

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

«До захисту допущено»

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

(підпис)

(ініціали, прізвище)

“ ___ ” червня 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп'ютерні системи»

123 «Комп'ютерна інженерія»

на тему: «IoT-система розумного дому з візуалізацією у Node-RED»

Виконав: студент IV курсу, групи КВ-12

Давидюк Микола Юрійович

(підпис)

Керівник ст. викладач каф. СПСКС, к.т.н..

Коляда Костянтин Вячеславович

(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС,

к.т.н., доц. Клятченко Ярослав Михайлович

(підпис)

Рецензент доц. каф. ОТ, к.т.н., доц.

Марковський Олександр Петрович

(підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Київ – 2025 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)

“ ___ ” червня 2025 р.

ЗАВДАННЯ
на дипломний проєкт студента
Давидюк Миколи Юрійовича

1. Тема проєкту «IoT–система розумного дому з візуалізацією у Node-RED»
керівник проєкту ст. викладач каф. СПСКС, к.т.н. Коляда Костянтин Вячеславович,
затверджені наказом по університету від «29» травня 2025 р. № 2205-С
2. Термін подання студентом проєкту _____
3. Вихідні дані до проєкту див. Технічне завдання
4. Зміст пояснювальної записки:
 - 1) Теоретичні основи та аналіз існуючих рішень
 - 2) Обрані компоненти для розробки іот-системи розумного дому
 - 3) Опис розробки іот-системи для розумного дому
 - 4) Тестування програми
5. Перелік графічного матеріалу:
 - 1) Схема взаємодії ESP32 та Node-RED.
 - 2) Схема роботи функції.

- 3) Схему передачі та обробки даних через MQTT.
- 4) Схема підключення датчиків.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я. М., доцент кафедри СПіСКС, к.т.н., доцент		

7. Дата видачі завдання «25» грудня 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Огляд літератури за темою роботи	15.04.2025	
2	Створення та узгодження технічного завдання	27.04.2025	
3	Аналіз існуючих рішень	28.04.2025	
4	Розробка структури застосунку	05.05.2025	
5	Програмна реалізація застосунку	09.05.2025	
6	Розробка дизайну застосунку	15.05.2025	
7	Тестування застосунку	17.05.2025	
8	Підготовка матеріалів першого розділу дипломного проекту	18.05.2025	
9	Підготовка матеріалів другого розділу дипломного проекту	22.05.25	
10	Підготовка матеріалів третього та четвертого розділів дипломного проекту	26.05.25	
11	Підготовка матеріалів графічної частини проекту	02.06.2025	
12	Оформлення технічної документації проекту	03.06.2025	

Студент

Давидюк Микола

Керівник проекту

Костянтин КОЛЯДА

АНОТАЦІЯ

Кваліфікаційна робота містить пояснювальну записку (51 с., 22 рис., список використаної літератури з 12 джерел), 4 додатки, 20 слайдів.

Об'єкт дослідження IoT-система моніторингу внутрішнього середовища для розумного дому на базі мікроконтролера ESP32.

Метою роботи є розробка прототипу системи, здатної збирати дані з сенсорів, передавати їх через мережу MQTT та відображати у зручному для користувача інтерфейсі Node-RED.

У межах проєкту:

- обґрунтовано вибір платформи віртуального моделювання Wokwi та сенсорних компонентів;
- реалізовано підключення сенсорів температури, вологості, газу та руху до ESP32;
- запрограмовано передавання даних через MQTT до сервера;
- реалізовано збереження останніх показників у пам'яті ESP32;
- налаштовано інтеграцію з Node-RED та створено інтерфейс для візуалізації.
- перевірено працездатність системи шляхом налагодження, тестування MQTT-передачі даних та візуалізації в реальному часі.

Програмна реалізація виконана на мові C++ з використанням бібліотек WiFi, PubSubClient, DHTesp та Preferences у середовищі Arduino IDE та онлайн-платформі Wokwi.

Ключові слова: ESP32, Wokwi, MQTT, Node-RED, IoT, розумний дім.

ANNOTATION

The qualification work contains an explanatory note (51 pages, 22 figures, a list of used literature from 12 sources), 4 appendices, 20 slides.

Object of Study: An IoT-system for monitoring the indoor environment for a smart home based on the ESP32 microcontroller.

Aim of the Work: The aim of the work is to develop a prototype system capable of collecting data from sensors, transmitting it via the MQTT network, and displaying it in a user-friendly Node-RED interface.

Within the scope of the project:

The choice of the Wokwi virtual simulation platform and sensor components was justified.

The connection of temperature, humidity, gas, and motion sensors to ESP32 was implemented.

Data transmission via MQTT to the server was programmed.

The saving of the latest readings in ESP32's memory was implemented.

Integration with Node-RED was configured, and an interface for visualization was created.

The system's operability was verified through debugging, testing of MQTT data transmission, and real-time visualization.

Software implementation was performed in C++ using the WiFi, PubSubClient, DHTesp, and Preferences libraries in the Arduino IDE environment and the Wokwi online platform.

Keywords: ESP32, Wokwi, MQTT, Node-RED, IoT, smart home.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ.467200.002 ТЗ	ІоТ-система розумного дому з візуалізацією у Node-RED Технічне завдання	4		
	A4	ІАЛЦ.467200.003 ТП	ІоТ-система розумного дому з візуалізацією у Node-RED Відомість технічного проекту	1		
	A4	ІАЛЦ.467200.004 ПЗ	ІоТ-система розумного дому з візуалізацією у Node-RED Пояснювальна записка	51		
	A4	ІАЛЦ.467200.005 Д1	Взаємодія ESP32 та Node-RED. Схема алгоритму	1		

					ІАЛЦ.467200.001 ОА		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив	Давидюк М.Ю.				Літ.	Аркуш	Аркушів
Перевірив	Коляда К.В.					1	2
Консульт.					КПІ ім. Ігоря Сікорського, ФПМ КВ-12		
Н. контроль	Клятченко Я. М.						
Зав. каф.	Романкевич В.О.						
					ІоТ - система розумного дому з візуалізацією у Node-RED Опис альбому		

ЗМІСТ

1. <u>НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ</u>	2
2. <u>ПІДСТАВА ДЛЯ РОЗРОБКИ</u>	2
3. <u>МЕТА І ПРИЗНАЧЕННЯ РОБОТИ</u>	2
4. <u>ДЖЕРЕЛА РОЗРОБКИ</u>	2
5. <u>ТЕХНІЧНІ ВИМОГИ</u>	2
5.1 <u>Вимоги до програмного продукту, що розробляється</u>	2
5.2 <u>Вимоги до апаратного забезпечення</u>	3
5.3 <u>Вимоги до програмного та апаратного забезпечення користувача</u>	3
6. <u>ЕТАПИ РОЗРОБКИ</u>	3

					ІАЛЦ.467200.002 ТЗ			
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>				
<i>Розроб.</i>		Давидюк М.Ю.			ІюТ - система розумного дому з візуалізацією у Node-RED Технічне завдання	<i>Літ.</i>	<i>Арк.</i>	<i>Акрушів</i>
<i>Перевір.</i>		Коляда К.В.					1	3
<i>Н. Контр.</i>		Клятченко Я. М.				КПІ ім. Ігоря Сікорського, ФПМ КВ-12		
<i>Затверд.</i>		Романкевич В.О.						

1. Найменування та галузь розробки

Назва розробки: « IoT - система розумного дому з візуалізацією у Node-RED ».

Галузь застосування: автоматизовані системи керування в побуті, системи моніторингу внутрішнього середовища.

2. Підстава для розробки

Підставою для розробки є завдання на дипломне проєктування для здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. Мета і призначення роботи

Метою проєкту є створення IoT-системи для моніторингу температури, вологості, концентрації газу та руху з використанням мікроконтролера ESP32, передачею даних через MQTT та візуалізацією у Node-RED. Система має працювати в реальному часі, зберігати останні вимірювання у пам'яті пристрою та демонструвати принципи реалізації розумного дому. Розробка здійснюється в онлайн-симуляторі Wokwi без використання фізичного обладнання.

4. Джерела розробки

Основними джерелами є технічна документація на мікроконтролер ESP32, бібліотеки Arduino та PubSubClient, публікації з офіційних ресурсів MQTT.org, документація Node-RED, електронні навчальні матеріали та науково-технічна література з теми Інтернету речей.

5. Технічні вимоги

5.1 Вимоги до програмного продукту, що розробляється

- Передача даних через протокол MQTT з використанням ESP32;

					ІАЛЦ. 467200.002 ТЗ	Арк.
						2
Змн.	Арк.	№ докум.	Підпис	Дата		

- Зчитування значень із сенсорів температури, вологості, газу та руху;
- Збереження останніх значень у пам'яті за допомогою Preferences;
- Виведення даних у Node-RED Dashboard у вигляді гейджів і текстових полів;
- Можливість запуску та тестування у середовищі Wokwi;

5.2 Вимоги до апаратного забезпечення

- Доступ до вебсайту wokwi.com;
- Стабільне інтернет-з'єднання.

5.3 Вимоги до програмного та апаратного забезпечення користувача

- Операційна система: Windows / Linux / macOS
- Браузер Google Chrome або Firefox ;
- Встановлене середовище Node.js для локального запуску Node-RED.

6. Етапи розробки

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1	Огляд літератури за темою роботи	15.04.2025
2	Створення та узгодження технічного завдання	27.04.2025
3	Аналіз існуючих рішень	28.04.2025
4	Розробка структури застосунку	05.05.2025
5	Програмна реалізація застосунку	09.05.2025
6	Розробка дизайну застосунку	15.05.2025
7	Тестування застосунку	17.05.2025
8	Підготовка матеріалів першого розділу дипломного проекту	18.05.2025
9	Підготовка матеріалів другого розділу дипломного проекту	22.05.25
10	Підготовка матеріалів третього та четвертого розділів дипломного проекту	26.05.25
11	Підготовка матеріалів графічної частини проекту	02.06.2025
12	Оформлення технічної документації проекту	03.06.2025
13	Попередній огляд матеріалів БДП на кафедрі	04.06.2025

					ІАЛЦ. 467200.002 ТЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		3

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ.467200.004 ПЗ	ІоТ-система розумного дому з візуалізацією у Node-RED	51		
			Пояснювальна записка			
	A4	ІАЛЦ.467200.005 Д1	Взаємодія ESP32 та Node-RED.	1		
			Схема алгоритму			
	A4	ІАЛЦ.467200.006 Д2	Робота функції loop().	1		
			Схема алгоритму			
	A4	ІАЛЦ.467200.007 Д3	Передача та обробка даних через MQTT.	1		
			Схема алгоритму			
	A3	ІАЛЦ.467200.008 Е2	Підключення датчиків до ESP32.	1		
			Схема функціональна			

ІАЛЦ. 467200.003 ТП				
Змін.	Арк.	№ докум.	Підпис	Дата
Розробив	Давидюк М.Ю.			
Перевірив	Коляда К.В.			
Консульт.				
Н. контроль	Клятченко Я. М.			
Зав. каф.	Романкевич В.О.			
ІоТ-система розумного дому з візуалізацією даних у Node-RED			Літ.	Аркуш
Відомість технічного проекту				1
Відомість технічного проекту			КПІ ім. Ігоря Сікорського, ФПМ КВ-12	

Пояснювальна записка

до дипломного проєкту

на тему: «ІoT-система розумного дому з візуалізацією у Node-RED»

Київ – 2025

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	3
ВСТУП	4
1. ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ	5
1.1. Концепція розумного дому	5
1.2. Архітектура IoT-систем і протоколи.....	6
1.3. Огляд платформ візуалізації даних	9
1.4. Механізм retained-повідомлень та персистентне збереження	11
1.5. Апаратні компоненти: огляд та застосування в IoT-системах	13
1.6. Обґрунтування теми дипломної роботи.	14
2. ОБРАНІ КОМПОНЕНТИ ДЛЯ РОЗРОБКИ ІОТ-СИСТЕМИ РОЗУМНОГО ДОМУ	15
2.1. Мікроконтролер: ESP32	15
2.2. Датчик температури та вологості: DHT22	16
2.3. Газовий датчик: MQ-2.....	18
2.4. Датчик руху: Пасивний інфрачервоний (PIR) сенсор.....	19
2.5. Платформа моделювання: Wokwi.....	20
2.6. Протокол обміну повідомленнями: MQTT	22
2.7. Інструмент візуалізації: Node-RED.....	22
3. РОЗРОБКА ІОТ-СИСТЕМИ РОЗУМНОГО ДОМУ	24
3.1. Програмування мікроконтролера ESP32 у середовищі Wokwi.....	24
3.1.1. Налаштування середовища Wokwi	24
3.1.2. Підключення бібліотек.....	26
3.1.3. Конфігурація WiFi та MQTT-клієнта	28
3.1.4. Реалізація підключення до сервера та обробка з'єднання.....	29
3.2. Підключення сенсорів температури, вологості, газу та руху.....	31
3.2.1. Ініціалізація DHT22.....	32
3.2.3. Обробка цифрового сигналу з PIR-сенсора	33
3.2.4. Вивід отриманих даних у консоль	33
3.3. Передача даних через MQTT та обробка повідомлень	34
3.3.1. Формування MQTT-топиків для кожного сенсора	34
3.3.2. Публікація даних у вигляді рядків.....	35

ІАЛЦ.467200.004 ПЗ				
Змін.	Арк.	№ докум.	Підпис	Дата
		Давидюк М.Ю.		
		Коляда К.В.		
		Клятченко Я. М.		
		Романкевич В.О.		
IoT - система розумного дому з візуалізацією у Node-RED Пояснювальна записка				
		Літ.	Аркуш	Акрушів
		1	1	51
КП Ім. Ігоря Сікорського, ФПМ КВ-12				

3.3.3. Створення callback-функції для обробки вхідних повідомлень	35
3.3.4. Автоматичне повторне з'єднання з сервером (reconnect).....	36
3.4. Збереження даних у пам'яті ESP32.....	37
3.5. Використання retained-повідомлень для збереження останнього стану	39
3.6. Інтеграція з Node-RED та створення інтерфейсу візуалізації	40
4. ТЕСТУВАННЯ ПРОГРАМИ.....	45
4.1. Тестування підключення та передачі даних через Wi-Fi.....	45
4.2. Тестування підключення та обміну даними через MQTT	46
4.3. Тестування функціональності віртуальних сенсорів	48
ВИСНОВКИ.....	49
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	50

ДОДАТКИ

Додаток А. Копії графічного матеріалу.

- 1) ІАЛЦ.467200.005 Д1. Взаємодія ESP32 та Node-RED. Схема алгоритму
- 2) ІАЛЦ.467200.006 Д2. Робота функції loop(). Схема алгоритму
- 3) ІАЛЦ.467200.007 Д3. Передача та обробка даних через MQTT. Схема алгоритму
- 4) ІАЛЦ.467200.008 Е2. Підключення датчиків до ESP32. Схема функціональна

Додаток Б. Фрагменти програмного коду

Додаток В. Презентація

					ІАЛЦ.467200.004 ПЗ	Арк.
						2
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

IoT (англ. Internet of Things — Інтернет речей) — це концепція об'єднання фізичних пристроїв (сенсорів, контролерів, побутової техніки тощо) в єдину мережу для автоматичного обміну даними через Інтернет без участі людини. Застосовується для автоматизації, моніторингу та керування в побуті, промисловості, транспорті тощо.

MQTT (англ. Message Queuing Telemetry Transport) — це легкий протокол обміну повідомленнями, створений для пристроїв з обмеженими ресурсами. Працює за моделлю «видавець–підписник» через MQTT-сервер і широко використовується в IoT для передавання даних і керування пристроями.

WiFi (Wireless Fidelity) – стандарт бездротової мережі, що дозволяє мікроконтролерам (наприклад, ESP32) підключатися до локальної мережі або Інтернету для обміну даними без використання фізичних дротів.

Топік (Topic) – адреса (рядок) в MQTT-протоколі, яка використовується для маршрутизації повідомлень між видавцем та підписниками. Має ієрархічну структуру, наприклад: /ThinkIOT/temp.

Dashboard – інтерактивна панель користувача в Node-RED, яка дозволяє в реальному часі візуалізувати отримані з пристрою дані у вигляді графіків, шкал, текстових полів тощо.

					ІАЛЦ.467200.004 ПЗ	Арк.
						3
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

У сучасному світі концепція розумного дому активно впроваджується у повсякденне життя, забезпечуючи підвищений комфорт, енергоефективність і безпеку. В основі таких систем лежать технології Інтернету речей (IoT), які об'єднують різноманітні сенсори та пристрої в єдину мережу з можливістю обміну даними.

Серед широкого спектра пристроїв, що використовуються в IoT, мікроконтролери відіграють ключову роль як вузли збору та обробки інформації. Особливої популярності набули мікроконтролери ESP32 завдяки поєднанню обчислювальної потужності, бездротових інтерфейсів та доступної ціни.

Ця бакалаврська кваліфікаційна робота присвячена розробці IoT-системи моніторингу параметрів внутрішнього середовища (температури, вологості, рівня газу та наявності руху), яка функціонує на базі ESP32, передає дані через протокол MQTT та візуалізує їх в Node-RED.

Особливістю роботи є те, що вся система була спроектована та протестована у віртуальному середовищі Wokwi без використання фізичного обладнання, що дозволило ефективно розробити, налагодити та продемонструвати її функціональність.

					ІАЛЦ.467200.004 ПЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

1. ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Концепція розумного дому

Концепція розумного дому передбачає створення інтелектуального середовища, у якому всі побутові пристрої працюють узгоджено, автоматично реагуючи на зміни навколишнього середовища або діючи за командами користувача. Такий дім здатен самостійно регулювати освітлення, температуру, вологість, контролювати доступ до приміщень, стежити за витоком газу чи води, повідомляти про надзвичайні ситуації та економно використовувати ресурси. Головною особливістю розумного дому є те, що людина безпосередньо не керує більшістю процесів — вони відбуваються автоматично або з можливістю дистанційного управління через смартфон, планшет чи комп'ютер.

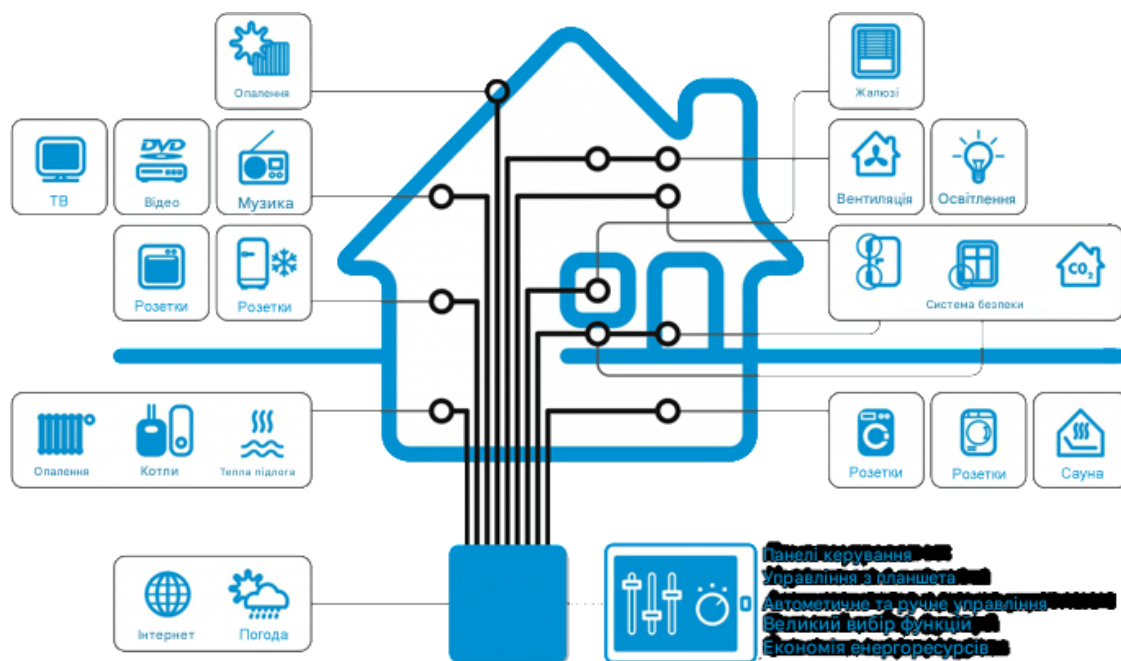


Рисунок 1.1 - Схематичне зображення системи розумного дому

Реалізація такої системи базується на використанні різноманітних сенсорів і виконавчих пристроїв, які з'єднані між собою мережею та централізовано

					ІАЛЦ.467200.004 ПЗ	Арк.
						5
Змн.	Арк.	№ докум.	Підпис	Дата		

обробляються за допомогою програмного забезпечення. Сенсори постійно фіксують стан середовища - наприклад, температуру, вологість або наявність руху - а виконавчі пристрої, реагуючи на ці дані, приймають відповідні рішення: вмикають опалення, відкривають жалюзі або надсилають повідомлення власнику. Комунікація між пристроями здійснюється через дротові або бездротові протоколи, серед яких найпоширенішими є Wi-Fi, Zigbee, Z-Wave та MQTT.

Інтерфейсом для керування системою зазвичай слугують мобільні застосунки або вебпанелі керування, де користувач може в реальному часі відстежувати стан усіх пристроїв, переглядати історію подій та налаштовувати сценарії автоматизації. Наприклад, при виявленні руху вночі система може автоматично увімкнути світло в коридорі, а при виявленні диму — надіслати сповіщення на телефон і відключити подачу живлення.

Загалом, розумний дім - це приклад втілення технологій Інтернету речей у повсякденному житті, який робить житло не лише зручнішим, а й безпечнішим, технологічнішим та економнішим.

1.2. Архітектура IoT-систем і протоколи

Після розгляду загальної концепції розумного дому доцільно перейти до технічних основ, які забезпечують його функціонування - а саме, до архітектури IoT-систем та комунікаційних протоколів[1]. Саме вони створюють інфраструктуру, завдяки якій пристрої здатні взаємодіяти між собою, зберігати дані та реагувати на команди користувача.

Типова IoT-система розумного дому має багаторівневу структуру. На базовому рівні знаходяться сенсори та актуатори - фізичні пристрої, які зчитують стан навколишнього середовища або виконують дії. Сенсори вимірюють температуру, вологість, рух, рівень газу, освітленість тощо. Актуатори керують приладами: вмикають освітлення, відкривають двері, регулюють опалення або

					ІАЛЦ.467200.004 ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

активують сигналізацію. Ці пристрої фізично розміщуються у приміщеннях і забезпечують зв'язок між цифровим середовищем та реальним світом.

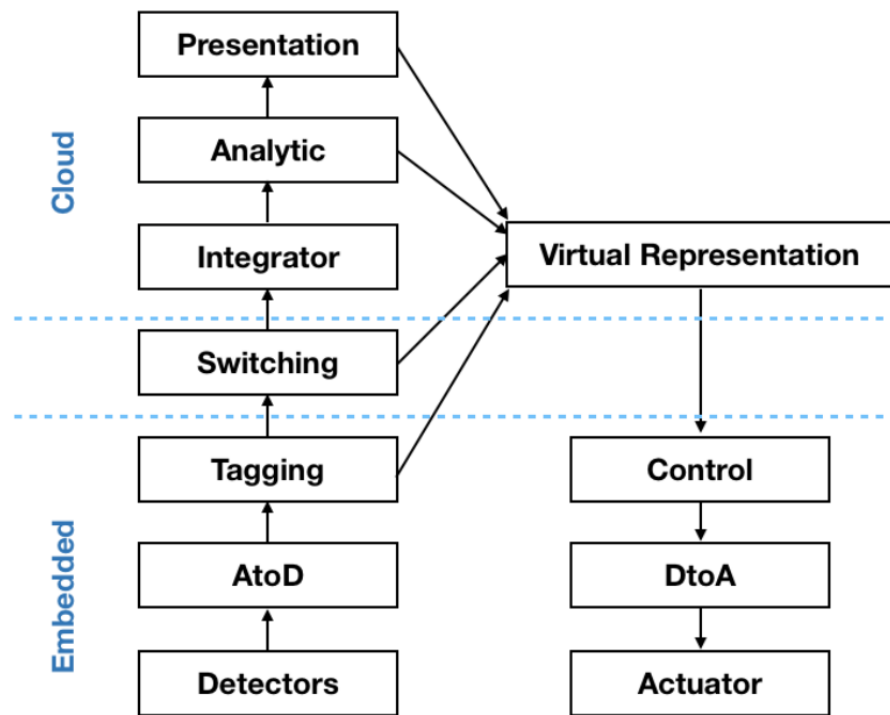


Рисунок 1.2 - Багаторівнева архітектура IoT-системи

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 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 відповідає за маршрутизацію повідомлень між пристроями, панеллю керування та іншими підписниками.

MQTT - один з найпоширеніших протоколів, що використовується в системах розумного дому. Його головні переваги - легкість, низьке енергоспоживання, малий розмір переданих пакетів і надійність доставки. Основний принцип роботи MQTT – це обмін даними за моделлю «публікація й підписка»: пристрої публікують повідомлення у певні топіки, а ті, хто підписаний на ці топіки, отримують повідомлення автоматично. Протокол підтримує декілька рівнів надійності доставки (QoS) і має функцію retained-повідомлень — це означає, що сервер може зберігати останнє значення повідомлення для кожного топіка, і воно буде автоматично доставлено кожному новому підписнику. У контексті розумного дому це дозволяє, наприклад, завжди бачити актуальне значення температури або стану руху після перезавантаження інтерфейсу.

Проте MQTT - не єдиний протокол, який використовується в IoT. Значне поширення має HTTP, добре відомий як основа для роботи вебсторінок. У системах розумного дому HTTP часто застосовується для одноразових запитів: наприклад, для конфігурації пристрою через вебінтерфейс або надсилання даних на REST-сервер. Попри зручність, HTTP має певні обмеження: він не підтримує постійного з'єднання, створює додаткове навантаження на мережу і не підходить для частого надсилання даних з сенсорів у реальному часі.

Інший популярний протокол - CoAP - був розроблений спеціально для пристроїв із обмеженими ресурсами. На відміну від HTTP, він базується на UDP, що дозволяє зменшити обсяг трафіку. CoAP дозволяє організувати обмін даними у форматі «запит-відповідь», а також підтримує спостереження за ресурсами (observe mode), що є корисним для отримання змін у значеннях сенсорів. Проте використання UDP робить його менш надійним у нестабільних мережах.

Для передачі даних між пристроями в межах однієї локальної мережі широко застосовуються бездротові протоколи ближньої дії - Zigbee, Z-Wave та

					ІАЛЦ.467200.004 ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

Bluetooth Low Energy (BLE). Zigbee — відкритий протокол з низьким енергоспоживанням, який працює у форматі mesh-мережі. Це означає, що пристрої можуть ретранслювати сигнал один одному, що підвищує надійність і радіус покриття. Z-Wave має подібні властивості, але є закритим стандартом, що потребує ліцензії. Обидва протоколи популярні у комерційних системах автоматизації, зокрема для керування освітленням, жалюзі, дверними замками. Bluetooth Low Energy застосовується переважно для мобільних пристроїв або точкових рішень, наприклад, смарт-замків, датчиків біля входу, персональних маяків.

Новим універсальним стандартом, який стрімко набирає популярності, є Matter — відкритий протокол, що дозволяє забезпечити сумісність між пристроями різних виробників. Він підтримується такими компаніями, як Apple, Google, Amazon і Samsung. Matter працює поверх IP (TCP/IP) і може використовувати Wi-Fi, Ethernet або Thread для транспортування даних. Його мета — зробити інтеграцію пристроїв простою, надійною та безпечною, без потреби у шлюзах від конкретного виробника.

Загалом, архітектура IoT-системи у розумному домі — це складне поєднання апаратних пристроїв, комунікаційних протоколів та програмних інтерфейсів, які разом створюють єдину функціональну екосистему. Саме завдяки грамотному вибору та взаємодії цих елементів вдається досягти стабільної, безпечної й гнучкої роботи системи, що автоматизує житло та покращує повсякденне життя користувачів.

1.3. Огляд платформ візуалізації даних

Ключовою складовою IoT-системи в розумному домі є можливість наочно відображати дані, отримані від сенсорів і пристроїв. Користувачеві важливо не просто знати, що температура в кімнаті — 22,7 °C або що зафіксовано рух, а бачити це в зручному форматі: на графіках, індикаторах, текстових повідомленнях або в режимі реального часу. Для цього використовуються

					ІАЛЦ.467200.004 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

платформи візуалізації даних, які дозволяють створювати інтерфейси моніторингу, графіки історії показників, повідомлення про події, а також забезпечують можливість керування пристроями.

Однією з основних платформ для візуалізації даних є Node-RED[2]. Це середовище з відкритим кодом, яке дозволяє створювати логіку взаємодії пристроїв у вигляді потоку даних (flows). Node-RED має інтегровану підтримку MQTT, що дозволяє легко підписуватися на повідомлення від пристроїв у реальному часі. Користувачі можуть візуально розміщувати вузли (nodes), з'єднувати їх лініями та формувати логіку обробки інформації без необхідності програмувати вручну. Важливою частиною Node-RED є dashboard-модуль, який дозволяє створювати вебінтерфейс із гейджами, графіками, кнопками, текстовими блоками - ідеально підходить для реалізації локального моніторингу «розумного дому».

Ще однією потужною платформою є Grafana — рішення для створення графіків і панелей керування, що часто використовується разом із базами даних часових рядів, як-от InfluxDB або Prometheus. Grafana дозволяє створювати динамічні графіки зміни температури, вологості, рівня газу тощо, що особливо корисно для аналітики. Вона більше орієнтована на візуалізацію історичних даних, а не миттєвого стану. Тому в контексті розумного дому Grafana зазвичай використовується як доповнення до інших систем — наприклад, для збереження даних у InfluxDB через MQTT, а далі — відображення цих даних у Grafana.

Також варто згадати платформи, орієнтовані на початківців, наприклад Blynk або Tinkercad Circuits, які дозволяють швидко побудувати мобільний застосунок для керування IoT-пристроями, але мають обмежену функціональність і часто є платними у просунутих версіях. Blynk — це потужна платформа для створення мобільних додатків для управління IoT-пристроями, яка дозволяє користувачам створювати інтерфейси для мобільних пристроїв (iOS та Android) для моніторингу та керування пристроями в реальному часі. Платформа підтримує Arduino, ESP32, Raspberry Pi і інші платформи, дозволяючи користувачам створювати інтерфейси для мобільних додатків без необхідності

					ІАЛЦ.467200.004 ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

програмування. Tinkercad Circuits — це онлайн-інструмент для моделювання та симуляції електронних схем і проектів на основі Arduino. Він дозволяє створювати електронні схеми, програмувати їх і симулювати роботу в браузері, що робить його ідеальним для навчання та прототипування.

Таким чином, платформи візуалізації є важливою ланкою у структурі IoT-системи. Вони дозволяють не лише переглядати дані, а й взаємодіяти з пристроями, створювати сценарії автоматизації, історичні звіти, а також забезпечують зрозумілий інтерфейс користувачу — незалежно від технічної складності «розумного дому».

1.4. Механізм retained-повідомлень та персистентне збереження

У світі систем обміну повідомленнями існує важлива концепція, відома як retained-повідомлення[3]. Її суть полягає в тому, що посередник, будь то сервер чи центральний сервер, бере на себе відповідальність за зберігання останнього опублікованого повідомлення, пов'язаного з конкретною темою, каналом або ідентифікатором. Головна мета цього механізму — забезпечити новим учасникам системи, клієнтам або підписникам, миттєвий доступ до найсвіжішої інформації, до актуального стану або останнього відомого значення. Це означає, що коли хтось вперше приєднується до певної інформаційної стрічки, він одразу отримує останнє збережене значення, минаючи необхідність очікування наступного оновлення. Важливо, що збереження повідомлення відбувається незалежно від того, чи були активні споживачі цієї інформації в момент її публікації. Коли ж з'являється нове повідомлення для того ж ідентифікатора з вказівкою на збереження, попередньо збережене повідомлення автоматично замінюється.

Сценарії використання retained-повідомлень є досить різноманітними. Їх часто застосовують для фіксації поточного статусу пристроїв або сервісів, наприклад, чи вони онлайн або офлайн, увімкнені чи вимкнені, що дозволяє новим системам моніторингу або інтерфейсам користувача швидко отримувати

					ІАЛЦ.467200.004 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

цю інформацію. Також вони корисні для збереження останніх вимірних значень від датчиків, таких як температура, вологість або рівень освітленості, надаючи актуальні дані новим користувачам систем моніторингу. Крім того, retained-повідомлення можуть використовуватися для зберігання останніх налаштувань системи або пристрою, щоб нові клієнти одразу знали поточну конфігурацію, або для інформування про останню доступну версію програмного забезпечення.

Іншою фундаментальною концепцією є персистентне збереження, яке визначає здатність системи зберігати дані таким чином, що вони не втрачаються після завершення роботи процесу, вимкнення живлення або інших тимчасових збоїв. Це досягається шляхом запису даних у нелетку пам'ять, таку як жорсткий диск, SSD або флеш-пам'ять, що дозволяє відновлювати їх після перезапуску системи. Ключовими характеристиками персистентного збереження є саме збереження в нелеткій пам'яті, здатність переживати перезапуски та стійкість до збоїв, що підвищує загальну надійність системи.

Існує тісний зв'язок між retained-повідомленнями та персистентним збереженням. Для того щоб механізм retained-повідомлень працював ефективно, посередник, який їх зберігає, обов'язково повинен використовувати персистентне збереження. Без нього всі збережені повідомлення будуть втрачені після перезапуску посередника, що зробить функціональність retained-повідомлень ненадійною та практично марною.

Приклади використання подібних концепцій можна знайти в різних технологіях. Деякі протоколи та сервери обміну повідомленнями також можуть мати механізми для збереження останнього стану. У вебсокетах з підтримкою стану сервер може зберігати останню інформацію про клієнта для відновлення контексту при перепідключенні. Бази даних реального часу можуть підтримувати "закріплені" або "останні відомі" значення, які автоматично надсилаються новим підписникам. Системи керування конфігурацією можуть зберігати останню застосовану конфігурацію для керованих вузлів, щоб нові вузли одразу отримували актуальні налаштування при підключенні.

					ІАЛЦ.467200.004 ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

Таким чином, хоча конкретні реалізації та термінологія можуть варіюватися, загальні ідеї збереження останнього стану для нових споживачів (retained-повідомлення) та забезпечення збереження даних між сесіями (персистентне збереження) є критично важливими для багатьох розподілених систем, особливо тих, що обробляють дані в реальному часі або потребують швидкого надання початкового контексту.

1.5. Апаратні компоненти: огляд та застосування в IoT-системах

IoT функціонує завдяки взаємодії різноманітних апаратних компонентів, які збирають, обробляють та передають дані про навколишнє середовище для подальшого аналізу та прийняття рішень[4]. Серед ключових елементів виділяються мікроконтролери та мікропроцесори, що є "мізками" пристроїв, відповідаючи за виконання коду, обробку даних та керування іншими компонентами. Мікроконтролери (MCU), такі як Arduino Uno (на базі ATmega328P), STM32 та Raspberry Pi Pico (на базі RP2040), є енергоефективними інтегрованими схемами з процесорним ядром, пам'яттю та периферією, ідеальні для керування апаратним забезпеченням у реальному часі. На противагу їм, мікропроцесори (MPU), як серія Raspberry Pi та Intel Atom/Celeron, є потужнішими та здатні запускати повноцінні операційні системи, хоча й потребують зовнішньої пам'яті.

Важливу роль відіграють сенсори (датчики), які перетворюють фізичні величини на електричні сигнали. До цієї категорії належать датчики температури та вологості (наприклад, DHT11, BME280, DS18B20), газові датчики (наприклад, MQ-135, Sensirion SGP30), датчики руху (ультразвукові HC-SR04, радарні, акселерометри/гіроскопи IMU), датчики освітленості (LDR, BH1750), датчики тиску (BMP180, MPX-серія) та датчики відстані (інфрачервоні, лазерні). Кожен з них має свої особливості та застосування в залежності від вимірюваної величини та необхідної точності.

					ІАЛЦ.467200.004 ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

Для забезпечення зв'язку IoT-пристроїв з мережею використовуються різноманітні модулі зв'язку. Серед них WiFi модулі (наприклад, ESP-01, NodeMCU), Bluetooth модулі (HC-05/HC-06, BLE), стільникові модулі (2G/3G/LTE/NB-IoT/LTE-M), LoRa/LoRaWAN модулі (RA-02, SX1276), Zigbee модулі (CC2530) та Ethernet модулі (ENC28J60). Вибір модуля залежить від вимог до дальності зв'язку, енергоспоживання та пропускної здатності.

Окрім основних категорій, важливими є джерела живлення (батареї, блоки живлення, сонячні панелі), модулі пам'яті (SD-карти, EEPROM, Flash-пам'ять), інтерфейсні модулі (дисплеї, кнопки, світлодіоди), реле та транзистори для керування навантаженнями, а також корпуси та кріплення для забезпечення захисту та зручності використання.

Вибір конкретних апаратних компонентів для будь-якої IoT-системи є результатом компромісу між функціональними вимогами, продуктивністю, енергоефективністю, вартістю та умовами експлуатації. Розуміння спектру доступних компонентів відкриває можливості для створення різноманітних та ефективних IoT-рішень.

1.6. Обґрунтування теми дипломної роботи.

Тема дипломного проекту, що передбачає розробку IoT-системи для моніторингу параметрів внутрішнього середовища (температури, вологості, якості повітря) та виявлення руху з подальшою візуалізацією даних через Node-RED, є актуальною в контексті зростаючої популярності систем розумного дому. Розробка такої системи надасть можливість дослідити та практично застосувати технології для створення комфортного, безпечного та енергоефективного житлового простору. Отримані знання та розроблений прототип можуть стати основою для подальшого впровадження функцій автоматизації, керування кліматом, безпеки та моніторингу в середовищі розумного дому, що відповідає сучасним тенденціям розвитку житлових технологій.

					ІАЛЦ.467200.004 ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

2. ОБРАНІ КОМПОНЕНТИ ДЛЯ РОЗРОБКИ ІОТ-СИСТЕМИ РОЗУМНОГО ДОМУ

2.1. Мікроконтролер: ESP32

Опис[5]: ESP32 є потужним та універсальним мікроконтролерним модулем, що поєднує в собі двоядерний процесор, вбудовану підтримку WiFi та Bluetooth, велику кількість GPIO-пінів, а також різноманітні периферійні інтерфейси.

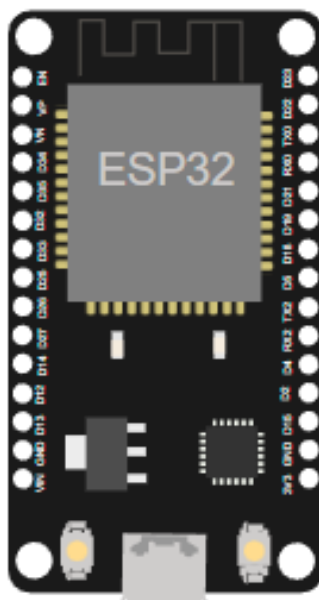


Рисунок 2.1- Мікроконтролер: ESP32

Принцип роботи: ESP32 функціонує як центральний мозок системи, відповідаючи за збір даних від сенсорів, їхню обробку, встановлення мережових з'єднань та обмін даними за протоколом MQTT. Двоядерна архітектура забезпечує можливість паралельного виконання завдань, що підвищує продуктивність системи. Вбудовані WiFi та Bluetooth модулі спрощують бездротову комунікацію.

Чому обрано ESP32:

Інтегрований Wi-Fi: Ключова вимога для безпосереднього підключення до локальної мережі та передачі даних на MQTT-сервер без використання додаткових модулів.

Достатня обчислювальна потужність та пам'ять: Забезпечує можливість ефективної обробки даних від кількох сенсорів та виконання мережевих протоколів.

Гнучкість та велика кількість GPIO: Дозволяє підключати необхідний набір сенсорів та потенційні майбутні розширення системи.

Підтримка Arduino IDE та великої кількості бібліотек: Спрощує процес розробки програмного забезпечення завдяки знайомому середовищу та наявності готових рішень для роботи з різними компонентами та протоколами.

Енергоефективність (у режимах сну): Хоча поточний проєкт не фокусується на автономному живленні, ця характеристика є важливою для потенційних майбутніх ітерацій.

Порівняння з альтернативами:

Arduino Uno + ESP8266: Комбінація Arduino Uno (для обробки даних) та ESP8266 (для Wi-Fi) є можливим варіантом, але ускладнює схему підключення та програмування. ESP32 інтегрує обидві функціональності в одному чипі, що робить рішення більш компактним та ефективним.

Raspberry Pi Zero W: Raspberry Pi Zero W має більшу обчислювальну потужність, але значно вище енергоспоживання та складнішу операційну систему, що може бути надлишковим для простого збору та передачі даних від сенсорів. ESP32 є більш енергоефективним та швидким у завантаженні для даного застосування.

2.2. Датчик температури та вологості: DHT22

Опис[6]: DHT22 є цифровим датчиком температури та вологості, що забезпечує відносно точні вимірювання в широкому діапазоні. Він використовує

					ІАЛЦ.467200.004 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

ємнісний сенсор вологості та термістор для вимірювання температури та передає дані через однопровідний цифровий інтерфейс.

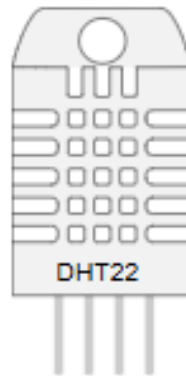


Рисунок 2.2- Датчик температури та вологості: DHT22

Принцип роботи: Датчик періодично вимірює температуру та вологість навколишнього середовища та перетворює ці значення в цифровий сигнал, який може бути зчитаний мікроконтролером.

Чому обрано DHT22:

Комбіноване вимірювання: Надання двох важливих параметрів в одному пристрої спрощує схему та зменшує кількість необхідних компонентів.

Достатня точність для розумного дому: Точність $\pm 0.5^{\circ}\text{C}$ для температури та $\pm 2\text{-}5\%$ RH для вологості є прийнятною для більшості застосувань у моніторингу домашнього клімату.

Широкий діапазон вимірювання: Охоплює типові температурні (-40°C до $+80^{\circ}\text{C}$) та вологісні (0% до 100% RH) діапазони внутрішнього середовища.

Простота підключення та використання з Arduino: Існує велика кількість готових бібліотек для Arduino IDE, що спрощує інтеграцію датчика в проєкт.

Порівняння з альтернативами:

DHT11[7]: Дешевший, але менш точний ($\pm 2^{\circ}\text{C}$, $\pm 5\%$ RH) та з вузьким діапазоном вимірювання. Для більш точного моніторингу DHT22 є кращим вибором.

					ІАЛЦ.467200.004 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

ВМЕ280[8]: Більш точний та функціональний датчик (також вимірює тиск), але дорожчий та потребує інтерфейсу I2C, що може бути дещо складнішим для початківців. Для базового моніторингу температури та вологості DHT22 є оптимальним за співвідношенням ціна/якість.

2.3. Газовий датчик: MQ-2

Опис[9]: MQ-2 є аналоговим газовим датчиком, чутливим до широкого спектру горючих газів та диму (пропан, бутан, LPG, дим). Він використовує нагрівальний елемент для забезпечення необхідної робочої температури чутливого шару, опір якого змінюється залежно від концентрації газів.



Рисунок 2.3- Газовий датчик: MQ-2

Принцип роботи: Зміна концентрації газів у повітрі призводить до зміни опору чутливого елемента датчика, що, в свою чергу, змінює вихідну напругу. Ця аналогова напруга зчитується аналоговим входом ESP32.

Чому обрано MQ-2:

Низька вартість: Економічно вигідний варіант для початкового етапу розробки та демонстрації можливості моніторингу якості повітря.

Простота використання: Легко підключається до аналогового входу мікроконтролера.

Змн.	Арк.	№ докум.	Підпис	Дата

Чутливість до широкого спектру газів: Дозволяє фіксувати загальні зміни в якості повітря або наявність потенційно небезпечних газів.

Порівняння з альтернативами:

Цифрові датчики якості повітря (наприклад, SGP30): Забезпечують більш точні та калібровані значення VOCs та еквівалентного CO₂, а також мають вбудовану компенсацію впливу температури та вологості. Однак вони значно дорожчі та можуть вимагати складнішої програмної обробки. Для базового виявлення змін у якості повітря MQ-2 є прийнятним рішенням.

2.4. Датчик руху: Пасивний інфрачервоний (PIR) сенсор

Опис[10]: PIR-сенсор є електронним датчиком, що виявляє зміни в інфрачервоному (тепловому) випромінюванні, спричинені рухом об'єктів з певною температурою (наприклад, людей або тварин) у його зоні чутливості.

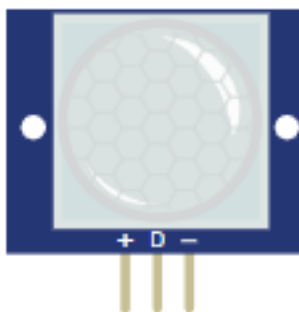


Рисунок 2.4-Датчик руху

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

Чому обрано PIR-сенсор:

Просте виявлення руху: Ефективний та надійний спосіб виявлення

					ІАЛЦ.467200.004 ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

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 дорогими та можуть споживати більше енергії. Для базового виявлення присутності PIR-сенсор є оптимальним за співвідношенням ціна/продуктивність.

2.5. Платформа моделювання: Wokwi

Опис[11]: Wokwi є безкоштовним онлайн-симулятором для розробки проєктів на базі мікроконтролерів, включаючи ESP32. Він надає віртуальне середовище для створення електронних схем, написання коду Arduino та його виконання з можливістю моніторингу через віртуальний послідовний порт та емуляцію мережевих з'єднань.

Принцип роботи: Wokwi емулює апаратне забезпечення ESP32 та підключених компонентів, дозволяючи розробникам тестувати свій код у віртуальному середовищі без необхідності використання фізичного обладнання на початкових етапах розробки.

					ІАЛЦ.467200.004 ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

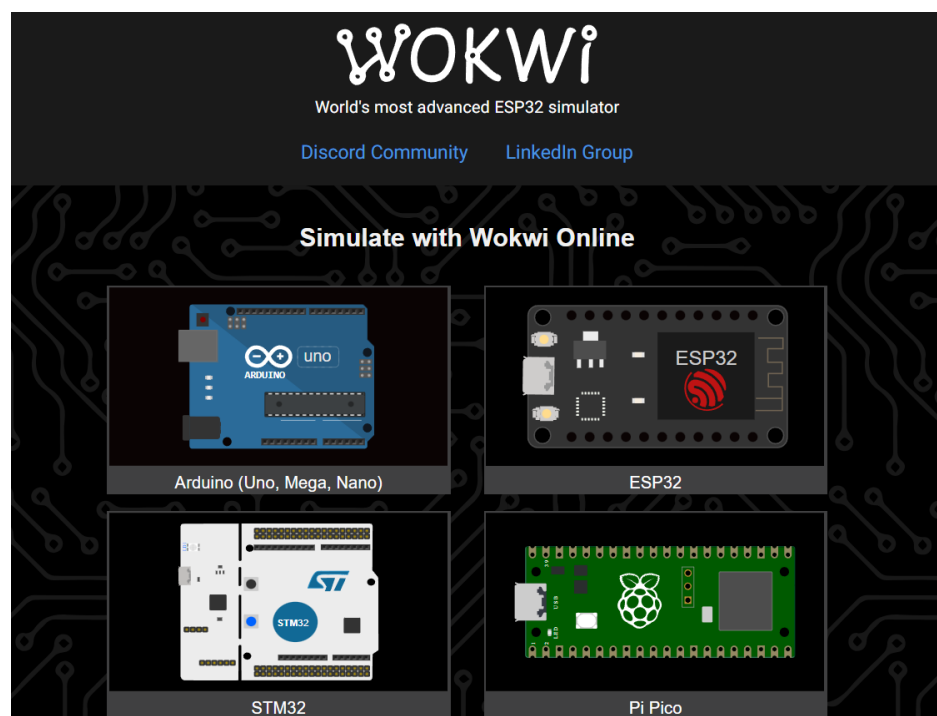


Рисунок 2.5-Онлайн симулятор Wokwi

Чому обрано Wokwi:

Зручність та доступність: Весь процес розробки та тестування відбувається онлайн, що не потребує встановлення складного програмного забезпечення.

Спеціалізація на ESP32: Забезпечує точну емуляцію мікроконтролера та підтримку його специфічних бібліотек.

Моделювання мережевих з'єднань: Дозволяє тестувати взаємодію з MQTT-сервером без реального підключення до мережі на початкових етапах.

Вбудований серійний монітор: Спрощує налагодження коду та відстеження вихідних даних.

Порівняння з альтернативами:

Tinkercad: Також онлайн-симулятор з підтримкою Arduino, але менш спеціалізований для ESP32 та має обмежені можливості моделювання мережевих з'єднань.

Proteus: Потужний професійний інструмент для моделювання електронних схем, але є платним, вимагає встановлення та може бути складнішим у використанні для початківців.

					ІАЛЦ.467200.004 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

2.6. Протокол обміну повідомленнями: MQTT

Опис[12]: MQTT є легким протоколом обміну повідомленнями за принципом "видавець-підписник", розробленим спеціально для IoT-пристроїв з обмеженими ресурсами та нестабільним з'єднанням.

Принцип роботи: Пристрої підключаються до центрального сервера. Видавці публікують повідомлення на певні теми, а підписники отримують повідомлення з тем, на які вони підписалися.

Чому обрано MQTT:

Легкість та низьке енергоспоживання: Ідеально підходить для мікроконтролерів з обмеженими ресурсами.

Надійність: Забезпечує різні рівні якості обслуговування (QoS) для гарантування доставки повідомлень.

Гнучка архітектура: Дозволяє легко додавати нові сенсори та сервіси до системи.

Широка підтримка: Існує велика кількість клієнтських бібліотек для різних платформ та мов програмування.

Порівняння з альтернативами:

HTTP: Більш "важкий" протокол, менш ефективний для постійного обміну невеликими обсягами даних, характерних для IoT.

CoAP: Ще один протокол для обмежених ресурсних середовищ, але MQTT має ширшу підтримку та кращу підтримку QoS.

2.7. Інструмент візуалізації: Node-RED

Опис[2]: Node-RED є візуальним інструментом програмування на основі потоків, що дозволяє легко створювати логіку обробки даних та інтерфейси користувача для IoT-застосувань за допомогою графічного редактора.

					ІАЛЦ.467200.004 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Принцип роботи: Користувач з'єднує функціональні вузли для створення потоків даних, які можуть виконувати різні завдання, такі як підключення до MQTT-серверів, обробка даних, візуалізація та інтеграція з іншими сервісами.

Чому обрано Node-RED:

Візуальне програмування: Спрощує процес розробки, особливо для створення інтерфейсу користувача.

Велика кількість готових вузлів: Існує безліч вузлів для MQTT, візуалізації, обробки даних та інших завдань.

Швидке прототипування: Дозволяє швидко створювати та розгорнути інтерфейси візуалізації.

Вбудована підтримка дашбордів: Легке створення вебінтерфейсів для відображення даних сенсорів.

Порівняння з альтернативами:

Grafana: Потужний інструмент для візуалізації часових рядів даних, але може бути складнішим у налаштуванні для простих IoT-дашбордів порівняно з Node-RED.

Хмарні IoT-платформи (наприклад, Ubidots, ThingsBoard): Надають комплексні рішення, але можуть мати обмеження у безкоштовних планах або вимагати постійного підключення до Інтернету. Node-RED забезпечує більшу гнучкість та локальний контроль.

					ІАЛЦ.467200.004 ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

3. РОЗРОБКА ІОТ-СИСТЕМИ РОЗУМНОГО ДОМУ

3.1. Програмування мікроконтролера ESP32 у середовищі Wokwi

3.1.1. Налаштування середовища Wokwi

Першим кроком у розробці проєкту стало налаштування віртуального середовища Wokwi. Для цього необхідно перейти на вебсайт Wokwi та створити новий проєкт. Після створення проєкту користувачеві надається доступ до віртуального робочого простору, що складається з редактора схем та редактора коду.

Файлова структура проєкту Wokwi включає щонайменше два основних файли:

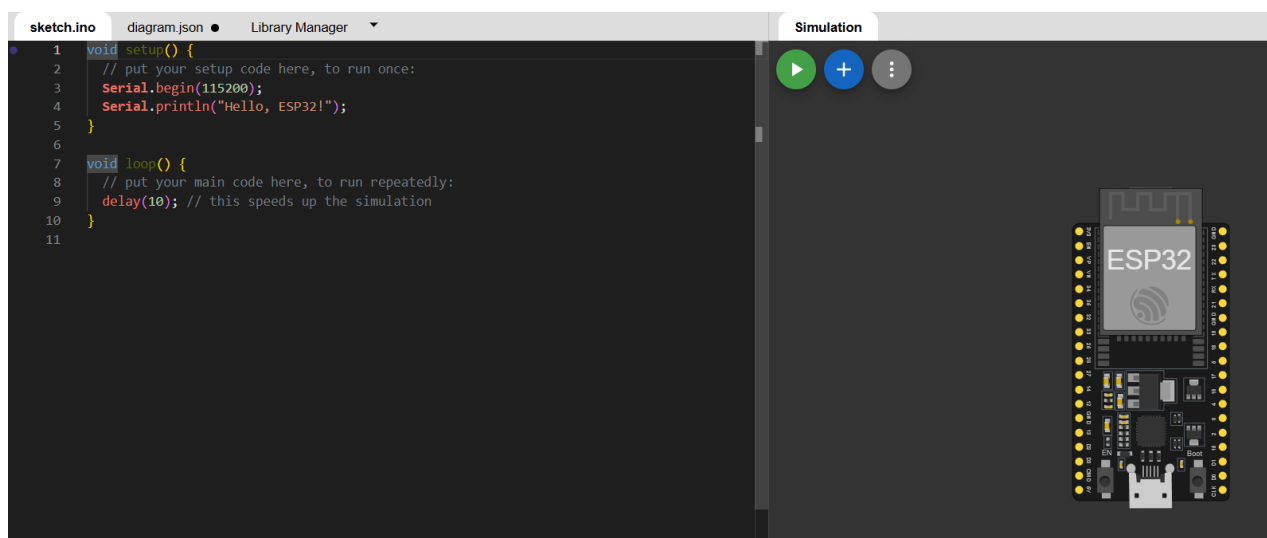


Рисунок 3.1-Новий проєкт на Wokwi

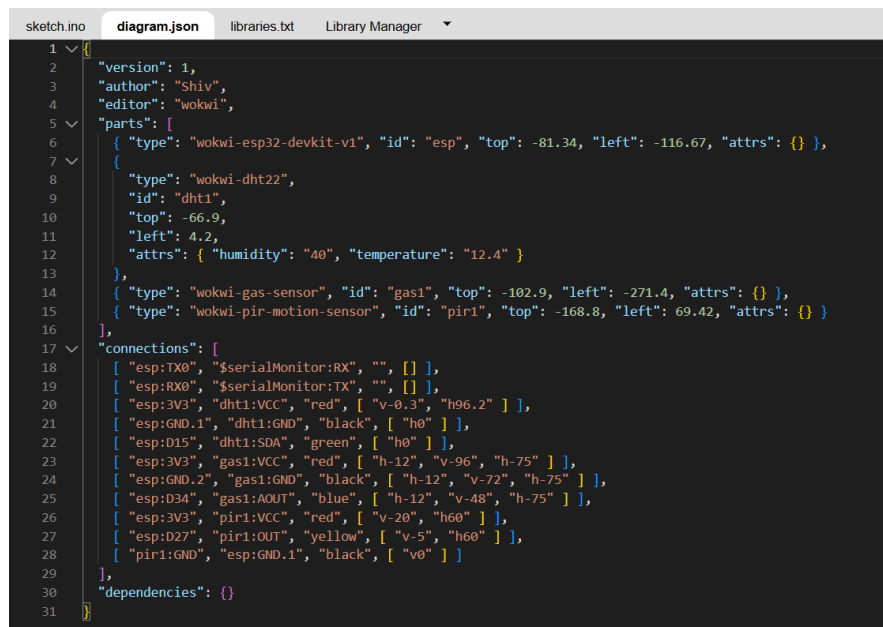
sketch.ino: Це основний файл проєкту Arduino, в якому міститься програмний код для мікроконтролера ESP32, написаний на мові C/C++. Саме в цьому файлі реалізується логіка роботи системи, включаючи ініціалізацію компонентів, зчитування даних з сенсорів, встановлення мережевого з'єднання, обмін даними через протокол MQTT та збереження локальних даних.

diagram.json: Цей файл у форматі JSON описує віртуальну електронну схему проекту. Він містить інформацію про використовувані електронні компоненти, їхні властивості та з'єднання між ними. Редагування цього файлу дозволяє додавати до віртуальної схеми мікроконтролер ESP32, сенсори (DHT22, газовий датчик, PIR-сенсор), а також інші необхідні елементи, такі як резистори, конденсатори тощо.

Налаштування емулятора у Wokwi передбачає:

Вибір плати ESP32: У файлі diagram.json необхідно вказати модель використовуваної плати ESP32. Wokwi підтримує різні варіанти ESP32, що може впливати на доступні пini та деякі внутрішні характеристики.

Конфігурація компонентів у diagram.json: Для кожного віртуального компонента, доданого на схему, у файлі diagram.json визначаються його тип, назва (ID) та параметри підключення до pinів мікроконтролера ESP32. Наприклад, для підключення датчика DHT22 необхідно вказати, до якого пина ESP32 під'єднано його вихід даних. Аналогічно налаштовуються підключення газового та PIR-сенсорів. Для газового сенсора, що є аналоговим, вказується підключення до одного з аналогових входів ESP32. Для PIR-сенсора - до цифрового входу.



```
1 {
2   "version": 1,
3   "author": "Shiv",
4   "editor": "wokwi",
5   "parts": [
6     { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": -81.34, "left": -116.67, "attrs": {} },
7     {
8       "type": "wokwi-dht22",
9       "id": "dht1",
10      "top": -66.9,
11      "left": 4.2,
12      "attrs": { "humidity": "40", "temperature": "12.4" }
13    },
14    { "type": "wokwi-gas-sensor", "id": "gas1", "top": -102.9, "left": -271.4, "attrs": {} },
15    { "type": "wokwi-pir-motion-sensor", "id": "pir1", "top": -168.8, "left": 69.42, "attrs": {} }
16  ],
17  "connections": [
18    [ "esp:TX0", "$serialMonitor:RX", "", [ ] ],
19    [ "esp:RX0", "$serialMonitor:TX", "", [ ] ],
20    [ "esp:3V3", "dht1:VCC", "red", [ "v-0.3", "h96.2" ] ],
21    [ "esp:GND.1", "dht1:GND", "black", [ "h0" ] ],
22    [ "esp:D15", "dht1:SDA", "green", [ "h0" ] ],
23    [ "esp:3V3", "gas1:VCC", "red", [ "h-12", "v-96", "h-75" ] ],
24    [ "esp:GND.2", "gas1:GND", "black", [ "h-12", "v-72", "h-75" ] ],
25    [ "esp:D34", "gas1:AOUT", "blue", [ "h-12", "v-48", "h-75" ] ],
26    [ "esp:3V3", "pir1:VCC", "red", [ "v-20", "h60" ] ],
27    [ "esp:D27", "pir1:OUT", "yellow", [ "v-5", "h60" ] ],
28    [ "pir1:GND", "esp:GND.1", "black", [ "v0" ] ]
29  ],
30  "dependencies": {}
31 }
```

Рисунок 3.2- Файл diagram.json з підключеними елементами

Після створення проєкту та налаштування віртуальної схеми у файлі `diagram.json` можна переходити до написання коду у файлі `sketch.ino`.

3.1.2. Підключення бібліотек

Для реалізації функціональності IoT-системи на мікроконтролері ESP32 необхідно використовувати спеціалізовані бібліотеки, які спрощують роботу з різними протоколами та периферійними пристроями. У даному проєкті використовуються наступні бібліотеки:

WiFi.h: Ця стандартна бібліотека Arduino ESP32 Core надає функціональність для встановлення бездротового з'єднання з мережею Wi-Fi. Вона містить класи та методи для сканування доступних мереж, підключення до заданої мережі за її SSID та паролем, отримання IP-адреси та керування Wi-Fi інтерфейсом ESP32. Використання цієї бібліотеки є необхідним для підключення пристрою до локальної мережі та подальшої взаємодії з MQTT-сервером через Інтернет.

PubSubClient.h: Ця популярна стороння бібліотека реалізує клієнтський функціонал протоколу MQTT. Бібліотека PubSubClient.h надає методи для підключення до MQTT-сервера, публікації повідомлень у заданих темах, підписки на цікаві теми та обробки отриманих повідомлень через callback-функцію.

DHTesp.h: Ця бібліотека розроблена спеціально для роботи з датчиками температури та вологості серії DHT (включаючи DHT11 та DHT22) на платформі ESP32. Вона надає зручні функції для ініціалізації датчика, зчитування значень температури та вологості, а також обробки можливих помилок при зчитуванні даних. Використання цієї бібліотеки значно спрощує процес отримання даних з датчика DHT22 порівняно з написанням коду для безпосередньої взаємодії з його інтерфейсом.

Preferences.h: Ця бібліотека, що також є частиною Arduino ESP32 Core, надає простий спосіб для збереження невеликих обсягів даних в

					ІАЛЦ.467200.004 ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

енергонезалежній пам'яті (NVS - Non-Volatile Storage) мікроконтролера ESP32. Це дозволяє зберігати важливі конфігураційні параметри або останні значення датчиків між перезавантаженнями пристрою. Бібліотека Preferences.h оперує даними за принципом ключ-значення та підтримує різні типи даних (цілі числа, числа з плаваючою комою, рядки).

Додавання бібліотек до проєкту Wokwi може бути здійснено двома основними способами:

Через Library Manager: Wokwi має вбудований менеджер бібліотек, який дозволяє знаходити та додавати популярні бібліотеки Arduino безпосередньо в онлайн-середовищі. Для цього зазвичай потрібно перейти до редактора коду та скористатися відповідною опцією в меню або контекстному меню. Цей спосіб був використаний в моїй роботі.

Шляхом копіювання файлів: У деяких випадках, якщо бібліотека не доступна через менеджер, її файли можна скопіювати у відповідну директорію проєкту Wokwi.

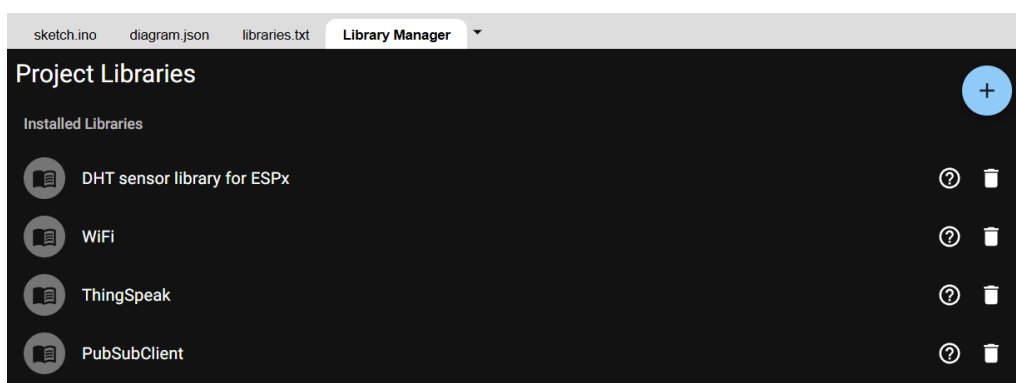


Рисунок 3.3 –Менеджер бібліотек

Після успішного додавання необхідних бібліотек їх можна використовувати у кодї програми за допомогою директиви `#include <назва_бібліотеки.h>`.

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

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

					ІАЛЦ.467200.004 ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

```
#include <DHTesp.h>
#include <Preferences.h>
```

3.1.3. Конфігурація WiFi та MQTT-клієнта

Для забезпечення підключення ESP32 до мережі WiFi та його взаємодії з MQTT-сервером у кодї програми визначено ряд важливих змінних та об'єктів:

`const char* ssid = "Wokwi-GUEST";` та `const char* password = "";`; Ці константні символічні вказівники (`const char*`) зберігають ідентифікатор (SSID) та пароль бездротової мережі Wi-Fi, до якої буде підключатися ESP32. У середовищі Wokwi зазвичай використовується мережа "Wokwi-GUEST" без пароля. У реальному застосуванні ці значення необхідно буде змінити на параметри власної WiFi мережі.

`const char* mqtt_server = "test.mosquitto.org";`; Цей константний символічний вказівник містить доменне ім'я або IP-адресу MQTT-сервера, з яким буде встановлюватися з'єднання для обміну повідомленнями. У даному кодї використовується публічний тестовий MQTT-сервер `test.mosquitto.org`. У реальних проєктах може використовуватися власний локальний сервер або хмарний сервіс MQTT.

Порт MQTT: Протокол MQTT за замовчуванням використовує TCP-порт 1883 для встановлення з'єднання між клієнтом та сервером. Цей порт вказується при налаштуванні з'єднання з сервером у кодї програми (хоча явно не визначений як константа, він використовується у функції `client.setServer()`).

`WiFiClient espClient;`; Цей об'єкт класу `WiFiClient` (з бібліотеки `WiFi.h`) відповідає за встановлення та підтримку TCP-з'єднання, яке є основою для роботи протоколу MQTT. Він створює клієнтський сокет для взаємодії з мережею Wi-Fi.

`PubSubClient client(espClient);`; Цей об'єкт класу `PubSubClient` (з бібліотеки `PubSubClient.h`) представляє MQTT-клієнта. При його створенні йому передається об'єкт `espClient`, що встановлює зв'язок між MQTT-клієнтом та

					ІАЛЦ.467200.004 ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

мережевим з'єднанням. Об'єкт `client` надає методи для виконання основних операцій MQTT, таких як підключення до сервера, публікація повідомлень, підписка на теми та встановлення `callback`-функції для обробки вхідних повідомлень.

Налаштування цих параметрів є критично важливим для успішного підключення ESP32 до мережі WiFi та встановлення зв'язку з MQTT-сервером, що є основою для передачі даних від сенсорів та отримання команд керування в IoT-системі розумного дому.

3.1.4. Реалізація підключення до сервера та обробка з'єднання

У коді програми реалізовано дві основні функції, відповідальні за встановлення мережевого з'єднання та підключення до MQTT-сервера: `setup_wifi()` та `reconnect()`.

Детальний розбір функції `setup_wifi()`:

`WiFi.mode(WIFI_STA);` Ця функція відповідає за ініціалізацію WiFi модуля ESP32 та підключення до вказаної бездротової мережі.

`delay(10);` Невелика затримка на початку функції для стабілізації роботи.

`Serial.begin(115200);` Ініціалізація послідовного порту на швидкості 115200 бод для виведення налагоджувальної інформації.

`Serial.println();` та `Serial.print("Connecting to "); Serial.println(ssid);` Виведення повідомлень у послідовний порт про початок процесу підключення та назву мережі, до якої відбувається підключення.

`WiFi.mode(WIFI_STA);` Встановлення режиму роботи WiFi модуля як станції (клієнта), що дозволяє йому підключатися до точки доступу (роутера).

`WiFi.begin(ssid, password);` Запуск процесу підключення до бездротової мережі з вказаними SSID та паролем.

`while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print("."); }` Цикл очікування встановлення WiFi з'єднання. У цьому циклі програма періодично перевіряє статус підключення (`WiFi.status()`). Доки підключення не

					ІАЛЦ.467200.004 ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

буде успішним (WL_CONNECTED), програма чекає 500 мілісекунд та виводить крапку в послідовний порт, інформуючи про триваючий процес підключення.

randomSeed(micros());: Ініціалізація генератора випадкових чисел на основі значення мікросекундного таймера. Це використовується для генерації унікального Client ID для MQTT-клієнта.

Serial.println(""); Serial.println("WiFi connected"); Serial.println("IP address:"); Serial.println(WiFi.localIP());: Після успішного підключення до WiFi мережі у послідовний порт виводяться повідомлення про успішне підключення та отриману ESP32 IP-адресу.

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

Рисунок 3.4-Повідомлення про успішне підключення

Детальний розбір функції reconnect():

Ця функція відповідає за встановлення або відновлення з'єднання з MQTT-сервером у випадку його втрати. Вона викликається у головному циклі loop(), якщо MQTT-клієнт не підключений (!client.connected()).

while (!client.connected()) { ... }: Цикл спроб підключення до MQTT-сервера. Цей цикл продовжується до тих пір, доки не буде встановлено успішне з'єднання з сервером.

Serial.print("Attempting MQTT connection...");: Виведення повідомлення у послідовний порт про спробу підключення до MQTT-сервера.

String clientId = "ESP32Client-" + String(random(0xffff), HEX);: Генерація унікального Client ID для MQTT-клієнта. Це важливо для ідентифікації клієнта

на сервері, особливо при одночасному підключенні кількох пристроїв. Використовується префікс "ESP32Client-" та випадкове шістнадцяткове число.

`if (client.connect(clientId.c_str())) { ... }`: Спроба підключення до MQTT-сервера за допомогою згенерованого Client ID. Функція `client.connect()` повертає `true` у разі успішного підключення та `false` у разі невдачі.

`Serial.println("Connected");`: У разі успішного підключення виводиться повідомлення про це у послідовний порт.

`client.publish("/ThinkIOT/Publish", "Welcome");`: Публікація вітального повідомлення на тему `/ThinkIOT/Publish`. Це може бути використано для інформування інших підписників про підключення нового пристрою.

`client.subscribe("/ThinkIOT/Subscribe");`: Підписка на тему `/ThinkIOT/Subscribe`. ESP32 буде отримувати всі повідомлення, опубліковані в цій темі. Це дозволяє пристрою отримувати команди керування або іншу інформацію від сервера.

`else { ... }`: У разі невдалої спроби підключення:

`Serial.print("failed, rc="); Serial.print(client.state()); Serial.println(" try again in 5 seconds");`: Виведення повідомлення про невдачу, коду помилки (`client.state()`) та інформації про повторну спробу підключення через 5 секунд.

`delay(5000);`: Затримка на 5 секунд перед наступною спробою підключення до сервера.

Ці механізми забезпечення WiFi підключення та автоматичного перепідключення до MQTT-сервера є критично важливими для стабільної та надійної роботи IoT-системи розумного дому.

3.2. Підключення сенсорів температури, вологості, газу та руху

Для забезпечення збору даних про внутрішнє середовище розумного дому до мікроконтролера ESP32 було підключено чотири основні сенсорні компоненти: датчик температури та вологості DHT22, аналоговий газовий

					ІАЛЦ.467200.004 ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

датчик (підключений до піна 34) та пасивний інфрачервоний датчик руху (PIR-сенсор).

3.2.1. Ініціалізація DHT22

У програмному коді для роботи з датчиком DHT22 використовується бібліотека DHTesp.h. На початку коду оголошується константа `const int DHT_PIN = 15;`, яка визначає цифровий пін мікроконтролера ESP32, до якого підключено вихід даних датчика DHT22.

Далі в коді створюється об'єкт класу DHTesp з назвою `dht`. Цей об'єкт використовується для взаємодії з функціоналом бібліотеки.

Ініціалізація датчика DHT22 відбувається у функції `setup()` за допомогою виклику методу `dht.setup(DHT_PIN, DHTesp::DHT22);`. Цей метод приймає два параметри: номер піна ESP32, до якого підключено датчик (15), та тип використовуваного датчика (DHT22). Виклик цієї функції налаштовує необхідні параметри для обміну даними з датчиком DHT22.

3.2.2. Зчитування значення з аналогового газового сенсора MQ-2

Для підключення аналогового газового сенсора у коді визначено константу `const int GAS_SENSOR_PIN = 34;`, яка вказує на аналоговий вхід мікроконтролера ESP32 (пін GPIO34), до якого підключено аналоговий вихід газового сенсора.

Зчитування аналогового значення з газового сенсора відбувається у функції `loop()` за допомогою виклику функції `analogRead(GAS_SENSOR_PIN)`. Функція `analogRead()` є стандартною функцією Arduino для ESP32, яка повертає ціле число в діапазоні від 0 до 4095, що відповідає рівню напруги на аналоговому вході. Отримане значення є відносним показником рівня "газу" або загального забруднення повітря, оскільки для точного визначення концентрації конкретних газів необхідне калібрування датчика.

					ІАЛЦ.467200.004 ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

3.2.3. Обробка цифрового сигналу з PIR-сенсора

Для роботи з PIR-сенсором у кодї визначено константу `const int PIR_SENSOR_PIN = 27;`, яка вказує на цифровий пін мікроконтролера ESP32 (пін GPIO27), до якого підключено вихід сигналу PIR-сенсора.

Для налаштування піна ESP32 як вхід використовується функція `pinMode(PIR_SENSOR_PIN, INPUT);`, яка викликається у функції `setup()`.

Зчитування цифрового стану PIR-сенсора відбувається у функції `loop()` за допомогою виклику функції `digitalRead(PIR_SENSOR_PIN)`. PIR-сенсор повертає цифровий сигнал HIGH при виявленні руху та LOW при його відсутності. Функція `digitalRead()` повертає відповідне значення (1 або 0).

3.2.4. Вивід отриманих даних у консоль

Для налагодження та моніторингу роботи сенсорів у кодї передбачено виведення отриманих даних у послідовний порт за допомогою функцій `Serial.print()` та `Serial.println()`.

```
Temperature: 43.10
Humidity: 72.5
Gas sensor value: 3790
Motion sensor: Motion Detected
Temperature: 43.10
Humidity: 72.5
Gas sensor value: 3790
Motion sensor: No Motion
```

Рисунок 3.5-Вивід отриманих даних у послідовний порт

У функції `loop()` після зчитування значень з кожного сенсора виводяться значення температури, вологості, аналогове значення з газового сенсора та

					ІАЛЦ.467200.004 ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

текстове повідомлення про стан датчика руху. Це дозволяє контролювати роботу сенсорів в режимі реального часу.

3.3. Передача даних через MQTT та обробка повідомлень

Однією з ключових функціональних можливостей розробленої IoT-системи є передача даних, отриманих від сенсорів, на MQTT-сервер для подальшої обробки та візуалізації. Протокол було обрано завдяки його легкості, ефективності та надійності при роботі в умовах обмежених ресурсів та нестабільного з'єднання, що є типовим для багатьох IoT-застосувань.

3.3.1. Формування MQTT-топіків для кожного сенсора

Для організації даних, що передаються від ESP32 до MQTT-сервера, використовуються MQTT-топіки. Топіки являють собою ієрархічні рядки, що використовуються сервером для фільтрації повідомлень, які цікавлять підписаних клієнтів. У даній системі для кожного типу сенсорних даних визначено окремий топик:

/ThinkIOT/temp: Для публікації значень температури.

/ThinkIOT/hum: Для публікації значень вологості.

/ThinkIOT/gas: Для публікації сирого аналогового значення газового сенсора.

/ThinkIOT/motion: Для публікації стану датчика руху (текстове повідомлення "Motion Detected" або "No Motion").

Використання окремих топіків для кожного типу даних дозволяє клієнтам (наприклад, Node-RED) підписуватися лише на ті дані, які їм необхідні, що зменшує мережевий трафік та спрощує обробку. Префікс /ThinkIOT/ використовується як загальний простір імен для даної IoT-системи, що допомагає уникнути конфліктів з іншими можливими MQTT-повідомленнями на сервері.

					ІАЛЦ.467200.004 ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

3.3.2. Публікація даних у вигляді рядків

У головному циклі `loop()` програми ESP32, після зчитування даних з сенсорів, відбувається їхня публікація на відповідні MQTT-топіки за допомогою методу `client.publish()` об'єкта `client` (класу `PubSubClient`).

Значення температури (`tempVal`), отримане з датчика DHT22, форматується до двох знаків після коми та перетворюється на рядок за допомогою функції `String(tempVal, 2)`. Отриманий рядок публікується на топик `/ThinkIOT/temp`.

Значення вологості (`humVal`) форматується до одного знака після коми та також перетворюється на рядок за допомогою `String(humVal, 1)`. Цей рядок публікується на топик `/ThinkIOT/hum`.

Початкове аналогове значення, отримане з газового сенсора (`gasValue`), перетворюється на рядок за допомогою `String(gasValue)` та публікується на топик `/ThinkIOT/gas`.

Стан датчика руху (`motion`), який є цілим числом (0 або 1), використовується для формування текстового повідомлення "Motion Detected" або "No Motion". Цей рядок публікується на топик `/ThinkIOT/motion`.

Метод `client.publish()` приймає два основні аргументи: MQTT-топик, на який необхідно опублікувати повідомлення, та корисне навантаження (`payload`) повідомлення, яке в даному випадку є рядковим представленням сенсорних даних. Перетворення числових значень на рядки спрощує їхню передачу та подальшу обробку на стороні клієнта MQTT (наприклад, у Node-RED).

3.3.3. Створення callback-функції для обробки вхідних повідомлень

Для обробки повідомлень, які MQTT-сервер надсилає ESP32 у відповідь на підписку на певні топіки, у кодї реалізовано callback-функцію `callback(char* topic, byte* payload, unsigned int length)`. Ця функція викликається автоматично

					ІАЛЦ.467200.004 ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

бібліотекою PubSubClient кожного разу, коли ESP32 отримує повідомлення на топик, на який він підписаний.

Функція callback приймає три аргументи:

topic: Вказівник на рядок, що містить назву топика, на який було опубліковано отримане повідомлення.

payload: Вказівник на масив байтів, що представляє корисне навантаження повідомлення.

length: Довжина масиву байтів payload.

У даній реалізації функція callback просто виводить у послідовний порт інформацію про отримане повідомлення, включаючи топик та його вміст перетворений на рядок. У більш складних системах ця функція може містити логіку для обробки отриманих команд керування або конфігураційних даних, які можуть надходити від інших клієнтів MQTT.

Для того, щоб бібліотека PubSubClient могла використовувати цю функцію для обробки вхідних повідомлень, необхідно зареєструвати її за допомогою виклику методу `client.setCallback(callback);` у функції `setup()`.

3.3.4. Автоматичне повторне з'єднання з сервером (reconnect)

Для забезпечення стабільного з'єднання з MQTT-сервером у кодї реалізовано механізм автоматичного повторного підключення у функції `reconnect()`, яка детально розглядалася у попередньому пункті. Ця функція викликається у головному циклі `loop()` кожного разу, коли перевіряється стан з'єднання з сервером за допомогою `!client.connected()`. Якщо з'єднання відсутнє, викликається `reconnect()`, яка намагається встановити нове з'єднання.

Використання автоматичного повторного підключення є важливим для забезпечення надійної роботи IoT-системи, оскільки мережеві з'єднання можуть бути нестабільними. У випадку втрати з'єднання ESP32 автоматично спробує відновити його, що дозволяє уникнути ручного втручання та забезпечити безперервну передачу даних. Також у функції `reconnect()` після успішного

					ІАЛЦ.467200.004 ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

підключення ESP32 знову підписується на необхідні топіки (/ThinkIOT/Subscribe), щоб продовжувати отримувати вхідні повідомлення. Виклик `client.loop()` у головному циклі `loop()` є необхідним для підтримки MQTT-з'єднання, обробки відправлених та отриманих повідомлень, а також для виклику зареєстрованої `callback`-функції при отриманні нових повідомлень.

3.4. Збереження даних у пам'яті ESP32

Для забезпечення збереження важливих даних між перезавантаженнями мікроконтролера ESP32 у IoT-системі була використана бібліотека Preferences. Ця бібліотека дозволяє здійснювати персистентне збереження невеликих обсягів даних в енергонезалежній пам'яті (NVS - Non-Volatile Storage) ESP32.

Першим кроком стало включення бібліотеки до мого проєкту за допомогою директиви `#include <Preferences.h>`. Потім був створив об'єкт класу `Preferences: Preferences prefs;`. Для початку роботи з пам'яттю необхідно відкрити так званий "розділ" (namespace). Виконано це у функції `setup()` за допомогою виклику `prefs.begin("sensors", false);`. Рядок "sensors" є назвою розділу, який був обрано для зберігання даних, пов'язаних із сенсорами. Другий аргумент (`false`) вказує, що режим читання-запису не використовується одночасно в кількох процесах. Це зазвичай не потрібне в простих проєктах з одним мікроконтролером. Використання розділів дозволяє логічно групувати дані та уникати конфліктів імен ключів, якщо мій проєкт у майбутньому стане складнішим і потребуватиме збереження інших типів даних.

Для збереження останніх значень, отриманих від сенсорів, використовувалися відповідні методи об'єкта `prefs` безпосередньо перед кожною публікацією даних на MQTT-сервер у функції `loop()`:

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

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

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

					ІАЛЦ.467200.004 ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

Тут були використані методи `putFloat()` для збереження значень температури (`tempVal`) та вологості (`humVal`) як чисел з плаваючою комою, та метод `putInt()` для збереження сирого значення газового сенсора (`gasValue`) як цілого числа. Кожен метод приймає два аргументи: ключ (рядок, який ідентифікує збережене значення) та саме значення, яке потрібно зберегти. Наприклад, "temp" є ключем для збереженого значення температури. Такий підхід "ключ-значення" є зручним способом організації збережених даних.

Для того щоб відновити збережені значення при кожному запуску ESP32, використано відповідні методи `get()` у функції `setup()`:

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

Кожен метод `get()` приймає два аргументи: ключ значення, яке потрібно отримати, та значення за замовчуванням. Значення за замовчуванням повертається в тому випадку, якщо за вказаним ключем ще не було збережено жодних даних (наприклад, при першому запуску пристрою). Таким чином, якщо дані про температуру ще не були збережені, змінна `lastTemp` отримає значення 0.

Для демонстрації того, що збережені дані успішно відновлюються, було додано виведення цих значень у послідовну консоль у функції `setup()`:

```
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 інструментом для збереження невеликих обсягів конфігураційних даних або останніх станів, вона має певні обмеження. Обсяг пам'яті NVS, доступний для `Preferences`, є відносно невеликим,

і кількість записів також може бути обмежена. Для збереження більших обсягів даних, таких як журнали подій або історичні дані сенсорів, можуть знадобитися альтернативні методи. До таких альтернатив належать використання файлової системи SPIFFS, яка дозволяє працювати з файлами на флеш-пам'яті ESP32, або використання SD-карти, якщо ESP32 має відповідний інтерфейс. Однак для поточного проєкту збереження останніх значень сенсорів за допомогою Preferences виявилось цілком достатнім.

3.5. Використання retained-повідомлень для збереження останнього стану

У протоколі MQTT існує важлива концепція під назвою "retained-повідомлення". Коли клієнт публікує повідомлення з встановленим прапорцем RETAIN, MQTT-сервер зберігає останнє таке повідомлення для відповідної теми. Це означає, що коли новий клієнт підписується на цю тему, сервер негайно надсилає йому це останнє збережене повідомлення.

Використання retained-повідомлень має значні переваги, особливо для систем моніторингу, таких як моя. Нові підписники, щойно підключившись, одразу отримують актуальний стан системи без необхідності чекати на нове оновлення від видавця (в моєму випадку – ESP32). Це особливо корисно для відображення поточних значень датчиків на інтерфейсі користувача. Наприклад, якщо користувач відкриває вебсторінку з дашбордом, він одразу побачить останню відому температуру, вологість тощо, замість того щоб бачити порожні поля до наступного оновлення від ESP32.

У наданому мною коді поки що не реалізовано публікацію retained-повідомлень. Однак для того, щоб це зробити, необхідно скористатися перевантаженою версією функції `client.publish()`, яка приймає додатковий булевий параметр. Встановивши цей параметр у `true`, було вказано серверу зберегти опубліковане повідомлення як retained для відповідної теми:

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

					ІАЛЦ.467200.004 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```
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.

3.6. Інтеграція з Node-RED та створення інтерфейсу візуалізації

Для створення візуального інтерфейсу користувача та обробки даних, що надходять від IoT-системи через MQTT, було застосовано Node-RED – потужний

```
18 Mar 22:36:01 - [info] Node-RED version: v1.0.4
18 Mar 22:36:01 - [info] Node.js version: v12.16.1
18 Mar 22:36:01 - [info] Windows_NT 10.0.18363 x64 LE
18 Mar 22:36:03 - [info] Loading palette nodes
18 Mar 22:36:04 - [info] Settings file : C:\Users\Я\.node-red\settings.js
18 Mar 22:36:04 - [info] Context store : 'default' [module=memory]
18 Mar 22:36:04 - [info] User directory : C:\Users\Я\.node-red
18 Mar 22:36:04 - [warn] Projects disabled : editorTheme.projects.enabled=false
18 Mar 22:36:04 - [info] Flows file : C:\Users\Я\.node-red\flows_WIN-3A6B8USGRNE.json
18 Mar 22:36:04 - [info] Creating new flow file
18 Mar 22:36:04 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----

18 Mar 22:36:04 - [info] Server now running at http://127.0.0.1:1880/
18 Mar 22:36:04 - [info] Starting flows
18 Mar 22:36:04 - [info] Started flows
```

Рисунок 3.6- Встановлення Node-RED

та інтуїтивно зрозумілий інструмент для побудови потоків даних та реалізації IoT-інтерфейсів за допомогою графічного редактора на основі вузлів.

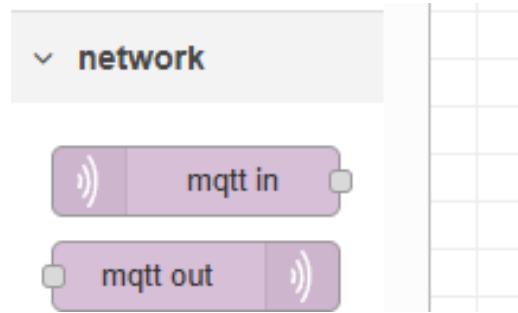


Рисунок 3.7- вузол mqtt in

Першим етапом стала інсталяція Node-RED на локальний комп'ютер. Після запуску було відкрито вебінтерфейс у браузері. Для підключення до MQTT-сервера, на який ESP32 публікує дані, було додано вузол mqtt in — перетягнуто його з палітри вузлів на полотно редактора.

Далі виконано налаштування вузла mqtt in. Подвійне натискання на вузлі відкриває вікно параметрів (рис. 3.8), де вказано:

Server – test.mosquitto.org:1883, тобто публічний MQTT-сервер, який використовується для передачі даних.

- Action – Subscribe to single topic – тип підписки на одну тему.
- Topic – для прикладу, /ThinkIOT/motion – тема, на яку підписується вузол.
- QoS – 0 – мінімальний рівень гарантій доставки, але з найменшою затримкою.
- Output – залишив тип auto-detect.

Аналогічні налаштування виконано для інших MQTT-тем:

- /ThinkIOT/temp – для температури,
- /ThinkIOT/hum – для вологості,
- /ThinkIOT/gas – для даних з газового сенсора,
- /ThinkIOT/motion – для стану сенсора руху.

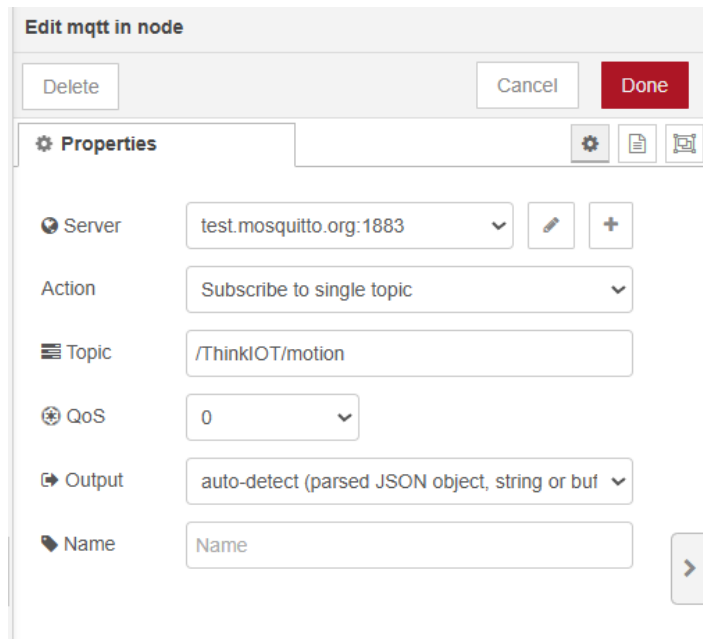


Рисунок 3.8 – Налаштування вузла mqtt in

До кожного з вузлів mqtt in було підключено відповідні вузли з категорії dashboard, призначені для візуального відображення отриманої інформації. Перед цим необхідно було встановити пакет node-red-dashboard.

Для цього у правому верхньому меню інтерфейсу Node-RED обрано пункт Manage palette, на вкладці Install введено dashboard, після чого встановлено відповідний пакет, натиснувши Install.

Після встановлення в палітрі вузлів з'явилася нова категорія dashboard, яка містить вузли для створення інтерфейсу: gauge, text, chart, switch тощо.

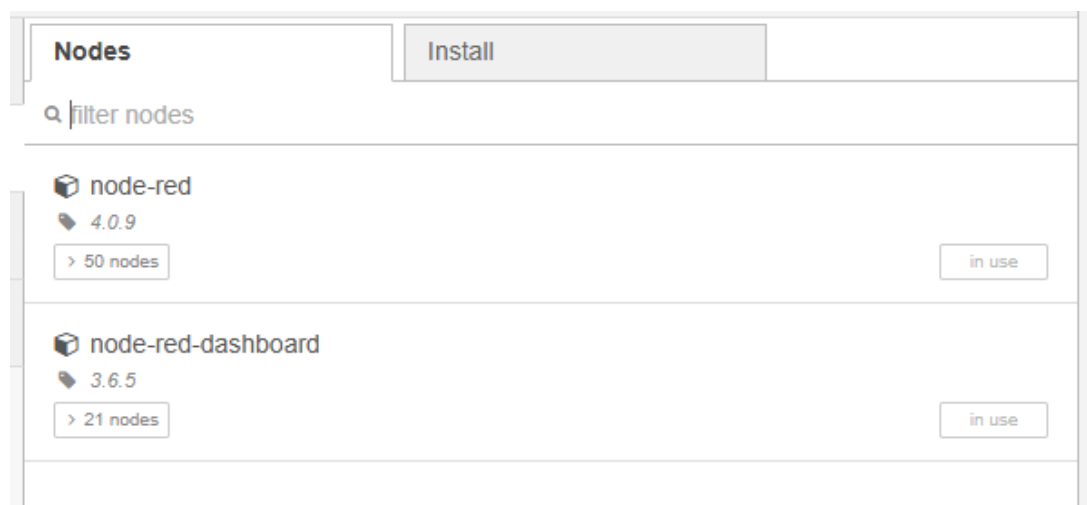


Рисунок 3.9-Встановлений dashboard

Було додано такі елементи візуалізації:

- вузли gauge для температури, вологості та газу;
- вузол text для стану сенсора руху (текстові повідомлення: Motion Detected або No Motion).

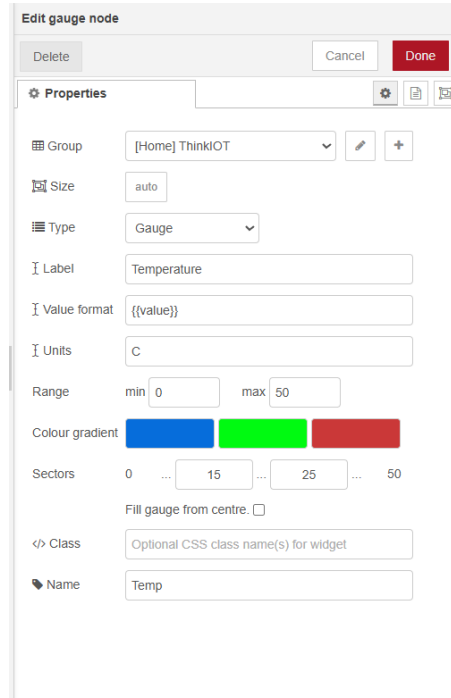


Рисунок 3.10-Налаштування вузла gauge для температури

Параметри вузла gauge для температури:

- Label — Temperature;
- Units — °C;
- Range (min–max) — 0 до 50;
- Group — група ThinkIoT (обрана або створена);
- Color gradient — синій (низько), зелений (нормально), червоний (високо).

Після завершення налаштувань натиснуто кнопку Deploy для активації потоку. Інтерфейс візуалізації став доступним за адресою:

<http://localhost:1880/ui>

Змн.	Арк.	№ докум.	Підпис	Дата

На дашборді Node-RED у режимі реального часу відображаються:

- температура,
- вологість,
- рівень газу,
- стан сенсора руху.

Таке відображення забезпечує зручний моніторинг внутрішнього середовища та створює основу для розширення системи — зокрема, додавання графіків, подій журналу, або елементів керування виконавчими пристроями.

					ІАЛЦ.467200.004 ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

4. ТЕСТУВАННЯ ПРОГРАМИ

4.1. Тестування підключення та передачі даних через WiFi

На цьому етапі було перевірено підключення ESP32 до віртуальної WiFi мережі Wokwi.

Перевірка встановлення WiFi з'єднання: Після запуску симуляції у Wokwi здійснювалося спостереження за вікном серійного монітора для підтвердження успішного підключення до віртуальної WiFi мережі та отримання віртуальної IP-адреси. На рисунку 4.1 відображено скриншот серійного монітора Wokwi з відповідними повідомленнями ("WiFi connected", "IP address: ...").

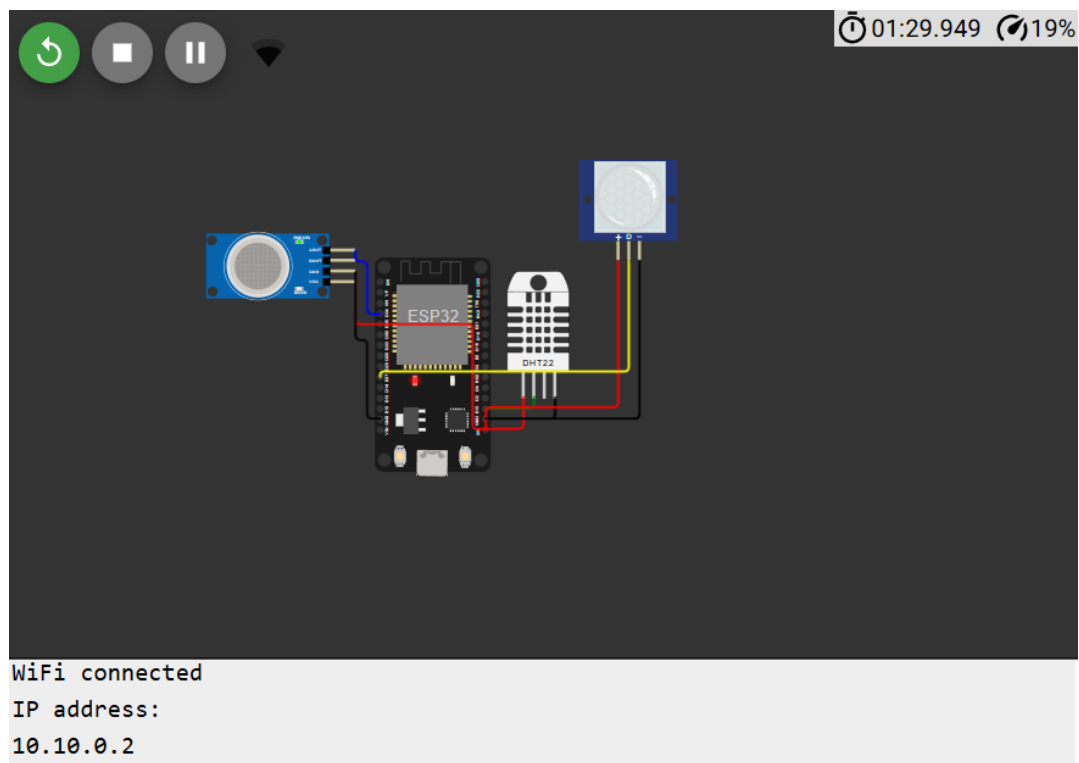


Рисунок 4.1-Успішне встановлення з'єднання

Тестування стабільності віртуального WiFi з'єднання: Протягом певного періоду часу роботи симуляції здійснювалося спостереження за відсутністю повідомлень про розриви та повторні підключення у серійному моніторі.

4.2. Тестування підключення та обміну даними через MQTT

На цьому етапі перевірялася імітація підключення ESP32 до MQTT-сервера та коректність передачі даних від віртуальних сенсорів.

Перевірка підключення до MQTT-сервера: У потоці Node-RED було здійснено візуальну перевірку стану вузлів mqtt in, підписаних на відповідні топіки (/ThinkIOT/temp, /ThinkIOT/hum, /ThinkIOT/gas, /ThinkIOT/motion). На рисунку 4.2 показано скриншот потоку Node-RED з вузлами mqtt in, що відображають статус "connected".

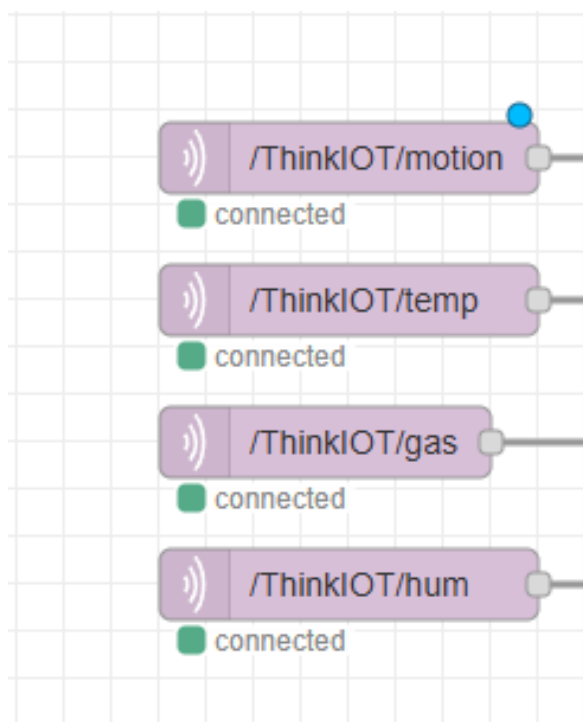


Рисунок 4.2-Вузли mqtt in у стані "connected"

Перевірка публікації даних від віртуального ESP32: За допомогою вузла debug у Node-RED було перевірено вміст MQTT-повідомлень, що надходять на кожен з топіків після зчитування даних з віртуальних сенсорів у Wokwi. На рисунку 4.3 представлено скриншот консолі налагодження Node-RED з прикладами отриманих повідомлень про стан руху.

Змн.	Арк.	№ докум.	Підпис	Дата

```

20.05.2025, 02:20:57 node: debug 1
/ThinkIoT/motion : msg.payload : string[9]
"No Motion"
20.05.2025, 02:21:03 node: debug 1
/ThinkIoT/motion : msg.payload : string[9]
"No Motion"
20.05.2025, 02:21:05 node: debug 1
/ThinkIoT/motion : msg.payload : string[9]
"No Motion"
20.05.2025, 02:21:11 node: debug 1
/ThinkIoT/motion : msg.payload : string[9]
"No Motion"
20.05.2025, 02:21:13 node: debug 1
/ThinkIoT/motion : msg.payload : string[9]
"No Motion"
20.05.2025, 02:21:19 node: debug 1
/ThinkIoT/motion : msg.payload : string[9]
"No Motion"
20.05.2025, 02:21:22 node: debug 1
/ThinkIoT/motion : msg.payload : string[9]
"No Motion"

```

Рисунок 4.3-Отримані повідомлення про стан руху

Тестування коректності передачі віртуальних значень: Порівнювалися значення, які імітувалися для віртуальних сенсорів зі значеннями, отриманими в Node-RED на відповідних віджетах дашборду, для підтвердження їхньої відповідності. На рисунку 4.4 відображено одночасний скриншот Wokwi та дашборду Node-RED з порівнюваними значеннями.

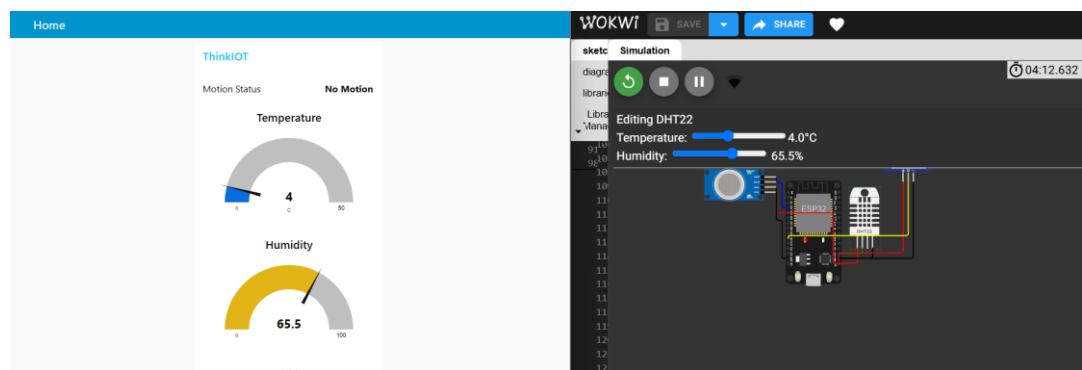


Рисунок 4.4-Порівняння даних з сенсорів та дашборду

4.3. Тестування функціональності віртуальних сенсорів

На цьому етапі перевірялася коректність імітації роботи кожного віртуального сенсора у Wokwi.

Тестування віртуального датчика температури та вологості (DHT22): Змінювалися параметри temperature та humidity у властивостях віртуального компонента DHT22. Спостерігалася відповідна зміна значень на віджетах "Temperature" та "Humidity" на дашборді Node-RED. На рисунку 4сунк.5 показано зміну віджетів на дашборді при зміні параметрів віртуального сенсора у Wokwi.

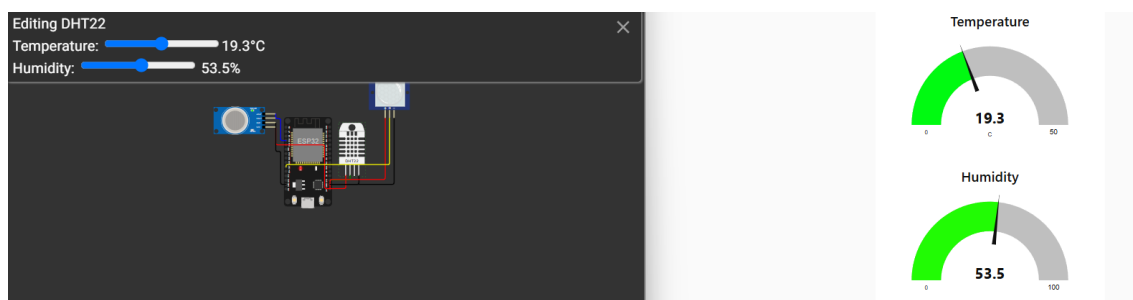


Рисунок 4.5-Зміна даних відносно датчиків температури та вологості

Тестування віртуального газового сенсора: Змінювався параметр, що імітує рівень газу у властивостях віртуального газового сенсора у Wokwi. Фіксували відповідну зміну значення на віджеті "gas" на дашборді

Тестування віртуального PIR-сенсора: При змінювався стану віртуального PIR-сенсора у Wokwi. спостерігалася відповідна зміна стану віджета "Motion Status" на дашборді. На рисунку 4.6 відображено зміну стану віджета "Motion Status" при зміні стану віртуального PIR-сенсора у Wokwi.

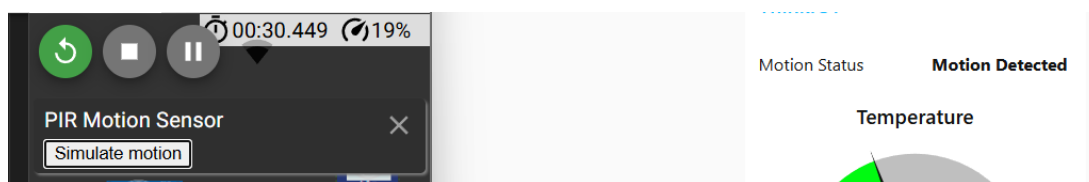


Рисунок 4.6-Зміна стан PIR-сенсора

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було реалізовано прототип IoT-системи для розумного дому на базі ESP32, що здійснює збирання та передачу екологічних даних. У системі використано сенсори температури, вологості, газу та руху.

Для передачі даних було застосовано протокол MQTT, що забезпечив ефективну та надійну комунікацію з сервером. На основі отриманих повідомлень створено візуальний інтерфейс у Node-RED, який дає змогу користувачу спостерігати за показниками в реальному часі.

Платформа Wokwi була використана як основне середовище моделювання, що дозволило швидко реалізувати схему, протестувати код та перевірити логіку взаємодії всіх компонентів.

Крім того, було реалізовано механізм збереження останніх даних у пам'яті ESP32, що підвищує надійність системи у випадках перебоїв з живленням або зв'язком.

Результати роботи свідчать про ефективність використання віртуальних середовищ для проєктування IoT-систем, а також підтверджують доцільність обраної архітектури на основі ESP32, MQTT і Node-RED для побудови прототипів систем розумного дому.

					ІАЛЦ.467200.004 ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. IBM. [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.ibm.com/topics/internet-of-things>.
2. Node-RED. [Електронний ресурс]. – Режим доступу до ресурсу: <https://nodered.org/>.
3. HiveMQ. [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages/>.
4. Tutorialspoint. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.tutorialspoint.com/internet_of_things/internet_of_things_hardware.htm.
5. Espressif Systems. [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.espressif.com/en/products/socs/esp32>.
6. Alldatasheet.com / SparkFun. [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.alldatasheet.com/datasheetpdf/view/1132459/ETC2/DHT22.html>.
7. Seeed Studio. DHT11 vs DHT22 [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.seeedstudio.com/blog/2020/04/20/dht11-vs-dht22-am2302-which-temperature-humidity-sensor-should-you-use/>.
8. MyProject.com.ua. [Електронний ресурс]. – Режим доступу до ресурсу: <https://myproject.com.ua/porivniannia-datchykiv-temperature-ya-volohosti-dht11-vs-dht22-vs-bme280.html>.
9. SparkFun Electronics. MQ-2 Gas Sensor Datasheet. [Електронний ресурс]. – Режим доступу до ресурсу: <https://cdn.sparkfun.com/datasheets/Sensors/Air/MQ-2%20Gas%20Sensor.pdf>.
10. SparkFun Electronics. PIR Motion Sensor Hookup Guide. [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.sparkfun.com/tutorials/pir-motion-sensor-hookup-guide>.

					ІАЛЦ.467200.004 ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

11. Random Nerd Tutorials. ESP32 & ESP8266 Simulator Wokwi. [Електронний ресурс]. – Режим доступу до ресурсу: <https://randomnerdtutorials.com/esp32-esp8266-simulator-wokwi/>.

12. MQTT.org. MQTT Essentials. [Електронний ресурс]. – Режим доступу до ресурсу: <https://mqtt.org/mqtt-essentials/>

					ІАЛЦ.467200.004 ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТОК А. Копії графічного матеріалу