів енергозбереження

Хоча в даному проєкті не передбачено автономного живлення, можливість переходу мікроконтролера в режими сну є важливою характеристикою для подальших удосконалень або використання системи у мобільних IoT-пристроях.

Порівняння з альтернативними рішеннями

Arduino Uno у поєднанні з ESP8266

Використання Arduino Uno для обробки даних разом із модулем ESP8266 для організації Wi-Fi-зв'язку є можливим варіантом реалізації системи. Однак

КОПЗ 2026

таке рішення ускладнює схему підключення та програмну взаємодію між модулями. ESP32 об’єднує обчислювальні можливості та бездротовий зв’язок в одному мікроконтролері, що робить систему компактнішою та простішою у реалізації.

Raspberry Pi Zero W

Ця платформа має вищу обчислювальну продуктивність, проте характеризується більшим енергоспоживанням і потребує використання повноцінної операційної системи. Для задачі збору та передавання даних від сенсорів така потужність є надлишковою. У цьому випадку ESP32 є більш енергоефективним рішенням, яке швидше запускається та краще підходить для типових IoT-застосувань.

Датчик температури та вологості: DHT22

DHT22 - це цифровий сенсор для вимірювання температури та відносної вологості повітря, який забезпечує достатньо високу точність у широкому діапазоні значень. Для визначення вологості використовується ємнісний елемент, а вимірювання температури здійснюється за допомогою термістора. Отримані дані передаються до мікроконтролера через однопровідний цифровий інтерфейс.

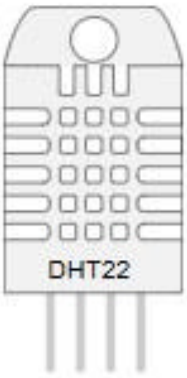


Рисунок 2.2- Датчик температури та вологості: DHT22

Принцип роботи

Датчик періодично вимірює температуру та відносну вологість навколишнього середовища, після чого перетворює отримані значення у цифровий

КбПЗ 2026

сигнал. Ці дані передаються до мікроконтролера, який може їх зчитувати та використовувати для подальшої обробки або передачі в систему моніторингу.

Причини вибору DHT22

Комбіноване вимірювання параметрів

Датчик дозволяє одночасно визначати температуру та вологість повітря, що зменшує кількість необхідних компонентів та спрощує схему підключення.

Достатня точність для систем розумного дому

Точність вимірювання становить приблизно ±0,5 °C для температури та ±2–5% відносної вологості, що є достатнім для більшості задач моніторингу мікроклімату у житлових приміщеннях.

Широкий діапазон вимірювань

DHT22 може працювати в температурному діапазоні від –40 °C до +80 °C та вимірювати вологість у межах від 0 до 100 % RH, що повністю покриває типові умови внутрішнього середовища.

Простота підключення та програмування

Для датчика існує велика кількість готових бібліотек для Arduino IDE, що значно спрощує його інтеграцію в IoT-проєкти та зменшує час розробки.

Порівняння з альтернативами

DHT11

Цей датчик є дешевшим, однак має нижчу точність вимірювань (близько ±2 °C для температури та ±5 % RH для вологості), а також працює у більш обмеженому діапазоні. Тому для задач точнішого контролю параметрів мікроклімату доцільніше використовувати саме DHT22.

BME280 - цей датчик забезпечує вищу точність вимірювань і має розширений функціонал, оскільки, крім температури та вологості, також визначає атмосферний тиск. Проте він є дорожчим і використовує інтерфейс I2C для обміну даними, що може бути дещо складнішим для початківців. З огляду на це, для базового моніторингу параметрів температури та вологості використання датчика DHT22 є більш доцільним з точки зору співвідношення вартості та

КПЗ–2026

функціональних можливостей.

Газовий датчик: MQ-2

MQ-2 - це аналоговий газовий сенсор, який здатний виявляти різні види горючих газів і дим, зокрема пропан, бутан, LPG та інші леткі речовини. Принцип його роботи базується на використанні нагрівального елемента, що підтримує необхідну температуру чутливого шару. Опір цього шару змінюється залежно від концентрації газів у повітрі, що дозволяє визначати їх наявність і приблизний рівень концентрації.

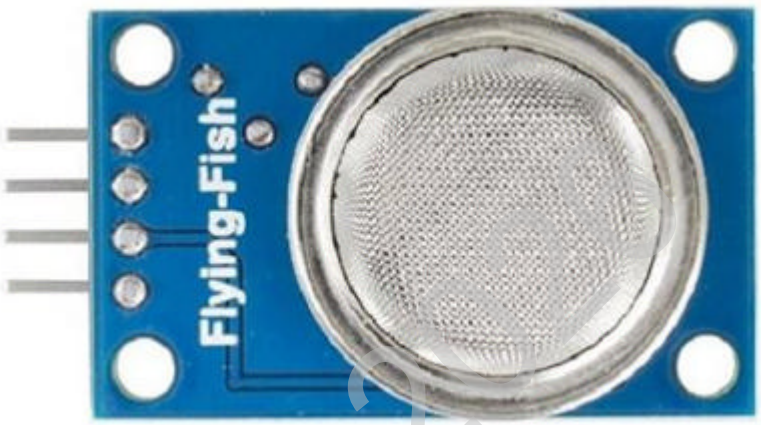


Рисунок 2.3- Газовий датчик: MQ-2

Принцип роботи. Зміна концентрації газів у повітрі впливає на опір чутливого елемента датчика. У результаті цього змінюється рівень вихідної напруги. Отриманий аналоговий сигнал зчитується аналоговим входом мікроконтролера ESP32 і може бути використаний для оцінювання рівня загазованості або наявності диму.

Причини вибору MQ-2 - Низька вартість. Датчик є доступним за ціною, що робить його доцільним для використання на етапі розробки прототипу та демонстрації можливостей системи моніторингу якості повітря.

Простота підключення

MQ-2 легко інтегрується в систему, оскільки його вихідний сигнал може бути безпосередньо підключений до аналогового входу мікроконтролера, що спрощує реалізацію апаратної частини проєкту.

КПЗ_4

Чутливість до різних газів

Датчик здатний реагувати на широкий спектр горючих газів, що дозволяє виявляти загальні зміни якості повітря або появу потенційно небезпечних газових домішок у приміщенні.

Порівняння з альтернативами

Цифрові сенсори якості повітря (наприклад, SGP30). Такі датчики забезпечують точніші та калібровані вимірювання летких органічних сполук (VOCs) і еквівалентної концентрації CO_2. Крім того, вони мають вбудовані механізми компенсації впливу температури та вологості. Проте подібні рішення є значно дорожчими та можуть потребувати складнішої програмної обробки даних. Тому для базового виявлення змін якості повітря використання датчика MQ-2 є достатньо ефективним і економічно доцільним.

Датчик руху: пасивний інфрачервоний (PIR) сенсор

PIR-сенсор є електронним пристроєм, призначеним для виявлення руху об’єктів на основі змін інфрачервоного (теплового) випромінювання. Він реагує на переміщення об’єктів із температурою, відмінною від температури навколишнього середовища, зокрема людей або тварин, у межах зони його чутливості.

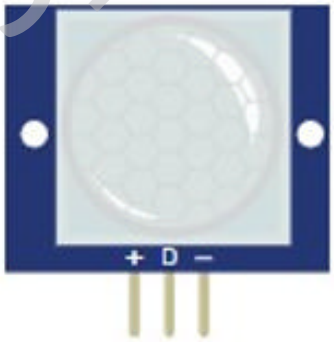


Рисунок 2.4-Датчик руху

Принцип роботи

Сенсор фіксує зміни рівня інфрачервоного випромінювання. При появі рухомого теплого об'єкта в полі зору сенсора генерується імпульс на його вихідному піні (зазвичай цифровий сигнал HIGH).

КбПЗ–2026

Чому обрано PIR-сенсор

Просте виявлення руху - ефективний та надійний спосіб виявлення присутності в приміщенні.

Низьке енергоспоживання - важливо для потенційних систем з батарейним живленням.

Просте підключення (цифровий вихід) - легко інтегрується з мікроконтролером.

Широке застосування в системах безпеки та автоматизації: Стандартне рішення для виявлення руху в розумних будинках.

Порівняння з альтернативами

Ультразвукові датчики руху: Можуть бути більш чутливими до дрібних рухів, але їхня робота залежить від умов навколишнього середовища (температура, вологість) та вони можуть споживати більше енергії.

Радарні датчики руху: Мають більший діапазон виявлення та меншу чутливість до змін температури, але є дорожчими та можуть споживати більше енергії. Для базового виявлення присутності PIR-сенсор є оптимальним за співвідношенням ціна/продуктивність.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Платформа моделювання: Wokwi

Wokwi - це безкоштовна онлайн-платформа для моделювання та розробки проєктів на основі мікроконтролерів, зокрема ESP32. Вона надає віртуальне середовище, у якому можна створювати електронні схеми, писати програмний код для Arduino та запускати його для перевірки роботи. Також симулятор дозволяє здійснювати моніторинг через віртуальний послідовний порт і підтримує емуляцію мережевих з’єднань.

КбПЗ_2026

Принцип роботи

Wokwi відтворює роботу мікроконтролера ESP32 та підключених до нього електронних компонентів у віртуальному середовищі. Це дає можливість розробникам тестувати програмний код і перевіряти логіку роботи системи без використання реального апаратного обладнання, що особливо зручно на початкових етапах проєктування.

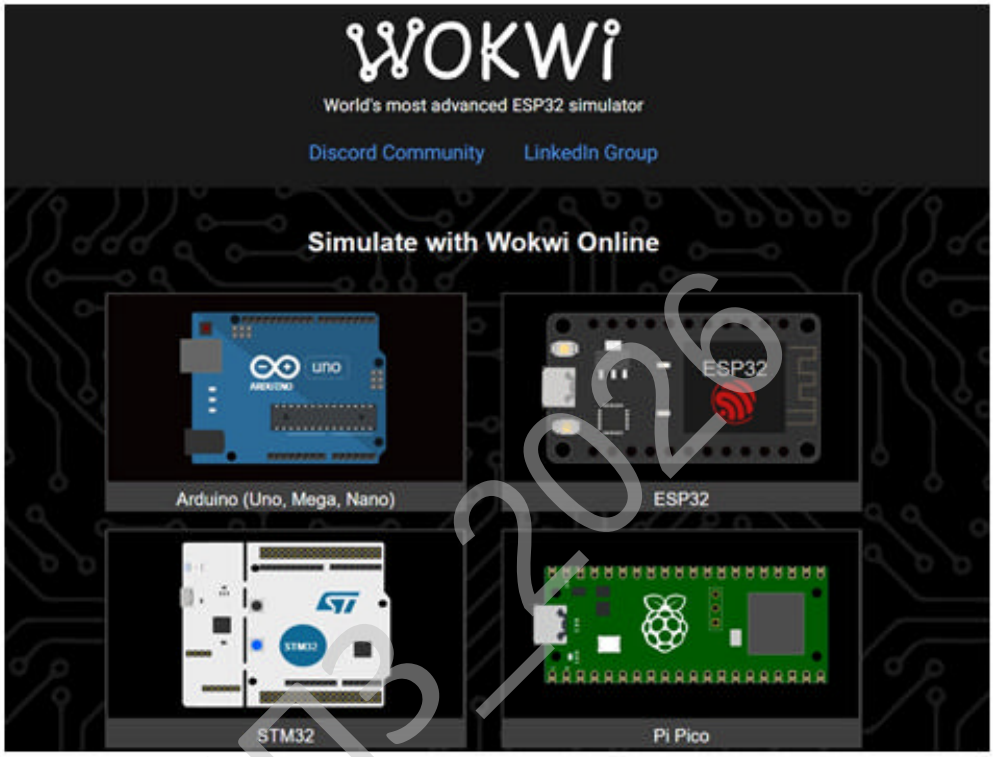


Рисунок 2.5 - Онлайн симулятор Wokwi

Копія 2026

Причини вибору Wokwi

Зручність та доступність

Розробка і тестування проєкту виконуються безпосередньо в онлайн-середовищі, що усуває необхідність встановлення додаткового програмного забезпечення та спрощує роботу з симулятором.

Орієнтація на ESP32

Платформа забезпечує якісну емуляцію роботи мікроконтролера ESP32 та підтримує більшість бібліотек, які використовуються під час програмування цього пристрою.

Можливість моделювання мережевих з’єднань

Середовище дозволяє перевіряти роботу системи з MQTT-сервером і тестувати передачу даних у мережі навіть без фактичного підключення до реального обладнання.

Вбудований серійний монітор

Наявність інтегрованого Serial Monitor спрощує процес налагодження програмного коду та дає змогу відстежувати результати виконання програми.

Порівняння з альтернативними рішеннями

Tinkercad

Цей онлайн-симулятор також підтримує моделювання проєктів на базі Arduino, проте має меншу спеціалізацію для ESP32 і обмежені можливості щодо емуляції мережевих функцій.

Proteus

Proteus є потужним професійним середовищем для моделювання електронних схем, однак воно є платним, потребує встановлення на комп’ютер та може бути складнішим у використанні для початківців.

Протокол обміну даними MQTT

MQTT - це легкий мережевий протокол обміну повідомленнями, що працює за моделлю «видавець–підписник». Він був розроблений спеціально для застосувань Інтернету речей, де пристрої мають обмежені обчислювальні ресурси та можуть працювати в умовах нестабільного мережевого з’єднання.

Принцип роботи

У системі MQTT усі пристрої підключаються до центрального сервера - брокера повідомлень. Пристрої-видавці (publishers) надсилають повідомлення у певні тематичні канали (topics), а пристрої-підписники (subscribers) отримують повідомлення з тих тем, на які вони підписані.

Причини вибору MQTT

Невелике навантаження на систему та низьке енергоспоживання. Протокол має компактний формат повідомлень і добре підходить для використання

КПЗ_2026

на мікроконтролерах з обмеженими ресурсами.

Надійність передачі даних

MQTT підтримує декілька рівнів якості обслуговування (QoS), що дозволяє контролювати доставку повідомлень і забезпечувати їх надійне передавання.

Гнучкість архітектури системи

Використання моделі «видавець–підписник» спрощує масштабування системи та дозволяє легко додавати нові сенсори або сервіси.

Широка підтримка

Для MQTT існує велика кількість клієнтських бібліотек для різних платформ і мов програмування, що значно спрощує інтеграцію протоколу у різні проєкти.

Порівняння з альтернативами

HTTP - цей протокол є більш ресурсомістким і менш ефективним для постійного передавання невеликих обсягів даних, що характерно для IoT-систем.

CoAP - також призначений для систем з обмеженими ресурсами, проте MQTT має ширшу підтримку серед розробників і кращі можливості керування доставкою повідомлень завдяки механізму QoS.

Інструмент візуалізації Node-RED

Node-RED - це середовище візуального програмування, яке базується на концепції потоків даних (flow-based programming). Воно дозволяє створювати логіку обробки інформації та інтерфейси для IoT-систем за допомогою графічного редактора, у якому елементи програми представлені у вигляді вузлів, з’єднаних між собою.

Принцип роботи: У середовищі Node-RED користувач формує логіку роботи системи шляхом з’єднання функціональних вузлів у потоки даних. Такі потоки можуть виконувати різноманітні завдання: підключення до MQTT-брокера, приймання та обробку повідомлень, візуалізацію інформації, а також інтеграцію з іншими сервісами або пристроями.

КПЗ–2026

Причини вибору Node-RED

Візуальний підхід до програмування

Графічний інтерфейс дозволяє створювати логіку обробки даних без написання великої кількості програмного коду, що значно спрощує процес розробки.

Велика бібліотека готових вузлів

Платформа має значну кількість готових модулів для роботи з MQTT, обробки даних, створення інтерфейсів користувача та інтеграції з різними сервісами.

Можливість швидкого прототипування

Node-RED дає змогу оперативно створювати та тестувати прототипи IoT-систем, що особливо корисно на етапі розробки.

Підтримка панелей керування (dashboard)

Платформа містить інструменти для створення вебінтерфейсів, які дозволяють відображати дані від сенсорів у вигляді графіків, індикаторів або текстових повідомлень.

Порівняння з альтернативами

Grafana є потужним інструментом для аналізу та візуалізації часових рядів даних, проте її налаштування може бути складнішим для створення простих IoT-інтерфейсів у порівнянні з Node-RED.

Хмарні IoT-платформи (наприклад, Ubidots або ThingsBoard). Такі сервіси пропонують комплексні рішення для керування IoT-пристроями, однак часто мають обмеження у безкоштовних тарифних планах або потребують постійного підключення до Інтернету. Node-RED, у свою чергу, дозволяє розгортати систему локально та забезпечує більшу гнучкість у налаштуванні.

КБПЗ_2026

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну роботу, реалізації розробка програмного забезпечення системи системи моніторингу та керування Smart Home з використанням Node-RED середовища.

В процесі розробки кваліфікаційної роботи необхідно виконати наступний обсяг проекту:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику розробки відповідної системи та програмного забезпечення. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран результатів аналізу та тестування;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) сформувати висновки про виконаний обсяг робіт та одержані результати.

КОПЗ_2026

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Налаштування середовища Wokwi

Початковим етапом розробки проєкту стало налаштування віртуального середовища Wokwi. Для цього потрібно перейти на офіційний вебсайт платформи та створити новий проєкт. Після цього користувач отримує доступ до віртуального робочого середовища, яке складається з редактора електронних схем і редактора програмного коду.

Структура файлів проєкту в середовищі Wokwi зазвичай містить щонайменше два основні файли:

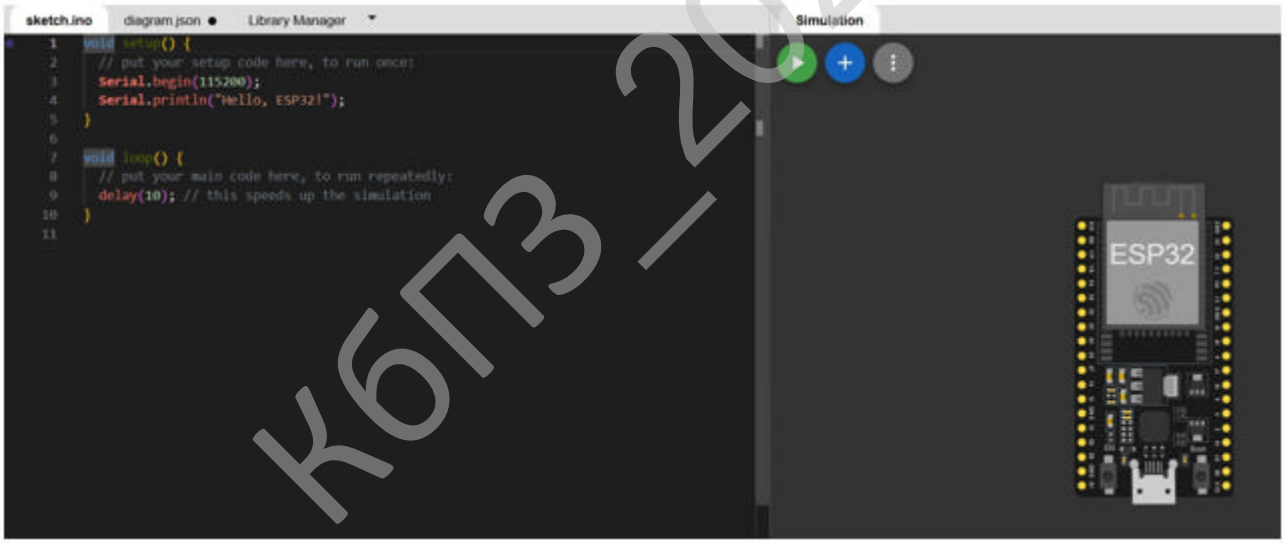


Рисунок 3.1-Новий проект на Wokwi

sketch.ino

Це головний файл проєкту Arduino, у якому розміщується програмний код для мікроконтролера ESP32, написаний мовою C/C++. У цьому файлі реалізується основна логіка роботи системи: ініціалізація підключених компонентів, зчитування даних із сенсорів, встановлення мережевого з’єднання, передавання інформації за протоколом MQTT, а також обробка та збереження локальних даних.

КбПЗ_2026

diagram.json

Цей файл у форматі JSON містить опис віртуальної електронної схеми проєкту. У ньому зберігається інформація про використані електронні компоненти, їхні параметри та з’єднання між окремими елементами. Редагування файлу дає можливість додавати до схеми мікроконтролер ESP32, сенсори (DHT22, газовий датчик, PIR-сенсор) та інші необхідні компоненти, наприклад резистори або конденсатори.

Налаштування емуляції у середовищі Wokwi включає такі етапи

Вибір плати ESP32

У файлі *diagram.json* необхідно вказати модель плати ESP32, яка використовується у проєкті. Платформа Wokwi підтримує кілька варіантів ESP32, що можуть відрізнятися набором доступних контактів та деякими апаратними характеристиками.

Конфігурація компонентів у файлі diagram.json

Для кожного елемента, який додається до віртуальної схеми, у файлі *diagram.json* задаються його тип, унікальний ідентифікатор (ID) та параметри підключення до виводів мікроконтролера ESP32. Наприклад, при підключенні датчика DHT22 необхідно визначити пін ESP32, до якого під’єднаний сигнальний контакт датчика. Аналогічним чином задаються з’єднання для газового датчика та PIR-сенсора. Для газового сенсора, що має аналоговий вихід, зазначається підключення до одного з аналогових входів ESP32, тоді як PIR-сенсор підключається до цифрового входу мікроконтролера.

КПЗ_2026

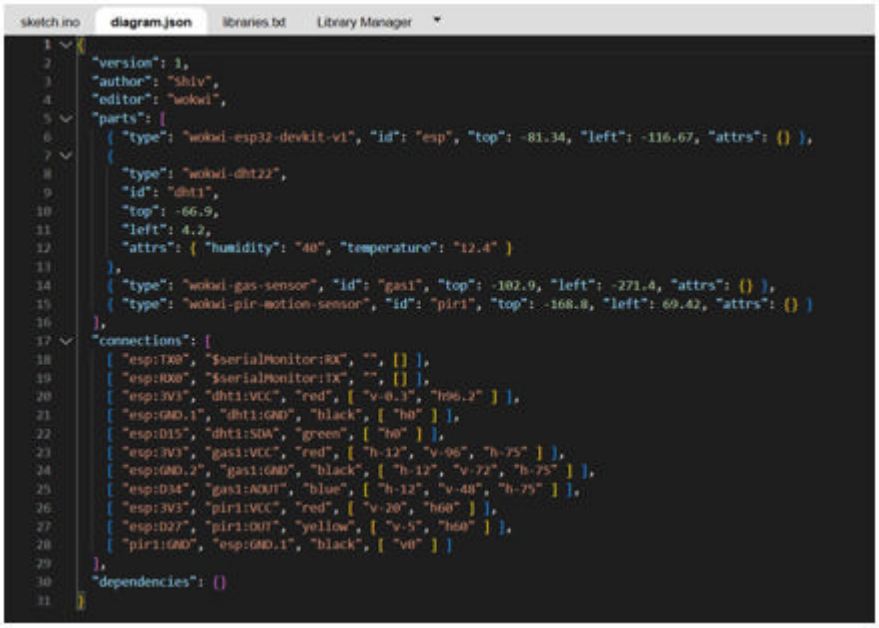


Рисунок 3.2- Файл diagram.json з підключеними елементами

Після створення проєкту та налаштування віртуальної електронної схеми у файлі *diagram.json* можна переходити до розробки програмного коду у файлі *sketch.ino*.

Підключення бібліотек

Для реалізації функціональних можливостей IoT-системи на базі мікроконтролера ESP32 необхідно використовувати спеціалізовані програмні бібліотеки. Вони значно спрощують взаємодію з мережевими протоколами, сенсорами та іншими периферійними пристроями. У даному проєкті застосовуються такі бібліотеки:

WiFi.h

Це стандартна бібліотека, яка входить до складу Arduino ESP32 Core та забезпечує можливість підключення мікроконтролера до бездротових мереж Wi-Fi. Вона містить набір класів і функцій для пошуку доступних мереж, встановлення з’єднання за допомогою SSID і пароля, отримання IP-адреси, а також керування мережевим інтерфейсом ESP32. Використання цієї бібліотеки є необхідним для підключення пристрою до локальної мережі та подальшої взаємодії з MQTT-сервером через Інтернет.

КБПЗ_2026

PubSubClient.h

Ця бібліотека реалізує клієнтську частину протоколу MQTT і широко використовується у проєктах Інтернету речей. Вона надає функції для встановлення з'єднання з MQTT-брокером, публікації повідомлень у певних темах, підписки на необхідні теми, а також обробки отриманих повідомлень за допомогою спеціальної callback-функції.

DHTesp.h

Бібліотека призначена для роботи з датчиками температури та вологості сімейства DHT, зокрема DHT11 і DHT22, на платформі ESP32. Вона містить зручні методи для ініціалізації датчика, отримання значень температури та вологості, а також для обробки можливих помилок під час зчитування даних. Використання цієї бібліотеки значно спрощує роботу з датчиком порівняно з реалізацією протоколу обміну даними вручну.

Preferences.h

Ця бібліотека, яка також входить до складу Arduino ESP32 Core, забезпечує простий механізм для збереження невеликих обсягів даних у енергонезалежній пам'яті мікроконтролера.

Бібліотека **Preferences.h** забезпечує можливість збереження невеликих обсягів даних у енергонезалежній пам'яті мікроконтролера ESP32 (NVS — Non-Volatile Storage). Завдяки цьому можна зберігати важливі параметри конфігурації або останні отримані значення датчиків навіть після перезавантаження пристрою. Бібліотека працює за принципом «ключ–значення» та підтримує різні типи даних, зокрема цілі числа, числа з плаваючою комою та текстові рядки.

Додавання бібліотек до проєкту у середовищі Wokwi може виконуватися двома основними способами:

Через Library Manager

У середовищі Wokwi передбачено вбудований менеджер бібліотек, який дозволяє знаходити та підключати популярні бібліотеки Arduino безпосередньо у вебінтерфейсі. Для цього необхідно відкрити редактор програмного коду та

КПЗ_2026

скористатися відповідною функцією в меню або контекстному меню середовища. Саме цей спосіб був використаний під час виконання даного проєкту.

Копіюванням файлів бібліотеки

Якщо потрібна бібліотека відсутня у менеджері, її файли можна додати вручну, скопіювавши їх у відповідну директорію проєкту Wokwi. Це дозволяє використовувати додаткові або менш поширені бібліотеки у процесі розробки.

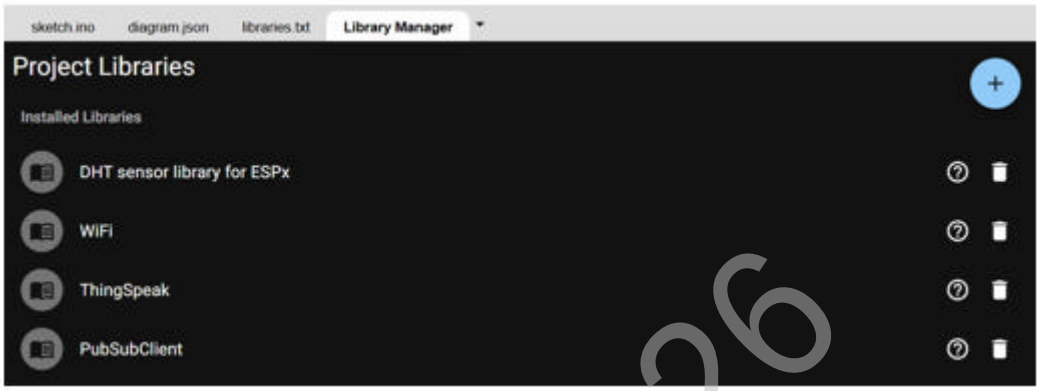


Рисунок 3.3 –Менеджер бібліотек

Після підключення необхідних бібліотек їх можна використовувати у програмному коді шляхом додавання директиви препроцесора #include <назва_бібліотеки.h>. Це дозволяє підключити функції та класи відповідної бібліотеки і використовувати їх у програмі.

```
#include <WiFi.h>
#include <PubSubClient.h>
```

Конфігурація WiFi та MQTT-клієнта

Для забезпечення підключення мікроконтролера ESP32 до мережі Wi-Fi та організації обміну даними з MQTT-сервером у програмному коді визначено кілька важливих змінних і об’єктів.

```
const char* ssid = "Wokwi-GUEST"; та const char* password = "";
```

Ці константні символьні вказівники (const char*) зберігають ідентифікатор бездротової мережі (SSID) та пароль, які використовуються для підключення ESP32 до Wi-Fi. У середовищі Wokwi зазвичай застосовується мережа **"Wokwi-GUEST"**, що не потребує введення пароля. У реальних умовах ці параметри

КПЗ_2026

необхідно змінити відповідно до налаштувань власної Wi-Fi мережі.

```
const char* mqtt_server = "test.mosquitto.org";
```

Ця константна змінна містить доменне ім’я або IP-адресу MQTT-брокера, з яким встановлюється з’єднання для обміну повідомленнями. У даному проєкті використовується публічний тестовий сервер **test.mosquitto.org**.

У практичних застосуваннях замість нього може використовуватися локальний MQTT-сервер або хмарний сервіс.

Порт MQTT

За стандартом протокол MQTT використовує TCP-порт **1883** для встановлення з’єднання між клієнтом і сервером. У програмному коді цей порт задається під час налаштування з’єднання з брокером (наприклад, у виклику функції client.setServer()), навіть якщо він не оголошений як окрема константа.

WiFiClient espClient

Цей об’єкт класу **WiFiClient** (з бібліотеки WiFi.h) використовується для створення та підтримки TCP-з’єднання. Саме на його основі реалізується мережеве підключення, необхідне для подальшої роботи MQTT-протоколу.

PubSubClient client(espClient)

Даний об’єкт класу **PubSubClient** (з бібліотеки PubSubClient.h) виконує функції MQTT-клієнта. Під час його створення передається об’єкт espClient, що забезпечує зв’язок між MQTT-клієнтом та мережевим TCP-з’єднанням.

Об’єкт **client** надає набір методів, необхідних для виконання основних операцій протоколу MQTT. Серед них - підключення до сервера, публікація повідомлень у визначених темах, підписка на потрібні теми, а також встановлення callback-функції для обробки отриманих повідомлень.

Правильне налаштування зазначених параметрів є важливим етапом для забезпечення стабільного підключення мікроконтролера ESP32 до мережі Wi-Fi та встановлення з’єднання з MQTT-сервером. Саме це забезпечує передачу даних від сенсорів і приймання команд керування в системі розумного дому.

Реалізація підключення до сервера та обробка з’єднання

У програмному коді реалізовано дві основні функції, які відповідають за

КПІ_2026

встановлення мережевого підключення та взаємодію з MQTT-сервером: **setup_wifi()** та **reconnect()**.

Детальний розгляд функції setup_wifi()

Ця функція виконує ініціалізацію Wi-Fi модуля мікроконтролера ESP32 та забезпечує підключення пристрою до бездротової мережі.

На початку функції встановлюється невелика затримка (delay(10)), необхідна для стабілізації роботи системи перед початком ініціалізації мережевих компонентів.

Serial.begin(115200)

Виконується ініціалізація послідовного порту зі швидкістю 115200 бод. Це дозволяє виводити діагностичні повідомлення для контролю процесу підключення та налагодження програми.

Serial.println();, Serial.print("Connecting to ");, Serial.println(ssid);

Ці команди виводять у послідовний порт повідомлення про початок процесу підключення до бездротової мережі та відображають її назву (SSID).

WiFi.mode(WIFI_STA)

Дана команда встановлює режим роботи Wi-Fi модуля як **станції (STA)**, тобто клієнтського пристрою, який підключається до точки доступу (маршрутизатора).

WiFi.begin(ssid, password)

Ініціює процес підключення ESP32 до бездротової мережі з використанням заданих параметрів SSID та пароля.

while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print("."); }

Цей цикл виконує очікування встановлення з’єднання з мережею Wi-Fi. Програма періодично перевіряє стан підключення за допомогою функції WiFi.status(). Поки з’єднання не встановлено, кожні 500 мс у серійний монітор виводиться символ крапки, що дозволяє спостерігати за процесом підключення.

Поки з’єднання не буде успішно встановлено (стан **WL_CONNECTED**), програма продовжує виконувати цикл очікування. У цей час кожні 500 мілісекунд

КПЗ-2026

виконується затримка, а в послідовний порт виводиться символ крапки, що сигналізує про те, що процес підключення до мережі триває.

randomSeed(micros())

Ця команда ініціалізує генератор випадкових чисел, використовуючи значення мікросекундного таймера. Отримане випадкове значення застосовується для формування унікального **Client ID** під час підключення до MQTT-сервера.

```
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
```

Після успішного підключення до бездротової мережі у серійний монітор виводиться повідомлення про встановлення з'єднання, а також IP-адреса, яку отримав мікроконтролер ESP32 від мережевого маршрутизатора. Це дозволяє переконатися, що пристрій коректно підключився до мережі.

```
Connecting to Wokwi-GUEST
..
WiFi connected
IP address:
10.10.0.2
```

Рисунок 3.4 - Повідомлення про успішне підключення

Детальний розбір функції reconnect()

Функція **reconnect()** призначена для встановлення або повторного відновлення з'єднання з MQTT-сервером у випадку його втрати. Вона викликається в основному циклі програми **loop()**, якщо MQTT-клієнт не має активного підключення (!client.connected()).

while (!client.connected()) { ... }

Цей цикл відповідає за повторні спроби підключення до MQTT-брокера. Виконання циклу триває доти, доки не буде встановлено успішне з'єднання з сервером.

КОПЗ 2026

Serial.print("Attempting MQTT connection...")

Дана команда виводить у серійний монітор повідомлення про початок спроби підключення до MQTT-сервера, що допомагає відстежувати процес роботи програми під час налагодження.

String clientId = "ESP32Client-" + String(random(0xffff), HEX)

Цей рядок формує унікальний ідентифікатор клієнта (**Client ID**) для MQTT-з'єднання. Унікальність ідентифікатора необхідна для коректної ідентифікації пристрою на сервері та запобігання конфліктам, якщо до брокера підключається кілька клієнтів одночасно.

Для забезпечення коректної ідентифікації пристрою на сервері, особливо у випадку одночасного підключення декількох клієнтів, використовується унікальний **Client ID**. Він формується шляхом поєднання префікса **"ESP32Client-"** та випадкового шістнадцяткового числа.

if (client.connect(clientId.c_str())) { ... }

Ця умова виконує спробу підключення до MQTT-сервера із використанням сформованого ідентифікатора клієнта. Метод client.connect() повертає значення **true**, якщо з'єднання встановлено успішно, або **false** у разі помилки.

Serial.println("Connected")

Якщо підключення виконано успішно, у серійний монітор виводиться повідомлення про встановлення з'єднання.

client.publish("/ThinkIOT/Publish", "Welcome")

Після підключення клієнт надсилає повідомлення у тему **/ThinkIOT/Publish**. Таке повідомлення може використовуватися для інформування інших підписників про появу нового пристрою в системі.

client.subscribe("/ThinkIOT/Subscribe")

Далі виконується підписка на тему **/ThinkIOT/Subscribe**. Це дозволяє ESP32 отримувати повідомлення, опубліковані у цій темі, наприклад команди керування або іншу службову інформацію.

КПЗ_2026

else { ... }

Якщо підключення до MQTT-сервера не вдалося, виконується блок обробки помилки.

Serial.print("failed, rc="); Serial.print(client.state()); Serial.println(" try again in 5 seconds");

У серійний монітор виводиться повідомлення про помилку підключення, код помилки (client.state()), а також інформація про повторну спробу підключення.

delay(5000);

Після невдалої спроби програма робить паузу тривалістю 5 секунд перед повторною спробою встановлення з'єднання.

Реалізовані механізми підключення до Wi-Fi та автоматичного відновлення з'єднання з MQTT-сервером є важливими для забезпечення стабільної та надійної роботи IoT-системи розумного дому.

Підключення сенсорів температури, вологості, газу та руху

Для організації збору даних про параметри внутрішнього середовища до мікроконтролера ESP32 було підключено кілька сенсорних модулів. Зокрема, у системі використовуються датчик температури та вологості **DHT22**, аналоговий газовий датчик (підключений до **піна 34**) та пасивний інфрачервоний датчик руху **PIR**.

Ініціалізація DHT22

У програмному коді для взаємодії з датчиком температури та вологості DHT22 використовується бібліотека DHTesp.h. На початку програми оголошується константа const int DHT_PIN = 15;, яка визначає цифровий пін мікроконтролера ESP32, до якого підключений сигнальний вихід датчика.

Після цього створюється об'єкт класу DHTesp з назвою dht. Цей об'єкт використовується для доступу до функцій бібліотеки та забезпечує взаємодію між програмою і датчиком.

КБПЗ_2026

Ініціалізація датчика виконується у функції setup() за допомогою виклику методу dht.setup(DHT_PIN, DHTesp::DHT22);. У цьому виклику передаються два параметри: номер контакту мікроконтролера, до якого підключений датчик (у даному випадку 15), а також тип сенсора — DHT22. Виконання цієї команди забезпечує налаштування необхідних параметрів для коректного обміну даними між ESP32 та датчиком.

Зчитування значення з аналогового газового сенсора MQ-2

Для підключення аналогового газового датчика у програмному коді оголошується константа const int GAS_SENSOR_PIN = 34;. Вона визначає аналоговий вхід мікроконтролера ESP32 (GPIO34), до якого під'єднано аналоговий вихід сенсора.

Зчитування даних із газового датчика виконується у функції loop() за допомогою виклику analogRead(GAS_SENSOR_PIN). Функція analogRead() є стандартною для платформи Arduino та використовується для отримання значення напруги з аналогового входу мікроконтролера.

Результатом виконання цієї функції є ціле число в діапазоні від 0 до 4095, що відповідає рівню напруги на вході. Отримане значення відображає відносний рівень концентрації газів або загального забруднення повітря. Для точного визначення концентрації конкретних газів необхідно виконати додаткове калібрування датчика.

Обробка цифрового сигналу з PIR-сенсора

Для роботи з PIR-сенсором у програмному коді оголошується константа const int PIR_SENSOR_PIN = 27;, яка визначає цифровий контакт мікроконтролера ESP32 (GPIO27), до якого підключено сигнальний вихід датчика руху.

Щоб налаштувати цей контакт як вхід, у функції setup() використовується команда pinMode(PIR_SENSOR_PIN, INPUT);. Вона конфігурує відповідний пін мікроконтролера для приймання сигналів від сенсора.

Зчитування стану датчика виконується у функції loop() за допомогою

КОПЗ_2026

виклику digitalRead(PIR_SENSOR_PIN). PIR-сенсор формує цифровий сигнал: значення HIGH передається у випадку виявлення руху, а LOW - коли рух відсутній. Функція digitalRead() повертає відповідне значення у вигляді 1 або 0.

Виведення отриманих даних у консоль

Для контролю роботи сенсорів і налагодження системи у програмному коді передбачено виведення отриманих даних у послідовний порт. Це реалізується за допомогою функцій Serial.print() та Serial.println(), які дозволяють відображати значення параметрів у серійному моніторі.

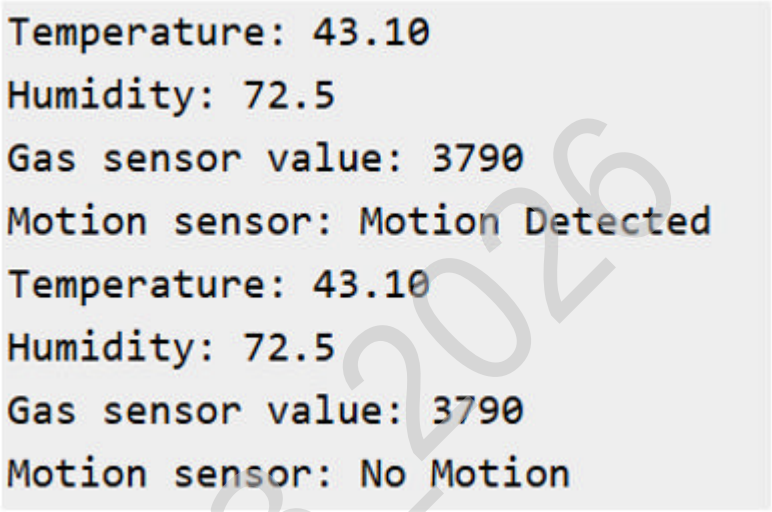


Рисунок 3.5 - Вивід отриманих даних у послідовний порт

У функції **loop()** після отримання даних із кожного сенсора виконується їх виведення у послідовний монітор. Зокрема, відображаються значення температури, вологості, аналоговий сигнал газового датчика, а також текстове повідомлення про стан датчика руху. Такий підхід дає можливість у режимі реального часу контролювати роботу сенсорів і перевіряти коректність отриманих даних.

Передача даних через MQTT та обробка повідомлень

Однією з основних функціональних можливостей розробленої IoT-системи є передавання даних, отриманих від сенсорів, на MQTT-сервер для подальшої обробки та відображення. Протокол MQTT було обрано через його простоту,

КПЗ_2026

ефективність та надійність при роботі в умовах обмежених обчислювальних ресурсів і нестабільного мережевого з'єднання, що характерно для багатьох IoT-рішень.

Формування MQTT-топіків для кожного сенсора

Для впорядкування даних, що передаються від мікроконтролера ESP32 до MQTT-брокера, використовуються MQTT-топіки. Топіки являють собою ієрархічні текстові рядки, які застосовуються сервером для фільтрації та розподілу повідомлень між підписаними клієнтами.

У межах розробленої системи для кожного типу сенсорних даних визначено окремий топік:

- /ThinkIOT/temp - використовується для передавання значень температури;
- /ThinkIOT/hum - призначений для публікації даних про вологість;
- /ThinkIOT/gas - застосовується для передавання аналогового значення з газового сенсора;
- /ThinkIOT/motion - використовується для передавання інформації про стан датчика руху (повідомлення *"Motion Detected"* або *"No Motion"*).

Розділення інформації за окремими топіками дає можливість клієнтським застосункам, наприклад Node-RED, підписуватися лише на необхідні типи даних. Це зменшує навантаження на мережу та спрощує подальшу обробку інформації. Використання префікса /ThinkIOT/ слугує загальним простором імен для цієї системи, що дозволяє уникнути конфліктів із повідомленнями інших пристроїв, які можуть працювати на тому ж MQTT-сервері.

Публікація даних у вигляді рядків

У головному циклі програми loop(), після отримання даних із сенсорів, виконується їх передавання на MQTT-сервер. Для цього використовується метод client.publish() об'єкта client, який належить до класу PubSubClient. Кожне значення публікується у відповідний MQTT-топік.

Значення температури (tempVal), отримане з датчика DHT22,

КБПЗ_2026

форматується з точністю до двох знаків після коми та перетворюється у текстовий рядок за допомогою функції String(tempVal, 2). Після цього сформоване значення передається у топік /ThinkIOT/temp.

Аналогічно обробляється значення вологості (humVal). Воно форматується до одного знака після коми за допомогою String(humVal, 1), після чого отриманий рядок публікується у топік /ThinkIOT/hum.

Аналогове значення, отримане з газового сенсора (gasValue), перетворюється у текстовий формат через функцію String(gasValue) і передається у топік /ThinkIOT/gas.

Стан датчика руху (motion), який має значення 0 або 1, використовується для формування текстового повідомлення. У випадку виявлення руху передається повідомлення "Motion Detected", а за його відсутності — "No Motion". Цей рядок публікується у топік /ThinkIOT/motion.

Метод client.publish() приймає два основні параметри: назву MQTT-топіка, до якого відправляється повідомлення, та корисне навантаження (payload), що містить передані дані. У даному випадку payload представлений у вигляді текстових рядків, що спрощує їх передачу мережею та подальшу обробку на стороні клієнта MQTT, наприклад у системі Node-RED.

Створення callback-функції для обробки вхідних повідомлень

Для обробки повідомлень, що надходять від MQTT-сервера до ESP32 після підписки на певні топіки, у програмному коді реалізовано callback-функцію callback(char* topic, byte* payload, unsigned int length). Ця функція автоматично викликається бібліотекою PubSubClient кожного разу, коли мікроконтролер отримує повідомлення з топіка, на який було оформлено підписку.

Функція callback приймає три основні параметри:

- topic - вказівник на рядок, який містить назву топіка, у якому було опубліковано отримане повідомлення;
- payload - вказівник на масив байтів, що містить корисне навантаження повідомлення;

КОПЗ 2026

- length - довжина масиву байтів payload.

У даній реалізації функція callback виконує просту обробку отриманих повідомлень - вона виводить у послідовний порт інформацію про отриманий топік та його вміст, попередньо перетворений у текстовий рядок. У більш складних IoT-системах ця функція може містити додаткову логіку, наприклад обробку команд керування або конфігураційних параметрів, які надходять від інших MQTT-клієнтів.

Для того щоб бібліотека PubSubClient могла використовувати цю функцію для обробки вхідних повідомлень, її необхідно зареєструвати у програмі. Це виконується у функції setup() за допомогою виклику методу:

```
client.setCallback(callback);
```

Автоматичне повторне з'єднання з сервером (reconnect)

Для забезпечення стабільної роботи системи у програмному коді реалізовано механізм автоматичного відновлення з'єднання з MQTT-сервером. Ця функціональність реалізована у функції reconnect(), яка була розглянута у попередньому підрозділі.

Функція reconnect() викликається у головному циклі loop() у випадку, коли перевірка стану з'єднання !client.connected() показує, що підключення до MQTT-сервера відсутнє. У такій ситуації програма ініціює повторну спробу встановлення з'єднання.

Механізм автоматичного перепідключення є важливим елементом для надійної роботи IoT-систем, оскільки мережеві з'єднання можуть бути нестабільними. У разі втрати зв'язку мікроконтролер ESP32 автоматично намагається відновити підключення до сервера, що дозволяє уникнути ручного втручання та забезпечує безперервну передачу даних від сенсорів.

Після повторного встановлення з'єднання мікроконтролер ESP32 знову виконує підписку на необхідні MQTT-топіки (наприклад, /ThinkIOT/Subscribe), що дозволяє продовжувати отримувати вхідні повідомлення від сервера.

Виклик функції client.loop() у головному циклі loop() є необхідним для підтримання активного MQTT-з'єднання. Ця функція забезпечує обробку

КОПЗ_2026

відправлених і отриманих повідомлень, а також викликає зареєстровану callback-функцію у випадку надходження нових повідомлень від MQTT-сервера.

Збереження даних у пам’яті ESP32

Для забезпечення збереження важливих даних після перезавантаження мікроконтролера ESP32 у розробленій IoT-системі було використано бібліотеку Preferences. Вона дозволяє реалізувати персистентне збереження невеликих обсягів інформації в енергонезалежній пам’яті мікроконтролера (NVS — Non-Volatile Storage).

Першим етапом стало підключення бібліотеки до проєкту за допомогою директиви `#include <Preferences.h>`.

Після цього було створено об’єкт класу Preferences:

```
Preferences prefs;
```

Для початку роботи з енергонезалежною пам’яттю необхідно відкрити так званий namespace (розділ пам’яті). Це виконується у функції setup() за допомогою виклику:

```
prefs.begin("sensors", false);
```

У цьому виклику рядок "sensors" визначає назву розділу, який використовується для збереження даних, пов’язаних із сенсорами. Другий параметр (false) означає, що пам’ять відкривається у режимі читання та запису для одного процесу. У простих проєктах з одним мікроконтролером цього режиму достатньо.

Використання окремих розділів пам’яті дозволяє логічно структурувати збережені дані та уникнути конфліктів імен ключів, особливо якщо в майбутньому система буде розширена і потребуватиме збереження додаткових параметрів.

Для запису останніх значень, отриманих від сенсорів, використовуються відповідні методи об’єкта prefs. Запис даних виконується безпосередньо перед їх публікацією на MQTT-сервер у функції loop()

```
prefs.putFloat("temp", tempVal);
prefs.putFloat("hum", humVal);
```

КПІЗ_2026

```
prefs.putInt("gas", gasValue);
```

Для збереження отриманих від сенсорів значень було використано методи putFloat() та putInt(). Метод putFloat() застосовується для запису значень температури (tempVal) та вологості (humVal) у вигляді чисел з плаваючою комою, тоді як метод putInt() використовується для збереження значення газового сенсора (gasValue) як цілого числа.

Кожен із цих методів приймає два параметри: ключ (текстовий ідентифікатор збереженого значення) та саме значення, яке необхідно записати у пам’ять. Наприклад, ключ "temp" використовується для збереження температури. Така структура «ключ–значення» є зручним способом організації даних у енергонезалежній пам’яті.

Для відновлення збережених значень під час кожного запуску мікроконтролера ESP32 у функції setup() використовуються відповідні методи get():

```
prefs.getFloat("temp", 0);
prefs.getFloat("hum", 0);
prefs.getInt("gas", 0);
```

Методи get() також приймають два аргументи: ключ даних, які необхідно отримати, та значення за замовчуванням. Це значення повертається у випадку, якщо за вказаним ключем ще не було збережено жодних даних (наприклад, під час першого запуску пристрою). Таким чином, якщо значення температури ще не було записано у пам’ять, змінна lastTemp отримає значення 0.

Для підтвердження того, що збережені дані коректно відновлюються після перезапуску системи, у функції setup() було додано виведення цих значень у послідовний монітор.

```
Serial.println("Last saved data:"); Serial.print("Temperature: ");
Serial.println(prefs.getFloat("temp", 0)); Serial.print("Humidity:
"); Serial.println(prefs.getFloat("hum", 0)); Serial.print("Gas: ");
Serial.println(prefs.getInt("gas", 0));
```

Хоча бібліотека Preferences є зручним засобом для збереження невеликих обсягів конфігураційних даних або останніх значень параметрів системи, вона має певні обмеження. Обсяг енергонезалежної пам’яті NVS, доступний для

КОПЗ_2026

використання цією бібліотекою, є відносно невеликим, а кількість циклів запису також обмежена.

Для збереження значно більших обсягів інформації, наприклад журналів подій або історичних даних сенсорів, можуть застосовуватися інші підходи. До таких рішень належить використання файлової системи SPIFFS, яка дозволяє працювати з файлами у флеш-пам'яті мікроконтролера ESP32, або застосування SD-карти, якщо апаратна конфігурація пристрою підтримує відповідний інтерфейс. Проте в межах цього проєкту збереження останніх значень сенсорів за допомогою бібліотеки Preferences є цілком достатнім.

Використання retained-повідомлень для збереження останнього стану

У протоколі MQTT передбачено важливий механізм, відомий як retained-повідомлення. Якщо клієнт публікує повідомлення з увімкненим прапорцем RETAIN, MQTT-сервер зберігає останнє отримане повідомлення для відповідної теми. Коли новий клієнт підписується на цю тему, сервер одразу надсилає йому збережене повідомлення.

Використання retained-повідомлень має суттєві переваги, особливо у системах моніторингу. Нові підписники після підключення миттєво отримують актуальний стан системи, не очікуючи наступної публікації даних від видавця (у даному випадку - мікроконтролера ESP32). Це особливо зручно для відображення поточних значень сенсорів у користувацькому інтерфейсі. Наприклад, коли користувач відкриває вебсторінку з інформаційною панеллю (дашбордом), він одразу бачить останні виміряні значення температури, вологості та інших параметрів, замість порожніх полів до моменту отримання нового повідомлення.

У поточній реалізації програмного коду публікація retained-повідомлень поки що не використовується. Проте для реалізації цієї функціональності можна скористатися розширеною версією методу client.publish(), яка приймає додатковий булевий параметр. Встановивши цей параметр у значення true, клієнт повідомляє серверу, що опубліковане повідомлення необхідно зберегти як retained для відповідної теми.

```
client.publish("/ThinkIOT/temp", temp.c_str(), true);
```

КПЗ – 2026

```
client.publish("/ThinkIOT/hum", hum.c_str(), true);
client.publish("/ThinkIOT/gas", gas.c_str(), true);
client.publish("/ThinkIOT/motion", motionState.c_str(), true);
```

Важливо ретельно обміркувати, для яких саме тем доцільно використовувати retained-повідомлення. У випадку з поточними значеннями датчиків (/ThinkIOT/temp, /ThinkIOT/hum, /ThinkIOT/gas, /ThinkIOT/motion) це, безумовно, є корисною практикою, оскільки нові підписники завжди будуть мати актуальну інформацію. Для інших типів повідомлень, таких як події або команди, використання retained-повідомлень може бути менш доцільним або навіть небажаним.

Коли MQTT-сервер отримує запит на підписку на певну тему, і для цієї теми існує retained-повідомлення, сервер негайно відправляє це повідомлення новому підписнику, а потім продовжує надсилати всі нові повідомлення, що публікуються в цій темі.

Інтеграція з Node-RED та створення інтерфейсу візуалізації

Для створення користувацького інтерфейсу та обробки даних, що надходять від IoT-системи через протокол MQTT, було використано платформу Node-RED. Це потужний і зручний інструмент, який дозволяє будувати потоки обробки даних та створювати IoT-інтерфейси за допомогою графічного редактора, що працює на основі взаємопов’язаних вузлів.

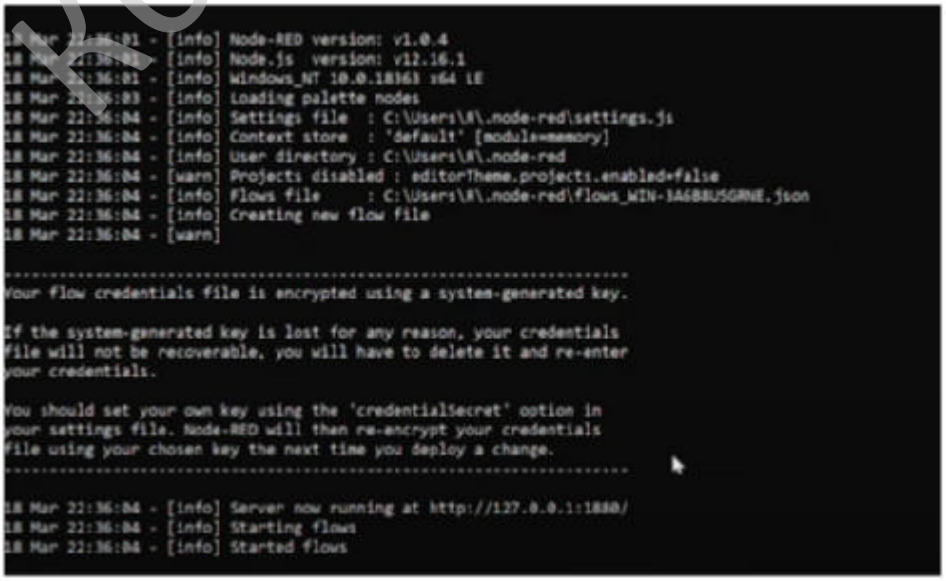


Рисунок 3.6- Встановлення Node-RED

КОПІЯ 2026

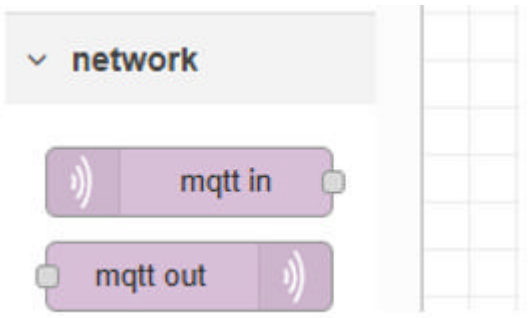


Рисунок 3.7- вузок mqtt in

Першим етапом роботи стало встановлення середовища Node-RED на локальний комп'ютер. Після запуску програми було відкрито її вебінтерфейс у браузері, який використовується для створення та налаштування потоків обробки даних.

Для підключення до MQTT-сервера, на який мікроконтролер ESP32 передає дані від сенсорів, до робочої області редактора було додано вузол mqtt in. Це було виконано шляхом перетягування відповідного елемента з палітри вузлів у лівій частині інтерфейсу.

Наступним кроком стало налаштування параметрів вузла. Після подвійного натискання на ньому відкривається вікно конфігурації (рис. 3.8), у якому задаються основні параметри підключення:

- Server – test.mosquitto.org:1883 - публічний MQTT-брокер, який використовується для приймання даних;
- Action – Subscribe to single topic - режим підписки на одну тему;
- Topic – /ThinkIOT/motion - MQTT-топік, на який підписується вузол;
- QoS – 0 - найнижчий рівень гарантії доставки повідомлень, що забезпечує мінімальну затримку;
- Output – auto-detect - автоматичне визначення формату отриманих даних.

Аналогічні налаштування було виконано і для інших MQTT-топіків, які використовуються у системі моніторингу:

КОПЗ_2026

- /ThinkIOT/temp - передавання значень температури;
- /ThinkIOT/hum - передавання значень вологості;
- /ThinkIOT/gas - передавання даних газового сенсора;
- /ThinkIOT/motion - передавання інформації про стан датчика руху.

Такий підхід дозволяє організувати приймання різних типів даних від IoT-пристрою та подальшу їх обробку й відображення у системі візуалізації.

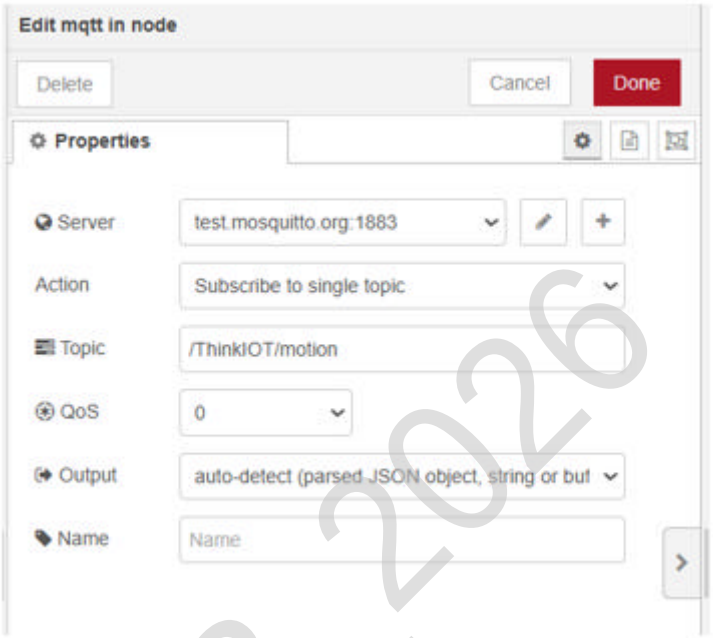


Рисунок 3.8 – Налаштування вузла mqtt in

До кожного вузла mqtt in було підключено відповідні елементи з категорії dashboard, які призначені для візуального відображення отриманих даних. Перед використанням цих елементів необхідно було встановити додатковий пакет node-red-dashboard.

Для цього у правому верхньому меню інтерфейсу Node-RED було обрано пункт Manage palette. На вкладці Install у поле пошуку введено слово dashboard, після чого знайдено та встановлено відповідний пакет, натиснувши кнопку Install.

Після завершення встановлення у палітрі вузлів з’явилася нова категорія dashboard, яка містить елементи для створення графічного інтерфейсу користувача. Серед доступних вузлів — gauge, text, chart, switch та інші, що дозволяють відображати дані сенсорів у різних візуальних формах.

КПІЗ_2026

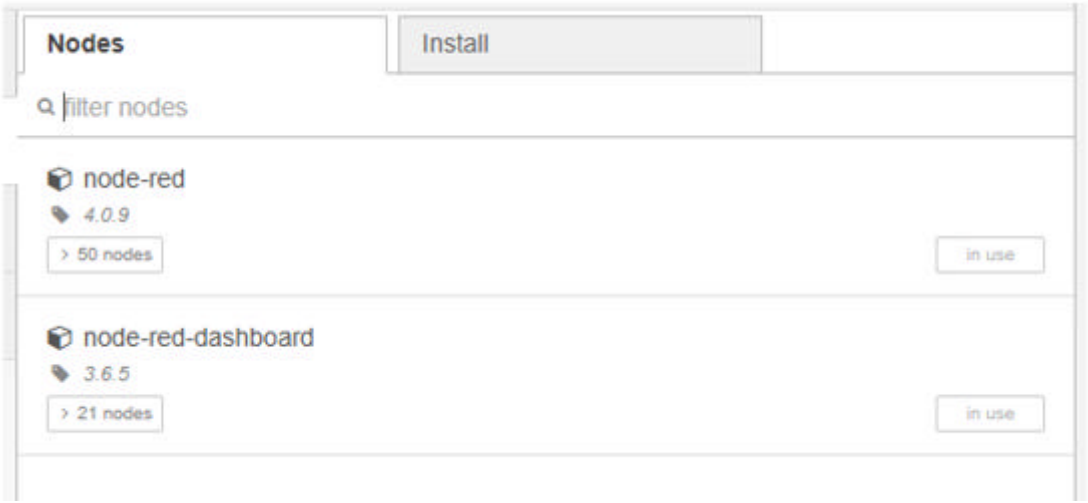


Рисунок 3.9 - Встановлений dashboard

Для відображення отриманих даних у системі було використано такі елементи візуалізації:

- вузли gauge - для відображення значень температури, вологості та показників газового сенсора;
- вузол text - для відображення стану датчика руху у вигляді текстового повідомлення (*Motion Detected* або *No Motion*).



Рисунок 3.10 - Налаштування вузла gauge для температури

КОПЗ_2026

Для вузла gauge, який використовується для відображення температури, було задано такі параметри:

- Label - *Temperature*;
- Units -°*C*;
- Range (min–max) - від 0 до 50;
- Group - група ThinkIOT (створена або вибрана серед наявних);
- Color gradient - кольорова шкала, де синій колір відповідає низьким значенням, зелений - нормальному діапазону, а червоний - підвищеним показникам.

Після завершення налаштування всіх вузлів було натиснуто кнопку Deploy, що активує створений потік обробки даних. Після цього інтерфейс візуалізації став доступним у браузері за адресою:

На інформаційній панелі Node-RED у режимі реального часу відображаються такі параметри:

- значення температури;
- рівень вологості;
- показники газового сенсора;
- стан датчика руху.

Таке представлення даних забезпечує зручний та наочний моніторинг параметрів внутрішнього середовища. Крім того, створений інтерфейс може бути легко розширений у майбутньому, наприклад шляхом додавання графіків змін показників, журналу подій або елементів керування різними виконавчими пристроями.

3.2 Розробка структурної схеми

Структурна схема відображає склад системи та зв’язки між основними блоками без деталізації внутрішніх процесів.

КПЗ_2026

Загальна структура системи

Система Smart Home складається з таких основних блоків:

– пристрої введення;

– мікропроцесорний контролер;

– мережевий рівень;

– сервер обробки та керування;

– сховище даних;

– пристрої виконання;

– інтерфейс користувача;

– блок живлення.

Пристрої введення

До блоку входять:

– датчики температури, вологості;

– датчики руху;

– датчики освітленості;

– газові сенсори;

– кнопки та перемикачі.

Функція:

- формування первинної інформації про стан середовища або дії користувача.

Контролер (мікропроцесорний пристрій)

Використовується ESP32

Функції:

– зчитування сигналів із датчиків;

– перетворення сигналів у цифрову форму;

– передача даних у мережу;

– прийом керуючих команд.

Це центральний елемент, який зв’язує “фізичний світ” і цифрову систему.

КПЗ_2026

Мережевий рівень

Включає:

– Wi-Fi / Ethernet;

– протоколи передачі даних (MQTT, HTTP).

Mosquitto MQTT Broker

Функція:

- забезпечення обміну даними між контролером і сервером.

Сервер обробки та керування

Реалізується на базі Node-RED

Функції:

– прийом даних;

– обробка та аналіз;

– реалізація логіки керування;

– формування команд.

Сховище даних

SQLite

Функція:

– зберігання даних, подій та налаштувань.

Пристрої виконання

– реле;

– контактори;

– освітлення;

– насос, вентилятор;

– електроклапани.

Функція:

- виконання фізичних дій за командами системи.

Інтерфейс користувача

веб-інтерфейс/мобільний додаток

КПЗ_2026

Функції:

– моніторинг стану;

– ручне керування;

– налаштування системи.

Блок живлення

Забезпечує:

– живлення контролера (5В / 3.3В);

– живлення датчиків;

– живлення виконавчих пристроїв.

Структурну схему наведено ри рисунку 3.11.

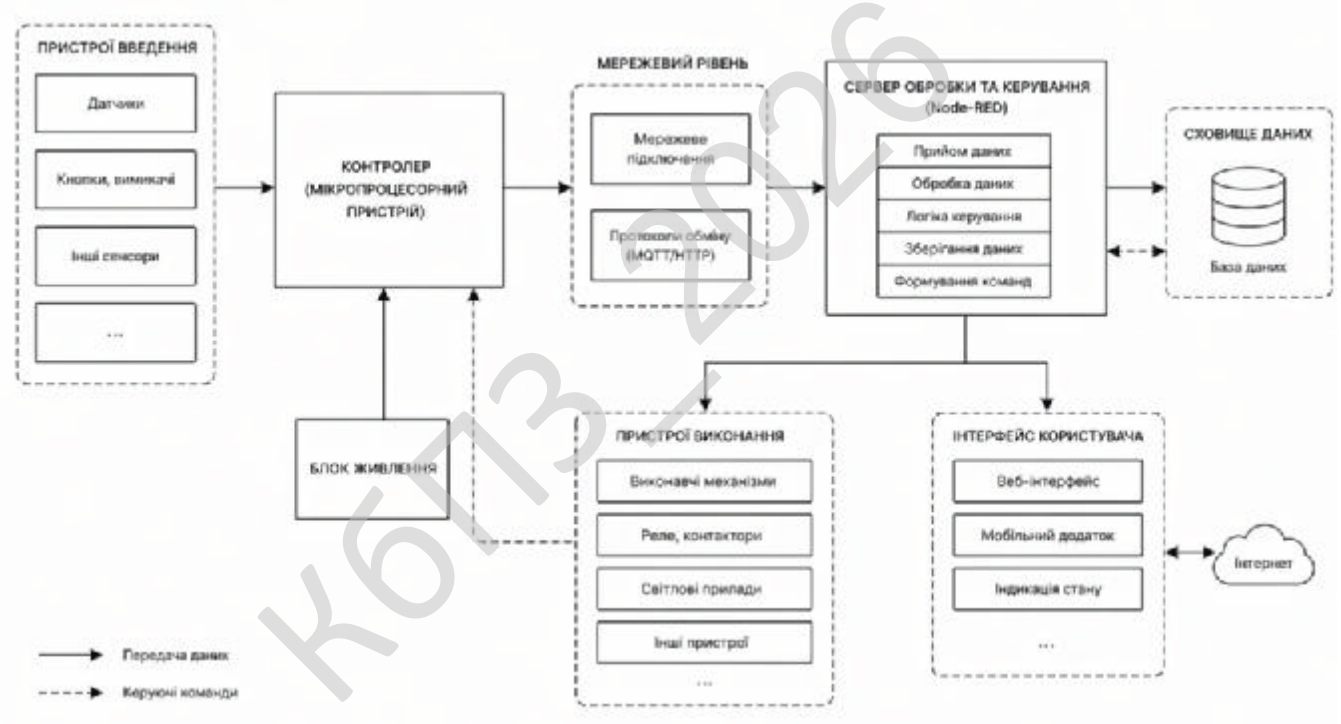


Рисунок 3.11 - Структурна схема системи

3.3 Розробка функціональної схеми

Функціональна схема мікропроцесорної IoT-системи Smart Home описує принцип роботи системи та послідовність обробки інформації під час моніторингу і керування. Робота системи починається зі збору даних від пристроїв введення, до яких належать різноманітні датчики та елементи ручного керування. Датчики

КбПЗ 2026

вимірюють параметри навколишнього середовища, такі як температура, вологість, освітленість, наявність руху або газу, а кнопки та вимикачі дозволяють користувачу вручну задавати команди.

Отримані сигнали надходять до контролера, який виконує їх зчитування, перетворення у цифрову форму та первинну обробку. На цьому етапі може здійснюватися фільтрація даних, усереднення або перевірка на коректність. Після обробки контролер формує інформаційні повідомлення та передає їх у мережу за допомогою відповідних протоколів обміну даними.

Передача даних здійснюється через мережевий рівень, який забезпечує зв’язок між контролером і сервером обробки та керування. Для обміну даними використовуються сучасні мережеві технології, зокрема бездротове підключення та протоколи типу MQTT або HTTP. У результаті інформація від контролера надходить до серверної частини системи.

На сервері обробки та керування виконується прийом даних, їх аналіз та подальша обробка. У середовищі Node-RED реалізуються алгоритми керування, які визначають поведінку системи залежно від отриманих значень параметрів. Дані можуть зберігатися у сховищі для подальшого аналізу, формування статистики або відображення користувачу.

На основі заданих правил і умов система формує керуючі команди. Наприклад, при зниженні температури нижче встановленого значення може бути сформована команда на увімкнення опалення, при виявленні руху - команда на увімкнення освітлення, а при виявленні газу - аварійне сповіщення.

Сформовані команди передаються назад до контролера через мережевий рівень. Контролер приймає ці команди, обробляє їх і формує відповідні сигнали керування для виконавчих пристроїв.

Виконавчі пристрої виконують безпосередній вплив на об’єкт керування. Це можуть бути реле, контактори, освітлювальні прилади, насоси, вентилятори або інші пристрої. Вони реалізують команди системи у вигляді фізичних дій, таких як увімкнення, вимкнення або регулювання режимів роботи.

КПЗ_2026

Паралельно з цим користувач взаємодіє із системою через інтерфейс користувача, який може бути реалізований у вигляді веб-інтерфейсу або мобільного додатку. Користувач має можливість переглядати поточні параметри, стан пристроїв, історію даних, а також вручну керувати системою або змінювати її налаштування.

Блок пристроїв введення призначений для отримання інформації про стан навколишнього середовища та об’єкта керування. До його складу входять різноманітні датчики, які вимірюють фізичні параметри, а також елементи ручного керування, такі як кнопки та вимикачі. Усі ці елементи формують первинні сигнали, що характеризують поточний стан системи, і передають їх до контролера для подальшої обробки.

Контролер, який є мікропроцесорним пристроєм, виконує функцію центрального вузла системи. Він забезпечує зчитування сигналів із пристроїв введення, їх перетворення у цифрову форму, первинну обробку та підготовку до передачі в мережу. Крім того, контролер приймає керуючі команди від серверної частини системи та формує сигнали для керування виконавчими пристроями.

Мережевий рівень призначений для забезпечення обміну даними між контролером та сервером обробки і керування. Він реалізує підключення до мережі та використовує відповідні протоколи передачі даних, що дозволяє організувати надійну та ефективну комунікацію між усіма компонентами системи.

Сервер обробки та керування виконує функції прийому, обробки та аналізу даних, що надходять від контролера. У цьому блоці реалізуються алгоритми керування, які визначають поведінку системи залежно від отриманої інформації. Сервер також формує керуючі команди та забезпечує їх передачу до виконавчих пристроїв через контролер.

Сховище даних призначене для зберігання інформації про роботу системи, включаючи результати вимірювань, події та налаштування. Воно забезпечує можливість подальшого аналізу даних, формування статистики та відображення історичної інформації користувачу.

КПЗ-2026

Блок виконавчих пристроїв забезпечує реалізацію керуючих впливів на об’єкт керування. До нього входять різні пристрої, такі як реле, контактори, освітлювальні прилади, насоси та інші механізми. Вони виконують команди системи та здійснюють фізичні дії, змінюючи стан середовища або обладнання.

Інтерфейс користувача призначений для забезпечення взаємодії людини з системою. Він дозволяє користувачу отримувати інформацію про стан системи, переглядати дані, змінювати налаштування та здійснювати ручне керування пристроями. Інтерфейс може бути реалізований у вигляді веб-додатку або мобільного застосунку.

Блок живлення забезпечує електроживлення всіх компонентів системи. Він формує необхідні рівні напруги для роботи контролера, датчиків, мережевих модулів та виконавчих пристроїв, забезпечуючи стабільну та надійну роботу всієї системи в цілому.

Таким чином, функціональна схема відображає повний цикл роботи системи - від отримання інформації до виконання керуючих дій і забезпечення зворотного зв’язку, що дозволяє реалізувати ефективне та автоматизоване керування об’єктами Smart Home.

Фнкціональна схема наведена на рисунку 3.12.

КбПЗ_2026

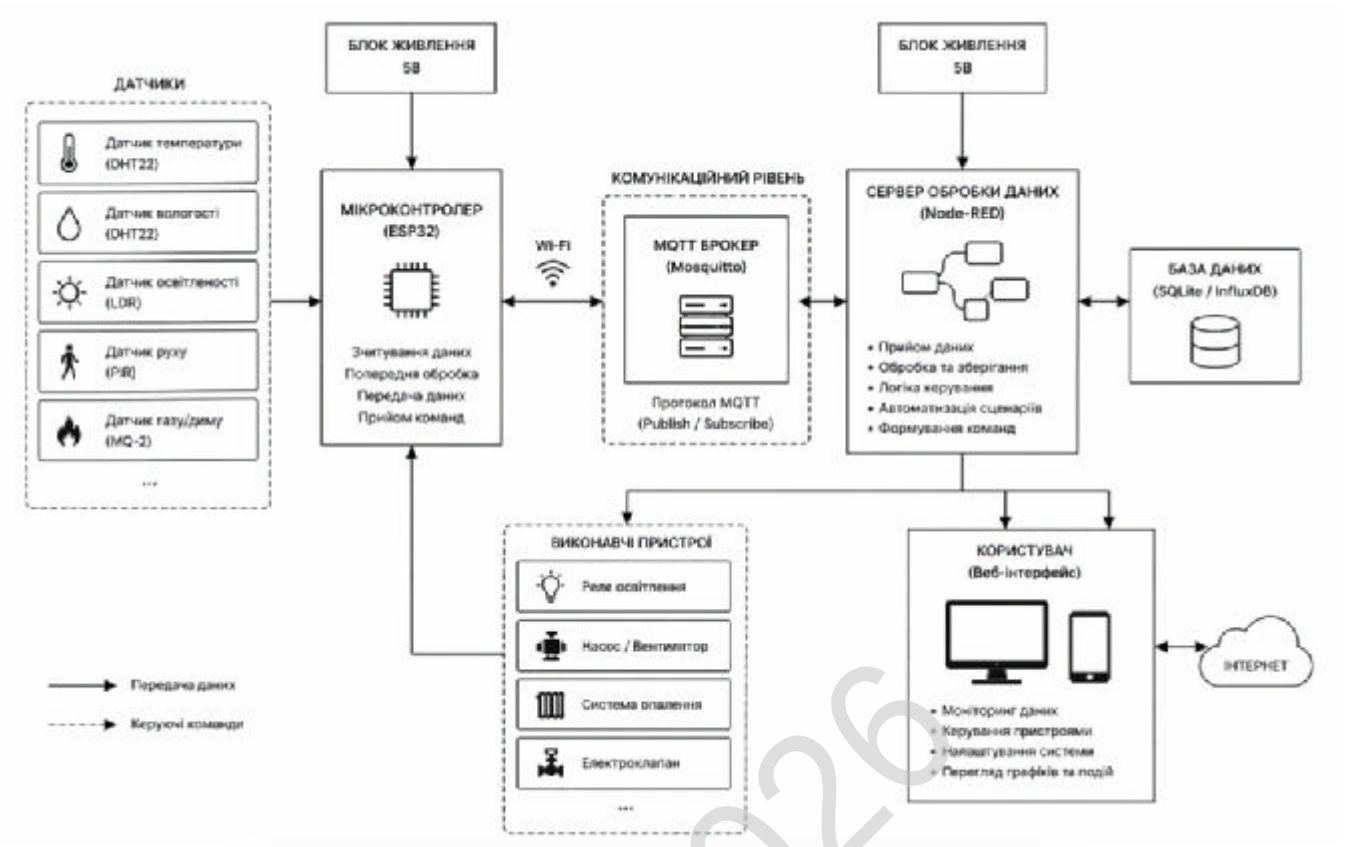


Рисунок 3.12 - Функціональна схема системи керування

3.4 Розробка діаграми процесів

Діаграма процесів призначена для відображення послідовності виконання операцій у системі Smart Home, а також логіки обробки даних і прийняття рішень. Вона показує, як саме відбувається перетворення вхідної інформації у керуючі дії, які реалізуються виконавчими пристроями.

Дана діаграма описує роботу системи як набір взаємопов'язаних процесів, що виконуються послідовно або циклічно, забезпечуючи безперервний моніторинг і керування.

Основним призначенням діаграми процесів є формалізація алгоритму роботи системи. Вона дозволяє чітко визначити, які операції виконуються на кожному етапі, у якій послідовності вони відбуваються та які дані передаються між окремими процесами.

Діаграма процесів використовується для аналізу функціонування системи,

КПЗ_2026

оптимізації її роботи, виявлення можливих помилок або вузьких місць, а також для спрощення розробки програмного забезпечення, зокрема у середовищі Node-RED.

Робота системи починається з процесу збору даних, під час якого датчики вимірюють параметри навколишнього середовища. Отримані значення надходять до контролера, де відбувається їх зчитування та первинна обробка. На цьому етапі дані можуть бути відфільтровані або перевірені на коректність.

Після цього запускається процес передачі даних, у межах якого контролер формує повідомлення та надсилає їх у мережу. Передача здійснюється за допомогою мережевих протоколів, що забезпечують надійний обмін інформацією між пристроями та сервером.

Далі виконується процес обробки даних на сервері. Отримана інформація аналізується, може зберігатися у базі даних і використовується для подальшого прийняття рішень. У цьому процесі реалізуються алгоритми керування, які визначають необхідні дії залежно від поточного стану системи.

Наступним етапом є процес прийняття рішення, у межах якого система визначає, чи потрібно виконувати певні керуючі дії. Якщо умови, задані алгоритмом, виконуються, формується відповідна команда керування.

Після цього відбувається процес передачі керуючих команд від сервера до контролера. Контролер приймає ці команди, обробляє їх і готує сигнали для виконавчих пристроїв.

На завершальному етапі виконується процес виконання, під час якого виконавчі пристрої реалізують отримані команди у вигляді фізичних дій. Це може бути увімкнення або вимкнення обладнання, зміна режиму роботи або інші дії.

Після виконання команда система знову переходить до процесу збору даних, що забезпечує безперервний цикл роботи та наявність зворотного зв’язку.

Висновок

Таким чином, діаграма процесів відображає логіку функціонування IoT-системи Smart Home у вигляді послідовності взаємопов’язаних процесів. Вона

КОПЗ-2026

дозволяє зрозуміти, як відбувається обробка інформації та керування системою, і є важливим елементом при розробці та аналізі програмного забезпечення, зокрема у середовищі Node-RED.

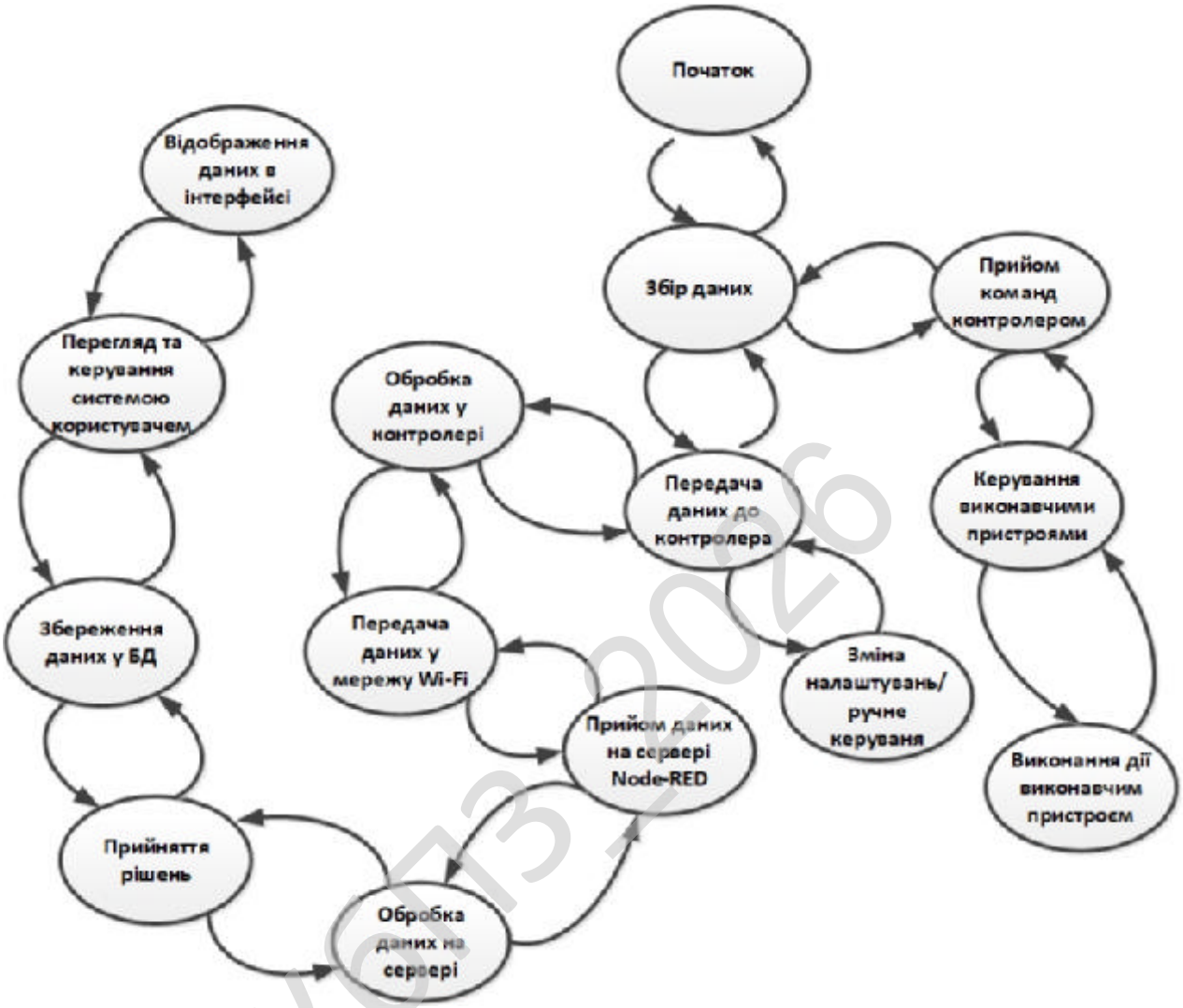


Рисунок 3.13 – Діаграма взаємодії процесів

КПЗ 2026

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

Розробка блок-схеми алгоритму функціонування мікропроцесорної IoT-системи Smart Home дозволяє формалізувати логіку її роботи та визначити послідовність виконання основних операцій. Алгоритм функціонування системи базується на безперервному циклі обробки інформації, що включає етапи збору даних, їх передачі, аналізу, прийняття рішень та виконання керуючих впливів. Такий підхід забезпечує адаптивність системи до змін зовнішніх умов і дозволяє реалізувати автоматизоване керування побутовими процесами.

На початковому етапі алгоритму виконується ініціалізація системи, під час якої відбувається налаштування мікроконтролера, перевірка підключених датчиків і виконавчих пристроїв, встановлення мережевого з’єднання та підключення до сервера. У цей момент визначаються початкові параметри роботи системи, завантажуються налаштування та граничні значення для контрольованих величин. Після успішної ініціалізації система переходить у режим основного циклу роботи.

У процесі виконання основного циклу алгоритму відбувається періодичне зчитування даних із датчиків. Контролер отримує значення фізичних параметрів, таких як температура, вологість, освітленість або наявність руху, після чого здійснює їх попередню обробку. На цьому етапі можуть виконуватися операції фільтрації шумів, усереднення показників або перевірка коректності отриманих значень. Це дозволяє підвищити достовірність даних і забезпечити стабільну роботу системи.

Після обробки дані передаються у мережу з використанням відповідних протоколів обміну інформацією. Контролер формує повідомлення у визначеному форматі та надсилає їх на сервер обробки та керування. Передача даних

КПЗ-2026

здійснюється з урахуванням періодичності опитування датчиків або при виникненні певних подій, що дозволяє оптимізувати навантаження на мережу та підвищити ефективність роботи системи.

На сервері відбувається прийом і подальша обробка отриманої інформації. Алгоритми, реалізовані у програмному середовищі, аналізують дані, порівнюють їх із заданими пороговими значеннями та визначають поточний стан системи. За необхідності інформація зберігається у базі даних для подальшого аналізу, формування статистики або відображення користувачу. На цьому ж етапі реалізується логіка керування, яка визначає необхідність виконання певних дій.

У разі виконання заданих умов алгоритм формує керуючі команди для виконавчих пристроїв. Ці команди передаються через мережу до контролера, який здійснює їх прийом і обробку. Контролер перетворює отримані команди у відповідні електричні сигнали, що подаються на виконавчі пристрої. Це можуть бути сигнали на увімкнення або вимкнення реле, запуск насосів, керування освітленням або іншими пристроями.

Виконавчі пристрої реалізують отримані команди у вигляді фізичних дій, що безпосередньо впливають на об’єкт керування. Після виконання дій система знову переходить до етапу збору даних, що забезпечує безперервний цикл роботи. Такий підхід дозволяє оперативно реагувати на зміни параметрів середовища та підтримувати задані умови функціонування.

КПЗ – 2026

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

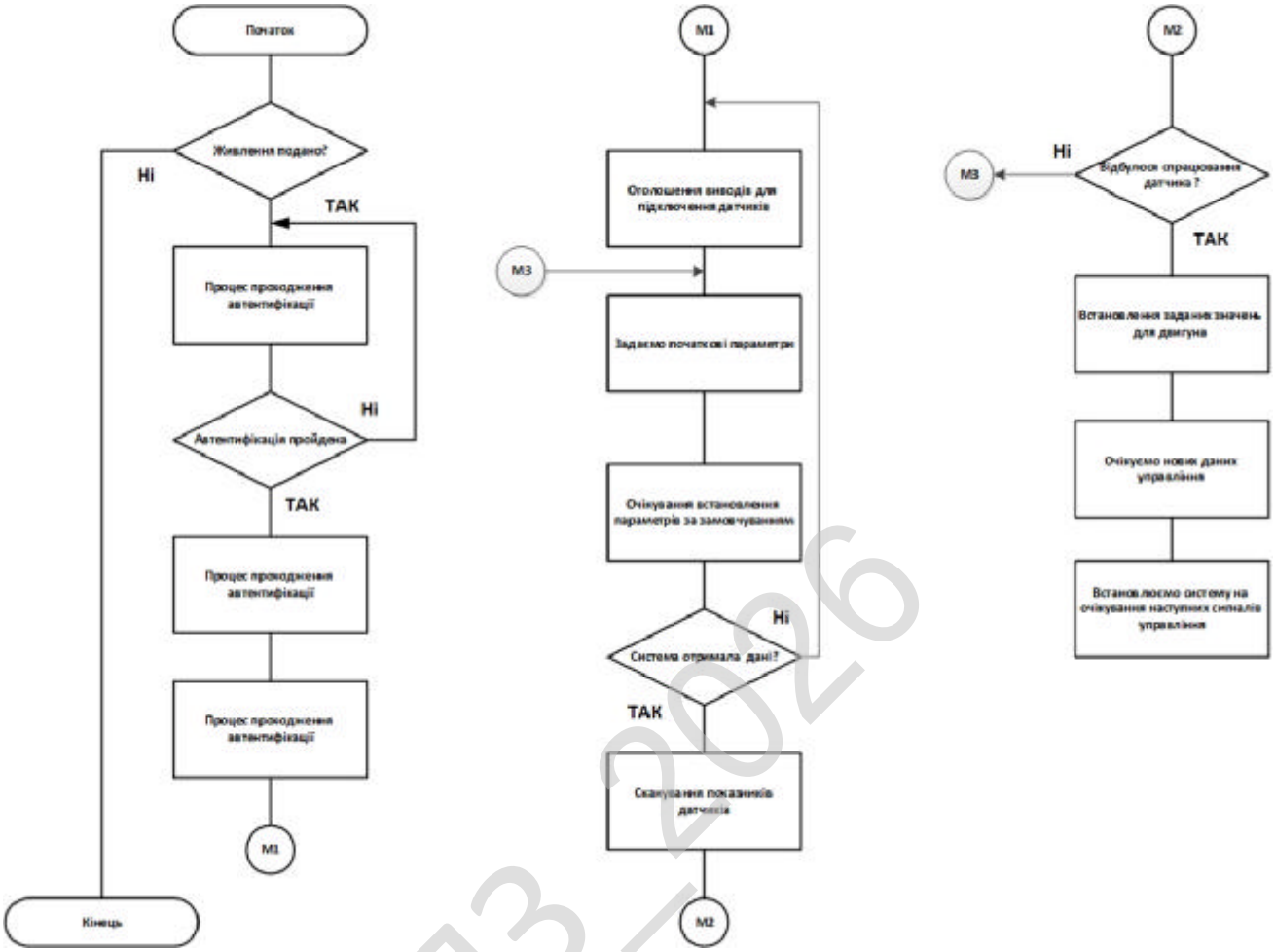


Рисунок 4.1 - Алгоритм функціонування системи Smart Home на базі Node-RED технології

КПЗ 2026



Рисунок 4.2 – Блок-схема алгоритму керування

КПЗ – 2026

Висновки до розділу

Важливою складовою алгоритму є організація зворотного зв'язку, яка дозволяє системі контролювати результати виконаних дій. Після кожного циклу роботи система отримує нові дані від датчиків, що дає можливість оцінити ефективність керуючих впливів і, за необхідності, скоригувати подальші дії. Це забезпечує стабільність і надійність роботи системи.

Окрім автоматичного режиму, алгоритм передбачає можливість взаємодії користувача із системою. Через інтерфейс користувач може отримувати інформацію про стан системи, змінювати налаштування або вручну керувати

виконавчими пристроями. У цьому випадку команди користувача обробляються сервером і передаються до контролера аналогічно автоматичним керуючим сигналам.

Розробка блок-схем та алгоритмів функціонування мікропроцесорної IoT-системи Smart Home є важливим етапом проєктування, оскільки дозволяє формалізувати всі процеси, що відбуваються в системі, та забезпечити їх логічну узгодженість. Алгоритм визначає порядок виконання операцій, умови переходу між етапами та реакцію системи на зміну вхідних параметрів. Завдяки цьому досягається передбачуваність поведінки системи, можливість її оптимізації та спрощення процесу програмної реалізації.

На етапі ініціалізації система виконує налаштування всіх апаратних і програмних компонентів. Мікроконтролер здійснює запуск внутрішніх модулів, ініціалізує інтерфейси введення-виведення, встановлює параметри підключення до мережі та перевіряє доступність датчиків і виконавчих пристроїв. Одночасно виконується встановлення з'єднання із сервером обробки даних, а також синхронізація часу, що є важливою умовою коректного ведення журналів подій. У разі виявлення помилок на цьому етапі алгоритм може передбачати повторні спроби ініціалізації або перехід у безпечний режим роботи.

Після завершення ініціалізації система переходить у режим безперервного функціонування, який реалізується у вигляді циклічного алгоритму. У кожному циклі виконується зчитування даних із датчиків, що дозволяє отримувати актуальну інформацію про стан середовища. Частота опитування може бути сталою або змінюватися залежно від умов роботи системи, що дає можливість оптимізувати використання ресурсів і зменшити навантаження на мережу.

Отримані дані проходять етап попередньої обробки, який включає фільтрацію випадкових завад, усереднення значень та перевірку на допустимість. Це дозволяє уникнути помилкових спрацьовувань системи та підвищити точність прийняття рішень. У деяких випадках алгоритм може передбачати нормалізацію даних або приведення їх до єдиного формату для подальшої передачі.

КОПЗ_2026

Після цього відбувається формування повідомлень і передача даних до серверної частини системи. Для цього використовуються мережеві протоколи, які забезпечують ефективний і надійний обмін інформацією. Алгоритм може передбачати як періодичну передачу даних, так і передачу за подією, наприклад при перевищенні певного порогового значення. Це дозволяє підвищити ефективність роботи системи та зменшити обсяг переданих даних.

На сервері обробки виконується більш складний аналіз отриманої інформації. Дані можуть об’єднуватися з іншими джерелами, зберігатися у базі даних та використовуватися для формування історії змін параметрів. Алгоритми, реалізовані на сервері, дозволяють враховувати різні фактори, такі як час доби, попередні стани системи або зовнішні умови, що значно розширює можливості керування.

Особливе місце в алгоритмі займає процес прийняття рішень. На цьому етапі система порівнює отримані дані з заданими умовами та визначає необхідність виконання певних дій. Умови можуть бути простими або складними, включати декілька параметрів і враховувати їх взаємозв’язок. Завдяки цьому забезпечується гнучкість системи та можливість реалізації різних сценаріїв автоматизації.

У разі необхідності формуються керуючі команди, які передаються до контролера. Передача здійснюється через мережу, а самі команди можуть містити інформацію про тип дії, її параметри та пріоритет виконання. Контролер, отримавши команду, виконує її декодування та перетворення у сигнали керування для виконавчих пристроїв.

Виконавчі пристрої здійснюють фізичну реалізацію команд, змінюючи стан об’єкта керування. Це може бути увімкнення або вимкнення обладнання, зміна режимів роботи або виконання інших дій. Алгоритм може передбачати контроль виконання команд, що дозволяє перевірити, чи була дія виконана успішно.

Після виконання дій система знову переходить до етапу збору даних, що

КПЗ – 2026

формує замкнений цикл роботи. Наявність зворотного зв’язку дозволяє системі адаптуватися до змін умов і коригувати свою поведінку в реальному часі. У разі виявлення відхилень від заданих параметрів алгоритм може передбачати повторне виконання команд або зміну стратегії керування.

Додатково алгоритм може включати механізми обробки помилок і аварійних ситуацій. У разі втрати зв’язку із сервером система може переходити у автономний режим роботи з використанням локально заданих правил. При виявленні несправності датчика або виконавчого пристрою можуть формуватися повідомлення для користувача або виконуватися дії для запобігання аварійним ситуаціям.

Важливою складовою алгоритму є взаємодія з користувачем. Користувач має можливість отримувати інформацію про стан системи, переглядати історію даних, змінювати налаштування та здійснювати ручне керування. У цьому випадку команди користувача інтегруються в загальний алгоритм і виконуються на рівні з автоматично сформованими командами.

Також алгоритм може передбачати механізми оптимізації роботи системи, зокрема регулювання частоти опитування датчиків, вибір оптимального режиму передачі даних або адаптацію логіки керування залежно від умов експлуатації. Це дозволяє підвищити енергоефективність системи та зменшити навантаження на апаратні ресурси.

Таким чином, розроблена блок-схема алгоритму функціонування системи відображає повний цикл обробки інформації та керування, забезпечує узгоджену роботу всіх компонентів і створює основу для реалізації ефективної та надійної IoT-системи Smart Home. Реалізація такого алгоритму дозволяє забезпечити автоматизацію процесів, підвищити комфорт користувача та забезпечити стабільну роботу системи в різних умовах експлуатації.

КбПЗ_2026

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Тестування підключення та передачі даних через WiFi

На цьому етапі було виконано перевірку підключення мікроконтролера ESP32 до віртуальної Wi-Fi мережі середовища Wokwi.

Перевірка встановлення Wi-Fi з'єднання

Після запуску симуляції у Wokwi здійснювалося спостереження за вікном серійного монітора, що дозволило переконатися у успішному підключенні до віртуальної бездротової мережі та отриманні пристроєм віртуальної IP-адреси.

На рисунку 5.1 наведено скриншот серійного монітора Wokwi, у якому відображаються відповідні повідомлення, зокрема «WiFi connected» та «IP address: ...», що підтверджують успішне встановлення мережевого з'єднання.

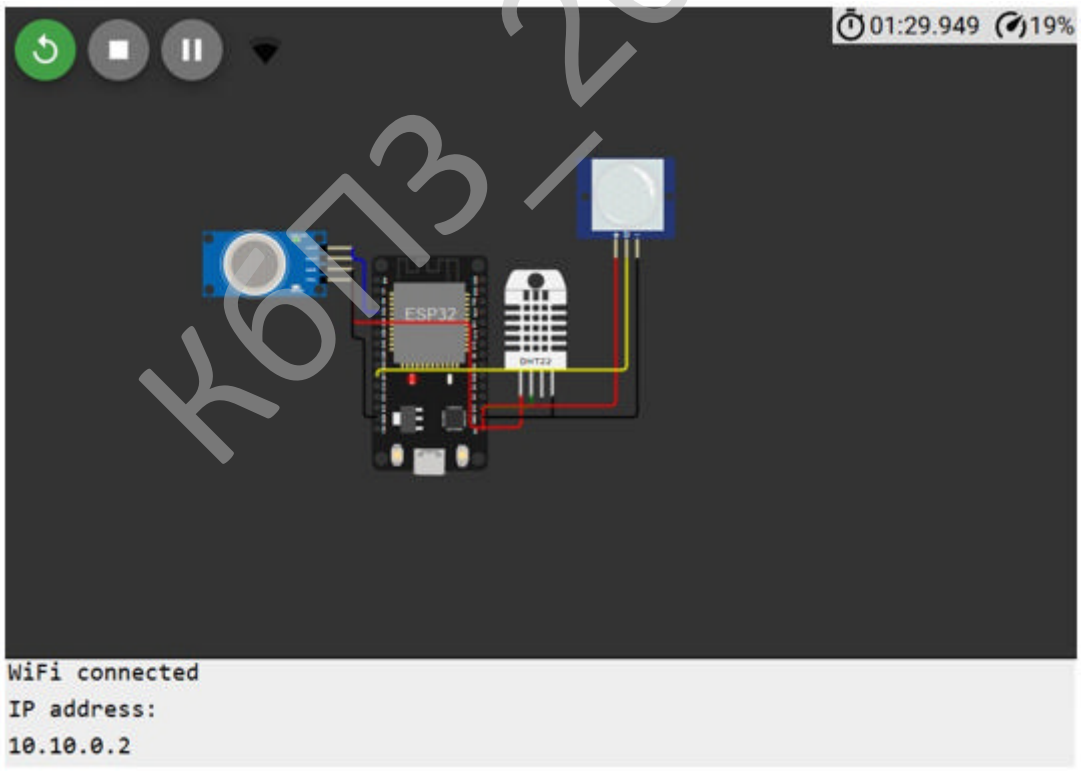


Рисунок 5.1-Успішне встановлення з'єднання

КбПЗ – 2026

Тестування стабільності віртуального Wi-Fi з’єднання

Протягом певного часу роботи симуляції проводилося спостереження за повідомленнями у серійному моніторі. Метою було переконатися у стабільності мережевого з’єднання та відсутності повідомлень про розрив підключення або повторні спроби під’єднання до мережі.

Тестування підключення та обміну даними через MQTT

На цьому етапі було перевірено процес імітації підключення мікроконтролера ESP32 до MQTT-сервера, а також коректність передавання даних від віртуальних сенсорів.

Перевірка підключення до MQTT-сервера

У середовищі Node-RED було виконано візуальний контроль стану вузлів mqtt in, які підписані на відповідні MQTT-топіки (/ThinkIOT/temp, /ThinkIOT/hum, /ThinkIOT/gas, /ThinkIOT/motion).

На рисунку 5.2 наведено скриншот потоку Node-RED, де вузли mqtt in відображають статус “connected”, що підтверджує успішне встановлення з’єднання з MQTT-сервером.

КбПЗ_2026

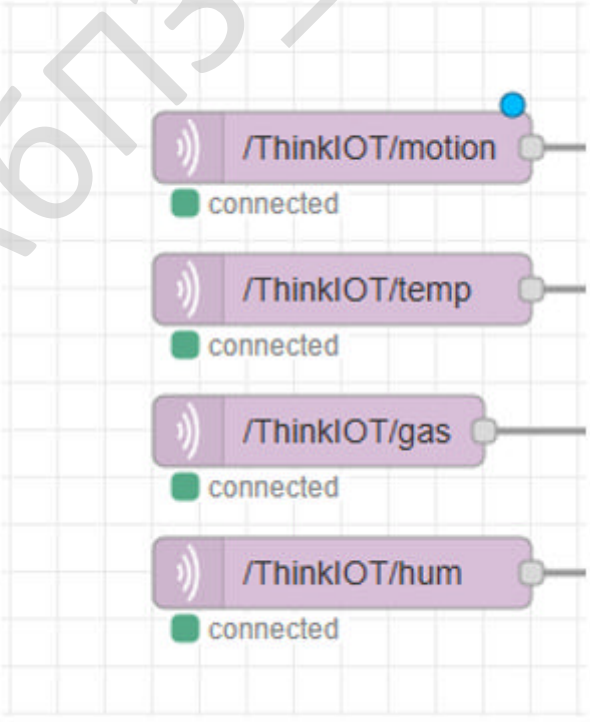


Рисунок 5.2 - Вузли mqtt in у стані "connected"

Перевірка публікації даних від віртуального ESP32

Для контролю правильності передавання повідомлень було використано вузол debug у середовищі Node-RED. За його допомогою перевірявся вміст MQTT-повідомлень, які надходили на відповідні топіки після зчитування даних з віртуальних сенсорів у симуляторі Wokwi.

На рисунку 5.3 наведено скриншот консолі налагодження Node-RED, де відображено приклади отриманих повідомлень, зокрема повідомлення про стан датчика руху.



Рисунок 5.3 - Отримані повідомлення про стан руху

Тестування коректності передачі віртуальних значень

На цьому етапі проводилося порівняння значень, які генерувалися віртуальними сенсорами у середовищі Wokwi, із показниками, що відображалися у Node-RED на відповідних елементах дашборду. Метою було підтвердити правильність передавання даних та їх відповідність між системами.

КОПЗ_2026

На рисунку 5.4 наведено одночасний скриншот середовища Wokwi та інформаційної панелі Node-RED, де показано порівнювані значення параметрів.

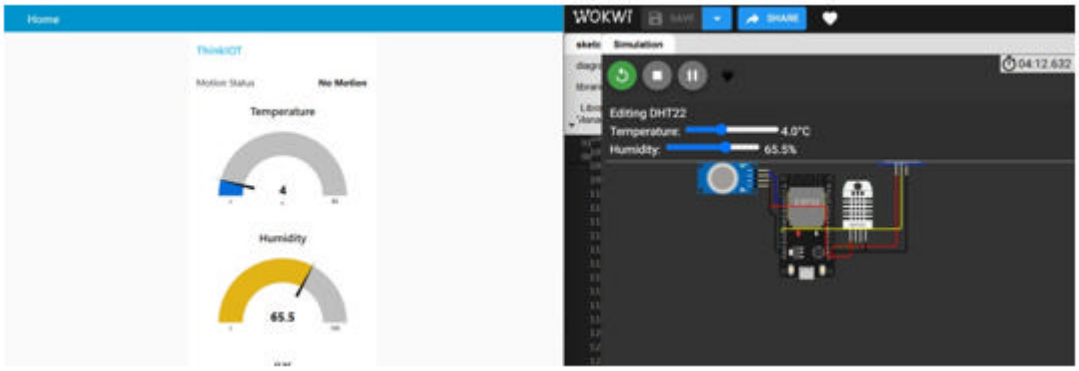


Рисунок 5.4-Порівняння даних з сенсорів та дашборду

Тестування функціональності віртуальних сенсорів

На цьому етапі було перевірено коректність імітації роботи кожного з віртуальних сенсорів у середовищі Wokwi.

Тестування віртуального датчика температури та вологості (DHT22)

Для перевірки роботи сенсора змінювалися параметри temperature та humidity у властивостях віртуального компонента DHT22. Після зміни цих значень спостерігалася відповідна зміна показників на віджетах Temperature та Humidity, які відображаються на дашборді Node-RED.

На рисунку 5.5 наведено приклад зміни значень віджетів на інформаційній панелі після коригування параметрів віртуального сенсора у середовищі Wokwi.

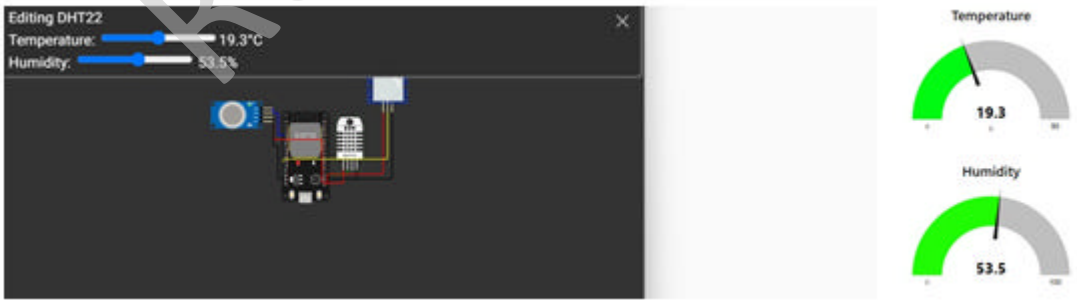


Рисунок 5.5 - Зміна даних відносно датчиків температури та вологості

Тестування віртуального газового сенсора

Для перевірки роботи газового датчика змінювався параметр, який імітує

КОПЗ_2026

рівень газу у властивостях відповідного віртуального компонента в середовищі Wokwi. Після зміни цього параметра спостерігалася відповідна зміна значення на віджеті Gas на дашборді Node-RED.

Тестування віртуального PIR-сенсора

Під час тестування змінювався стан віртуального PIR-сенсора у Wokwi. У результаті на інформаційній панелі Node-RED фіксувалася відповідна зміна стану віджета Motion Status.

На рисунку 5.6 показано зміну значення віджета Motion Status після зміни стану віртуального PIR-сенсора у середовищі Wokwi.



Рисунок 5.6 - Зміна стан PIR-сенсора

КбПЗ_2026

6 ОСНОВНІ ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено прототип IoT-системи для розумного дому на основі мікроконтролера ESP32, призначений для збору та передавання екологічних параметрів внутрішнього середовища. У системі використано датчики температури, вологості, газу та руху.

Передавання даних здійснюється за допомогою протоколу MQTT, який забезпечує ефективний і надійний обмін інформацією з сервером. На основі отриманих повідомлень було створено візуальний інтерфейс у середовищі Node-RED, що дозволяє користувачеві в режимі реального часу відстежувати значення основних параметрів.

Як основне середовище моделювання використано платформу Wokwi, що дало змогу швидко створити електронну схему, протестувати програмний код та перевірити логіку взаємодії всіх елементів системи.

Крім того, у системі реалізовано механізм збереження останніх отриманих даних у пам’яті ESP32, що підвищує надійність роботи у випадку перебоїв живлення або втрати мережевого з’єднання.

Отримані результати демонструють ефективність використання віртуальних середовищ для розроблення та тестування IoT-рішень, а також підтверджують доцільність застосування архітектури на основі ESP32, MQTT та Node-RED для створення прототипів систем розумного дому.

КБПЗ–2026

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Koval V., Adamiv O., Proc. of the Third IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2005). – Sofia (Bulgaria). – 2005. – P. 120- 124.

2. Yasin, J.N., Mohamed, S.A.S., Haghbayan, MH. et al. Low-cost ultrasonic based object detection and collision avoidance method for autonomous robots. Int. j. inf. tecnol. 13, 97–107 (2021). https://doi.org/10.1007/s41870-020-00513-w.

3. Sabry F. Self Driving Car: Solving Full Self-driving Need Solving Real-world artificial Intelligence. One Billion Knowledgeable. 2022 . 293 p.

4. Shapiro D. G. Three Anecdotes from the DARPA Autonomous Land Vehicle Project. AI Magazine, 2008. 29(2), 40. https://doi.org/10.1609/aimag.v29i2.2108.

5. Young R. Critical Analysis of Prototype Autonomous Vehicle Crash Rates Six Scientific Studies from 2015-2018 SAE International. 2021. 252 p.

6. Рудик А. В. Наукові основи та принципи побудови приладової системи вимірювання прискорення мобільного робота : дис. ... д-ра техн. наук : 05.11.01– Прилади та методи вимірювання механічних величин. Київ, 2018. – 460 с.

7. Поліщук М. М., Ткач М.М. Робототехнічні системи: проектування і моделювання: навч. посіб. Київ: КПІ ім. Ігоря Сікорського, 2021. 112 с.

8. Мигаль В. Д. Інтелектуальні системи в технічній експлуатації автомобілів: монографія. Х.: Майдан, 2018. 262 с.

9. Staron M. Automotive Software Architectures: An Introduction. Second Edition Springer International Publishing A&G. 2022. 274 p.

10. Vargas J. et al. An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions. Sensors. 2021; 21(16):5397.

11. Wevolver 2020 Autonomous Vehicle Technology Report. URL:https://www.coursehero.com/file/62601304/Wevolver2020AutonomousVehicechnologyReportpdf.

КПІ – 2026

12. React.js [Електронний ресурс] – Режим доступу до ресурсу: https://https://uk.reactjs.org/

13. AngularJS MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm.

14. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/

15. A Look Inside ADAS Modules. URL: https://amkor.com/semiconductor-story/alook.

16. AV and ADAS Sensors. URL https://community.sw.siemens.com/s/article/AVand- ADAS-Sensors.

17. Гуржій А. М. Основи автоматики та робототехніки: Навчальний посібник/ А.М. Гуржій, А. Т. Нельга, В. М. Співак, О. С. Ітякін:–Дніпро:«Гарант СВ», 2021.- 243с.

18. Lim, B.S. et al. Autonomous Vehicle Ultrasonic Sensor Vulnerability and Impact Assessment. In Proceedings of the IEEE World Forum on Internet of Things (WFIoT), Singapore, 5–8 February 2018; pp. 231–236.

19. Саліхов М. М. Arduino – перспективний інструмент протитопування у робототехніці // Концептуальні шляхи розвитку науки та освіти: матеріали VIII Міжнародної науково-практичної конференції м. Львів, 9-10 червня 2023 року.– Львів: Львівський науковий форум, 2023. – с. 73-79.

20. Bräunl T. Embedded Robotics: From Mobile Robots to Autonomous Vehicles with RaspberryPi and Arduino Springer Nature, 2022. 519 p.

21. Навігаційні системи [Електронний ресурс] : навч. посіб. для студ. Спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / С.Л. Лакоза; КПІ ім. Ігоря Сікорського, 2021. — 80 с.

22. Imad M. et al. Navigation system for autonomous vehicle: A survey. JCSTS., vol. 2, no. 2, pp. 20–35, 2020.

КБПЗ–2026

23. Ковалець І. В. та ін. Технологія планування траєкторій руху мобільних об’єктів з урахуванням перешкод на складній місцевості. Енергетика і автоматика. - 2017. - № 1. - С. 110-122.

24. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web/HTML.

25. Aguilo I. et al. Artificial Intelligence Research and Development Hardcover by IOS Press. 2003. 500 p.

26. Гребенюк Б. А. Розробка підсистеми управлення інтелектуальним роботом / Б. А. Гребенюк // «Automation and Development of Electronic Devices» ADED-2023: Collection of Students' Scientific Paper. – Kharkiv : Kind of Kharkiv National Universitety of Radio Elektronics [electronic edition], 2023. – Part 1. –336p. P. 263-269.

27. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MongoDB.

28. Becker M. et al Obstacle avoidance procedure for mobile robots. ABCM Symposium series in Mechatronics. Vol. 2, 2006. P. 250-257.

29. CORS, XSS [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/maleta/cors-xss-and-csrf-with-examples-in-10-minutes-35k3.

30. AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AJAX.

31. Fetch API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Fetch_API.

32. Звенігородський О.С. Інтелектуальна система планування тактики руху автономного робота в квазістаціонарному середовищі: Дис... канд. техн. наук: 05.13.23. — Д., 2002. — 127 с.

33. Адамів, О.П. Моделі та інтелектуальні засоби адаптивного керування автономним мобільним роботом. Дис. канд.техн. наук. Одеса, 2007. 124 с.

КБПЗ–2026

34. Koval V., Adamiv O., Proc. of the Third IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2005). – Sofia (Bulgaria). – 2005. – P. 120- 124.

35. npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/

36. Install MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://docs.mongodb.com/manual/administration/install-on-linux/

37. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/node-js-ubuntu-18-04-ua.

38. Linux [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Linux.

39. npm-audit [Електронний ресурс] – Режим доступу до ресурсу: https://docs.npmjs.com/cli/audit.

40. Kojima T. et al. Intelligent Technology for More Advanced Autonomous Driving. 2018. Hitachi Review Vol. 67, No. 1. p. 58–63.

41. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL.

42. MongoDB Compass [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/compass.

43. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/

44. Саліхов М. М. Від автоматизованих до автономних транспортних засобів // Наукові досягнення та інновації: шлях до успіху. X Всеукраїнська мультидисциплінарна науково-практична Інтернет-конференція, 31 травня 2023, Україна, Київ : зб. матеріалів — Електрон. дан. — Київ : Ярочéнко Я. В., 2023. — с. 136-142.

45. SPA (Single-page application) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SPA.

КПІ_2026

46. Саліхов М.М. Самокеровані автомобілі та системи їх навігації// Практичні та теоретичні питання розвитку науки та освіти: матеріали VIII Міжнародної науково-практичної конференції: м. Львів, 19-20 червня 2023 року. – Львів: Львівський науковий форум, 2023. – С.53-59.

47. SPA (Single-page application) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SPA.

48. Vargas J. et al. An Overview of Autonomous Vehicles Sensors and Their

49. Vulnerability to Weather Conditions. Sensors. 2021; 21(16):5397.

50.https://www.coursehero.com/file/62601304/Wevolver2020AutonomousVehicec hnologyReportpdf.

КБПЗ_2026