ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:« Розробка IoT системи моніторингу екологічного стану міста з використанням модулю ESP-32»

на здобуття освітнього ступеня бакалавра

зі спеціальності 126 Інформаційні системи та технології

освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій ІЛЮК

(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти
група ІСД-42 Андрій ІЛЮК

Керівник: к.н.т . доцент кафедри СЕНЬКОВ Олег

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти Бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКБ
Каміла СТОРЧАК
“___” _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ілюк Андрій Юрійович
(прізвище, ім’я)

1. Тема кваліфікаційної роботи: «Розробка IoT системи моніторингу екологічного стану міста з використанням модулю ESP-32»

керівник кваліфікаційної роботи Сеньков Олег, к.н.т . доцент кафедри.
(прізвище, ім’я, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 2025 року № ____.

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 31.06.2025 р.

3. Вихідні дані до кваліфікаційної роботи
інформаційні ресурси системи, що потребують захисту: екологічні показники з датчиків, журнал подій та налаштування пристроїв;
вимоги до зберігання інформації: збереження історії вимірювань у базі даних з можливістю фільтрації та перегляду.;
нормативні документи: закони та стандарти щодо охорони довкілля та моніторингу екологічного стану.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Огляд існуючих рішень та створення вимог до системи.
3. Тестування та оцінка ефективності системи.

4. Перелік графічного матеріалу
Презентація PowerPoint.

6. Дата видачі завдання 15.04.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз предметної області та огляд рішень екологічного моніторингу на базі IoT	15.04.2025 р.	
2	Вивчення технічної документації на ESP32 та датчики екологічних параметрів	20.04.2025 р.	
3	Постановка задачі, визначення вимог до IoT-системи, формування технічного завдання	25.04.2025 р.	
4	Розробка архітектури системи: апаратна частина (ESP32), сервер (Flask), клієнтська частина (веб-інтерфейс/мобільний додаток)	05.05.2025 р.	
5	Програмування та тестування роботи сенсорів з ESP32, збирання екологічних даних	15.05.2025 р.	
6	Інтеграція програмного забезпечення: передача даних, обробка та зберігання на сервері	25.05.2025 р.	
7	Завершальне тестування системи, візуалізація даних, підготовка демонстраційних матеріалів	30.05.2025 р.	

Здобувач вищої освіти _____ Андрій ІЛЮК
(підпис) *(ім'я, прізвище)*

Керівник кваліфікаційної роботи _____ Олег СЕНЬКОВ
(підпис) *(ім'я, прізвище)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу

студента **Ілюка Андрія Юрійовича**

на тему: ***«Розробка IoT-системи моніторингу екологічного стану міста з використанням модулю ESP-32»***

Актуальність.

У дипломній роботі розглянуто одна з актуальних проблем сучасного міського середовища — моніторинг екологічного стану з використанням технологій Інтернету речей. Актуальність теми обґрунтована необхідністю створення доступних, масштабованих та автономних систем, які дозволяють здійснювати постійний контроль якості повітря, рівня запиленості та інших параметрів довкілля. Застосування модуля ESP32 та спеціалізованих сенсорів забезпечує високу ефективність розробленої системи в умовах розумного міста.

Позитивні сторони.

Позитивні сторони.

1. Робота містить глибокий аналіз існуючих IoT-рішень та обґрунтований вибір апаратного й програмного забезпечення для реалізації власної системи.
2. У практичній частині представлено повний цикл розробки: від постановки задачі до реалізації клієнт-серверної архітектури, створення веб-інтерфейсу та тестування працездатності системи.
3. Здобувачом вищої освіти продемонстровано високий рівень володіння сучасними технологіями (ESP32, Flask, бази даних, веб-програмування), а також вміння працювати з апаратним і програмним забезпеченням у контексті IoT.
4. Робота написана грамотно, матеріал структурований логічно, між розділами збережено послідовний взаємозв’язок.

Недоліки.

1. В роботі трапляються незначні орфографічні та пунктуаційні помилки.

Ці недоліки не впливають на загальне позитивне враження від бакалаврської кваліфікаційної роботи.

Висновок: бакалаврська кваліфікаційна робота Ілюка Андрія Юрійовича виконана на високому рівні, повністю відповідає вимогам до кваліфікаційних робіт першого (бакалаврського) рівня вищої освіти. Робота заслуговує оцінки **«добре»**, а її автор — на присвоєння кваліфікації **бакалавра з інформаційних систем та технологій**.

ПОДАННЯ

ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ

ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

РЕФЕРАТ

Текстова частина бакалаврської роботи: 57 сторінок, 35 рисунків, 38 джерела.

Об'єкт дослідження – процеси моніторингу екологічного стану міських територій з використанням IoT-технологій.

Предмет дослідження – архітектура, принципи побудови та методи реалізації IoT-систем для збору, обробки та відображення екологічних даних на базі мікроконтролера ESP32.

Мета роботи – розробити та реалізувати ефективну IoT-систему для моніторингу екологічного стану міста, що дозволяє автоматизовано збирати, зберігати та візуалізувати дані про параметри довкілля, а також провести оцінку її ефективності та працездатності.

Методи дослідження:

1. Аналіз сучасних IoT-рішень для екологічного моніторингу – вивчення типових архітектур, апаратних засобів та методів передавання даних.
2. Вибір і обґрунтування апаратного забезпечення – підбір датчиків для фіксації параметрів довкілля та модуля ESP32.
3. Проєктування архітектури програмного забезпечення – розробка серверної та клієнтської частини, організація бази даних для зберігання екологічних показників.
4. Тестування та оцінка ефективності системи – перевірка роботи системи у тестових умовах, оцінка стабільності передачі даних та точності збору інформації.

Галузь використання – екологічний моніторинг міських територій та навколишнього середовища.

Ключові слова: ІОТ, ЕКОЛОГІЧНИЙ МОНІТОРИНГ, ESP32, ДАТЧИКИ ДОВКІЛЛЯ, РОЗУМНЕ МІСТО, АВТОМАТИЗОВАНИЙ ЗБІР ДАНИХ, СИСТЕМА КОНТРОЛЮ СТАНУ ДОВКІЛЛЯ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА.

ЗМІСТ

ВСТУП

Актуальність дослідження. Сучасні міста стикаються з численними екологічними викликами: забруднення повітря, зміни мікроклімату, підвищення рівня вуглекислого газу тощо. Ефективний моніторинг стану довкілля є важливою умовою забезпечення якості життя населення та своєчасного реагування на потенційні загрози. Водночас традиційні системи контролю часто є дорогими, маломасштабними та не забезпечують оперативного збору даних у реальному часі.

Застосування технологій Інтернету речей (IoT) відкриває нові можливості для побудови гнучких, масштабованих і економічно ефективних систем екологічного моніторингу. Мікроконтролер ESP32, як основа таких систем, поєднує в собі низьке енергоспоживання, широкі можливості бездротового зв'язку та підтримку численних сенсорів, що робить його ідеальним для реалізації автономних IoT-рішень.

Розробка IoT-системи моніторингу екологічного стану з використанням ESP32 є актуальною через необхідність створення доступних інструментів для постійного спостереження за параметрами довкілля, що дозволяє формувати екологічно обґрунтовані управлінські рішення на рівні міста.

Об'єкт дослідження – процеси моніторингу екологічного стану міських територій з використанням IoT-технологій.

Предмет дослідження – архітектура, принципи побудови та методи реалізації IoT-систем для збору, обробки та відображення екологічних даних на базі мікроконтролера ESP32.

Мета роботи – розробити та реалізувати ефективну IoT-систему для моніторингу екологічного стану міста, що дозволяє автоматизовано збирати, зберігати та візуалізувати дані про параметри довкілля, а також провести оцінку її ефективності та працездатності.

Наукові завдання:

– проаналізувати сучасні підходи до екологічного моніторингу із застосуванням IoT-технологій;

– вивчити технічні можливості модуля ESP32 та сумісних сенсорів для вимірювання параметрів довкілля;

– сформувати вимоги до архітектури IoT-системи з урахуванням завдань екологічного моніторингу в умовах міста;

– розробити апаратну частину системи з використанням мікроконтролера ESP32 та відповідних датчиків;

– реалізувати серверну частину для збору, обробки та збереження екологічних даних;

– створити інтерфейс користувача для візуалізації показників екологічного стану;

– провести тестування системи та оцінити її ефективність.

Методи дослідження:

5. Аналіз сучасних IoT-рішень для екологічного моніторингу – вивчення типових архітектур, апаратних засобів та методів передавання даних.

6. Вибір і обґрунтування апаратного забезпечення – підбір датчиків для фіксації параметрів довкілля та модуля ESP32.

7. Проєктування архітектури програмного забезпечення – розробка серверної та клієнтської частини, організація бази даних для зберігання екологічних показників.

8. Тестування та оцінка ефективності системи – перевірка роботи системи у тестових умовах, оцінка стабільності передачі даних та точності збору інформації.

Розроблена IoT-система моніторингу екологічного стану міста на базі модуля ESP32 забезпечує ефективний та доступний інструмент для збору, зберігання та візуалізації даних про стан довкілля в режимі реального часу. Система продемонструвала можливість побудови масштабованого рішення, яке може бути застосоване для контролю якості повітря, мікроклімату, рівня забруднень та інших параметрів в міських умовах.

Застосування модуля ESP32 та сумісних сенсорів дозволяє реалізувати автономну, енергоефективну та недорогу платформу, яку можна розгортати на

різних територіях із мінімальними витратами. Веб-інтерфейс системи забезпечує зручний доступ до даних як для фахівців, так і для широкого кола користувачів, зокрема муніципальних служб, екологів та громадськості.

Результати роботи можуть бути використані у практичних проєктах екологічного моніторингу, у навчальному процесі для ознайомлення студентів з принципами побудови IoT-систем, а також у дослідженнях, пов’язаних з розвитком концепції "розумного міста".

Апробація. Основні положення і результати бакалаврської роботи доповідались на:

- VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», ДУІКТ, 2025.

1 ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМ ВІДДАЛЕНОГО КЕРУВАННЯ НА БАЗІ ESP-32

1.1 Загальний огляд систем моніторингу екологічного стану

Системи моніторингу довкілля — це сукупність технічних, програмних і організаційних засобів, призначених для безперервного або періодичного збору, аналізу та обробки даних про стан навколишнього середовища. Основною метою таких систем є своєчасне виявлення змін у довкіллі, оцінка рівня забруднення та надання інформації для прийняття управлінських рішень, спрямованих на захист екосистем та забезпечення безпеки населення [1].

Призначення систем моніторингу довкілля:

– Виявлення небезпечних змін у стані атмосфери, водних ресурсів, ґрунтів тощо.

– Попередження екологічних катастроф шляхом раннього виявлення критичних рівнів забруднення.

– Забезпечення органів влади, підприємств і громадськості достовірною інформацією про стан навколишнього середовища.

– Формування баз даних для проведення наукових досліджень та екологічного прогнозування.

– Підтримка процесів сталого розвитку через контроль екологічних показників.

Загальні принципи роботи систем моніторингу довкілля:

Збір даних — здійснюється за допомогою спеціалізованих сенсорів і приладів, які вимірюють різні параметри довкілля (температура, вологість, рівень забруднюючих речовин, шум, випромінювання тощо).

Передача даних — отримана інформація передається у центр обробки даних через дротові або бездротові канали зв'язку (Wi-Fi, LoRa, GSM, супутниковий зв'язок).

Обробка даних — дані обробляються для виявлення відхилень, трендів,

критичних значень. Можлива попередня фільтрація та агрегація інформації.

Збереження даних — інформація зберігається в базах даних для подальшого аналізу та створення історичних звітів.

Візуалізація результатів — результати моніторингу відображаються у вигляді графіків, карт забруднення, таблиць та аналітичних звітів для зручного сприйняття.

Реагування — при виявленні перевищення критичних рівнів система може автоматично надсилати сповіщення або активувати відповідні протоколи дій.

Системи моніторингу довкілля дозволяють реалізувати комплексний підхід до управління екологічною безпекою, сприяючи збереженню природних ресурсів та підвищенню якості життя населення [2].

Рисунок 1.1 – Загальна схема роботи системи моніторингу довкілля

У сучасних системах екологічного моніторингу основну увагу приділяють вимірюванню ряду важливих параметрів навколишнього середовища. Найбільш поширеними серед них є вміст вуглекислого газу (CO_2), концентрація зважених часток у повітрі (пилу), температура та вологість.

Вуглекислий газ (CO_2) – це індикатор, якості повітря. Його надмірна концентрація свідчить про підвищений рівень забруднення, недостатню вентиляцію або інтенсивний трафік. Вимірювання рівня CO_2 дозволяє оцінити стан атмосферного повітря та своєчасно реагувати на потенційні екологічні загрози.

Концентрація пилу, зокрема дрібнодисперсних часток PM1.0, PM2.5 та PM10, є ще одним важливим параметром. Такі частинки утворюються внаслідок транспортних викидів, промислової діяльності або спалювання палива. Наявність великої кількості пилу в повітрі негативно впливає на здоров’я людини, викликаючи респіраторні та серцево-судинні захворювання [3].

Температура повітря є базовим показником стану довкілля. Вона використовується не лише для оцінки кліматичних умов, а й для розрахунку супутніх показників, таких як точка роси або індекс теплового комфорту. Температурні коливання мають суттєвий вплив на динаміку атмосферних процесів.

Вологість повітря визначає комфортність перебування людини в середовищі, а також впливає на розповсюдження пилу, патогенів та процеси очищення повітря. Показники вологості, разом із температурою, формують уявлення про мікроклімат досліджуваної території.

Окрім зазначених параметрів, залежно від специфіки системи, також можуть вимірюватися рівні оксиду вуглецю (CO), діоксиду азоту (NO_2), озону (O_3), атмосферного тиску, рівня шуму та ультрафіолетового випромінювання. Використання комплексного підходу до моніторингу дозволяє отримати повну картину стану навколишнього середовища та приймати обґрунтовані рішення щодо його захисту.

1.2 Порівняння існуючих IoT-рішень для екологічного моніторингу

З огляду на стрімкий розвиток технологій Інтернету речей (IoT), сьогодні існує велика кількість рішень для екологічного моніторингу, що відрізняються між собою архітектурою, технічними характеристиками та функціональними можливостями. Порівняння таких рішень дозволяє виявити сильні та слабкі сторони різних підходів, оцінити їх відповідність сучасним вимогам до систем збору та обробки екологічних даних, а також визначити оптимальні технології для розробки власної IoT-системи. Аналіз існуючих платформ і проєктів забезпечує основу для обґрунтованого вибору апаратних і програмних компонентів, що сприятиме підвищенню ефективності та надійності розробленого рішення [4].

AirVisual Node — це портативний пристрій для моніторингу якості повітря, який вимірює рівень CO_2, дрібнодисперсних часток PM2.5, температуру та вологість. Пристрій обладнаний екраном для локальної візуалізації даних і можливістю передавання інформації у хмару через Wi-Fi. Основними перевагами є висока точність сенсорів і зручний інтерфейс користувача, проте його вартість залишається досить високою, що обмежує масове використання в громадських моніторингових мережах [4].

Рисунок 1.1 – Пристрій для моніторингу AirVisual Node

OpenSenseMap — це відкрита онлайн-платформа для публікації даних з екологічних сенсорів, яку можуть використовувати як окремі користувачі, так і дослідницькі організації. Платформа підтримує інтеграцію з різними сенсорними системами на базі Arduino, ESP8266 та інших мікроконтролерів [5]. Відкрита архітектура та гнучкість у виборі сенсорів роблять OpenSenseMap привабливим рішенням для реалізації власних проєктів, однак надійність даних залежить від якості обладнання користувачів.

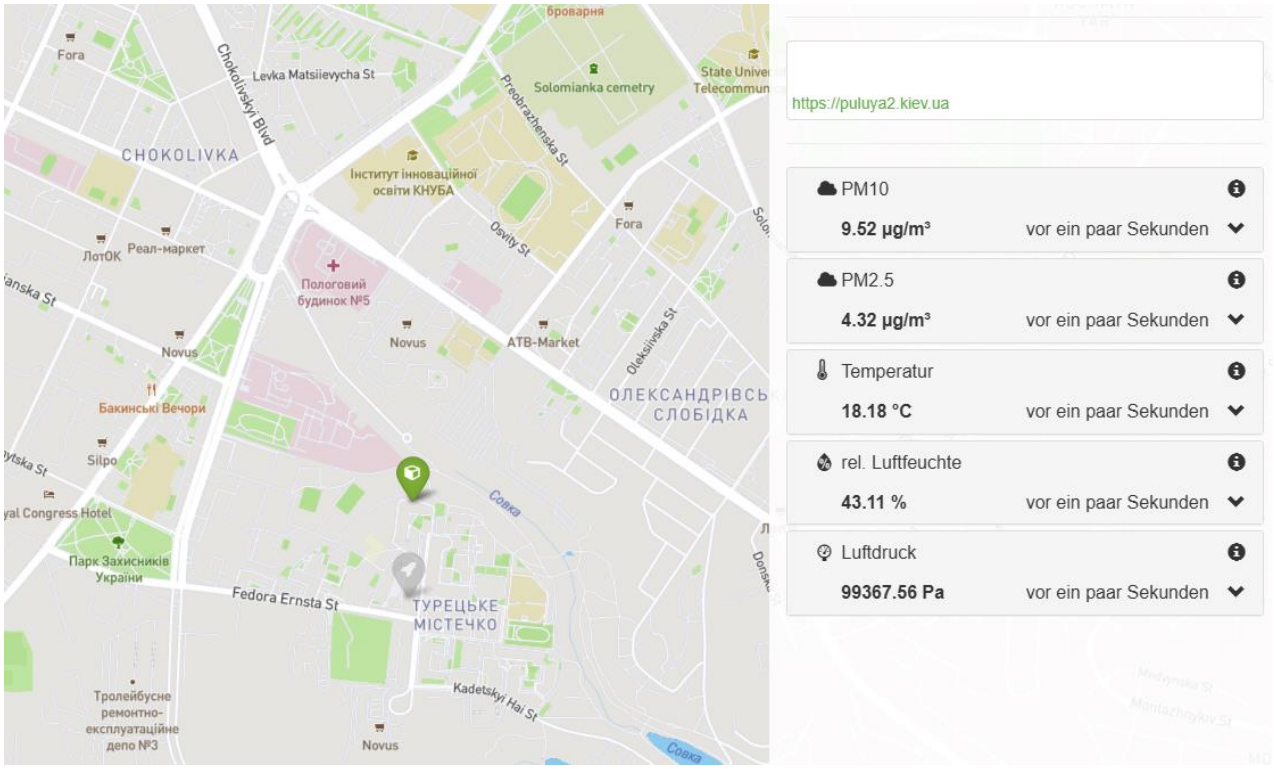


Рисунок 1.2 – OpenSenseMap відкрита онлайн-платформа

Smart Citizen Kit — це комплексне рішення для моніторингу навколишнього середовища, яке включає набір сенсорів для вимірювання температури, вологості, рівня шуму, забруднення повітря та освітленості. Пристрій підтримує бездротову передачу даних через Wi-Fi і має інтеграцію з відкритою платформою для візуалізації результатів. Перевагами Smart Citizen є модульність, простота встановлення та активна підтримка спільноти користувачів, хоча сам пристрій має обмеження за часом автономної роботи без зовнішнього живлення [6].

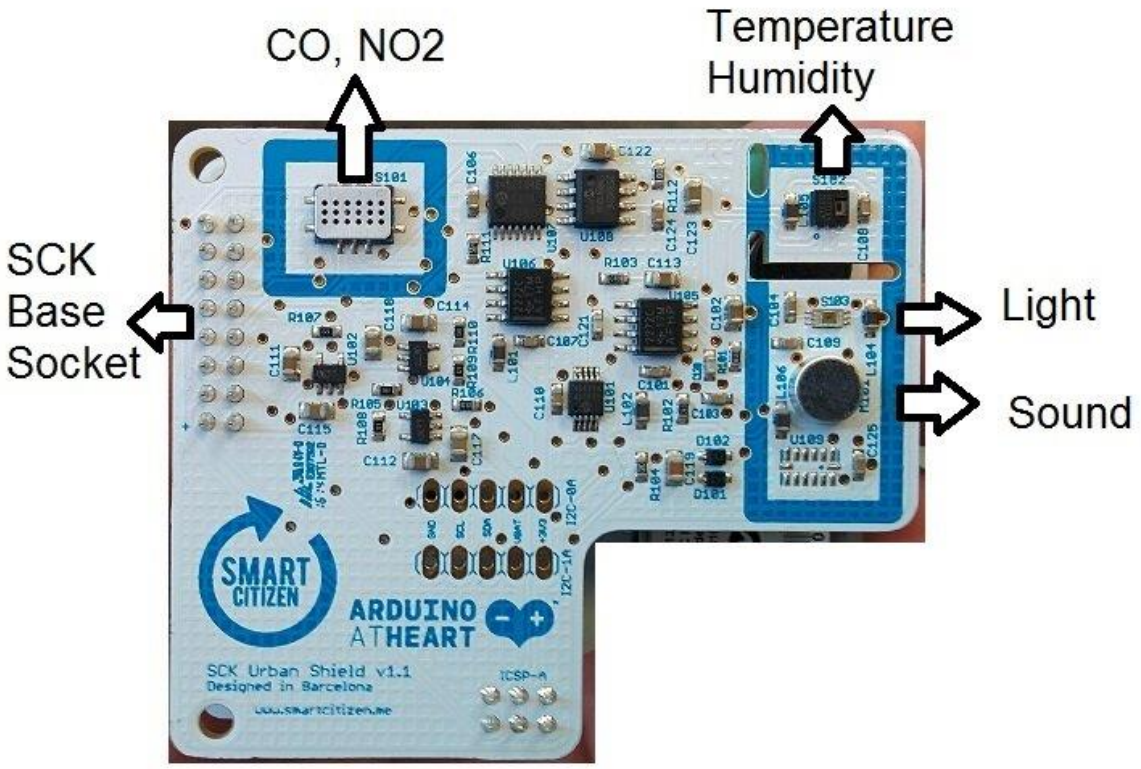


Рисунок 1.3 – Плата Smart Citizen Kit

У сучасному світі активно розвиваються різноманітні проєкти, що реалізують концепцію екологічного моніторингу на базі IoT-технологій. Їх аналіз дозволяє краще зрозуміти доступні технологічні підходи, обґрунтувати вибір апаратних компонентів і сформувати вимоги до майбутньої системи. У табл. 1.1 нижче представлено порівняльний огляд трьох популярних рішень – AirVisual Node, OpenSenseMap та Smart Citizen Kit – за ключовими параметрами: типами сенсорів, способом передавання даних, енергоспоживанням і орієнтовною вартістю.

Таблиця 1.1 – Порівняння існуючих IoT-рішень для екологічного моніторингу

Проєкт	**Типи сенсорів**	**Передача даних**	**Енергоспоживання**	**Орієнтовна вартість**
AirVisual Node	CO_2, PM2.5, температура, вологість	Wi-Fi (передача в хмару)	Помірне (потребує регулярного живлення через адаптер)	Близько 200–250 USD

Продовження таблиці 1.1

OpenSenseMap	Різні (PM2.5, температура, вологість, шум) залежно від конфігурації користувача	Wi-Fi, LoRa (через самостійно налаштовані модулі)	Залежить від обраного мікроконтролера (може бути оптимізоване)	Від 50 USD (самостійне складання)
Smart Citizen Kit	Температура, вологість, шум, CO, NO_2, освітленість	Wi-Fi (передача на відкриту платформу)	Відносно високе (потрібне постійне підключення до живлення)	Близько 150–180 USD

Аналіз представлених IoT-рішень свідчить про наявність різноманітних підходів до реалізації систем екологічного моніторингу. Кожен з розглянутих проєктів має свої переваги: AirVisual Node забезпечує високу точність та зручний інтерфейс, OpenSenseMap — гнучкість у конфігурації та відкритість, а Smart Citizen Kit — модульність і підтримку активної спільноти [7]. Водночас жодна з систем не позбавлена недоліків: висока вартість, залежність від постійного живлення або потреба у самостійній збірці. Узагальнені результати дозволяють сформувати технічні орієнтири для побудови власної IoT-системи, яка поєднуватиме переваги наявних рішень і відповідатиме сучасним вимогам до автономності, масштабованості та доступності.

1.3 Постановка задачі та визначення вимог до системи

З урахуванням результатів аналізу предметної області та існуючих IoT-рішень, постає завдання створення ефективної системи моніторингу екологічного стану міста, що ґрунтується на використанні технологій Інтернету речей. Основна мета системи — забезпечення безперервного збору, обробки та візуалізації даних про стан навколишнього середовища з використанням сенсорів, підключених до мікроконтролера ESP32.

Система повинна реалізовувати такі функції:

– автоматизований збір екологічних даних (рівень CO_2, температура, вологість, запиленість тощо) за допомогою відповідних сенсорів;

– передача зібраних даних на сервер у реальному часі або з визначеною періодичністю;

– збереження інформації у базі даних для подальшого аналізу та формування історичних звітів;

– відображення актуальних і архівних показників у зручному веб-інтерфейсі;

– формування попереджень у разі перевищення заданих екологічних порогів [8].

Задля досягнення зазначених цілей, до системи висуваються наступні функціональні та технічні вимоги:

1. Автономність

Система повинна функціонувати без постійного втручання користувача, з можливістю тривалого живлення від акумулятора або енергозберігаючого джерела. ESP32 має підтримувати режими сну для мінімізації енергоспоживання.

2. Точність сенсорів

Вибрані датчики повинні забезпечувати достатню точність для виявлення змін екологічних показників у межах допустимих похибок, встановлених екологічними стандартами.

3. Безперервна передача даних

Система повинна підтримувати стабільне бездротове з’єднання (Wi-Fi або LoRa) для регулярної передачі даних на сервер, а також мати механізм буферизації у разі втрати з’єднання.

4. Захист інформації

Передбачена передача даних повинна відбуватися з урахуванням базових принципів кібербезпеки — зокрема, з використанням унікального ідентифікатора пристрою, API-ключа або інших засобів автентифікації.

5. Інтерфейс користувача

Дані мають бути представлені у вигляді таблиць, графіків або інтерактивних панелей. Веб-інтерфейс має бути адаптивним для перегляду з ПК і мобільних пристроїв.

IoT-система повинна поєднувати апаратну надійність, гнучкість програмної архітектури та зручність взаємодії з користувачем, що дозволить ефективно контролювати екологічний стан міських територій.

1.4 Обґрунтування вибору апаратного та програмного забезпечення

Одним із ключових апаратних компонентів у запропонованій IoT-системі є мікроконтролер ESP32, який обрано завдяки його широким функціональним можливостям, високій енергоефективності та доступності. На відміну від багатьох альтернатив, ESP32 має вбудовані модулі Wi-Fi та Bluetooth, що дозволяє реалізувати бездротову передачу даних без потреби у додаткових модулях, зменшуючи загальну складність схеми та знижуючи витрати [9].

Мікроконтролер побудований на базі двоядерного процесора Tensilica, що забезпечує достатній рівень обчислювальної потужності для обробки даних з кількох сенсорів у режимі реального часу. Наявність великої кількості GPIO-портів дозволяє підключати різноманітні сенсори без використання додаткових розширювачів.

Перевага ESP32 – підтримка енергозберігаючих режимів, зокрема Deep Sleep, що критично важливо для автономної роботи пристрою протягом тривалого часу при живленні від акумулятора або сонячної батареї. Завдяки цьому мікроконтролер ідеально підходить для систем моніторингу, які розміщуються у віддалених або важкодоступних місцях.

ESP32 відзначається низькою вартістю (в середньому 4–8 USD), що дозволяє будувати масштабовані мережі з багатьох вузлів моніторингу без суттєвих фінансових витрат. Активна спільнота розробників, велика кількість бібліотек та документації спрощують процес програмування та інтеграції з іншими компонентами системи.

У сукупності ці характеристики роблять ESP32 оптимальним вибором для побудови доступної, надійної та масштабованої IoT-системи екологічного моніторингу.

Для реалізації системи моніторингу обрано низку сенсорів, які відповідають вимогам до точності, доступності та сумісності з мікроконтролером ESP32. Зокрема, у проєкті використовуються MQ-135, DHT22 та GP2Y1010AU0F — кожен з яких виконує конкретну функцію у вимірюванні параметрів навколишнього середовища.

MQ-135 застосовується для визначення якості повітря, зокрема вмісту таких газів, як CO_2, NH_3, NOx, спирти, бензин та дим (рис. 1.3). Він широко використовується в системах екологічного моніторингу завдяки здатності виявляти різні шкідливі речовини та адаптації до зміни умов середовища. Перевагами цього сенсора є його простота інтеграції, доступна ціна та наявність калібрувальних даних, що дозволяє підвищити точність вимірювання [10].

Рисунок 1.3 – Модуль MQ-135 якості повітря

DHT22 - це цифровий датчик температури та вологості, який забезпечує досить високу точність (±0.5 °C для температури та ±2–5 % RH для вологості) і стабільну роботу в умовах зовнішнього середовища (рис. 1.4). На відміну від свого

простішого аналога DHT11, цей сенсор має розширений діапазон вимірювання та кращу роздільну здатність, що дозволяє використовувати його в системах, де потрібна більша надійність.

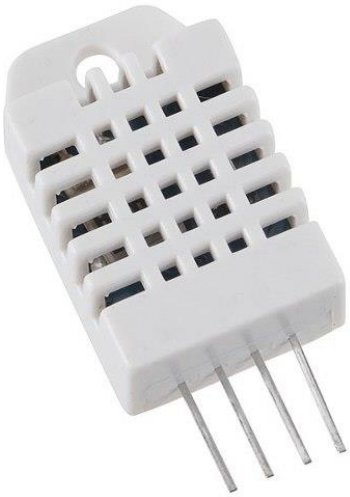


Рисунок 1.4 – Датчик вологи та температури DHT22

Для оцінки рівня запиленості повітря застосовується GP2Y1010AU0F — оптичний датчик пилу, який дозволяє виявляти зважені частки у повітрі, включаючи PM2.5. Принцип дії базується на розсіюванні інфрачервоного світла частками пилу, що дозволяє отримати швидкий та надійний результат у реальному часі [11]. Компактні розміри та низьке енергоспоживання роблять цей датчик придатним для автономних IoT-систем.

Рисунок 1.5 – Датчик пилу GP2Y1010AU0F

Обрані сенсори забезпечують комплексне вимірювання ключових параметрів стану довкілля, зберігаючи баланс між функціональністю, точністю та вартістю. Всі вони легко інтегруються з ESP32 та підтримуються стандартними бібліотеками Arduino, що спрощує процес розробки та налаштування системи.

Для реалізації повноцінної IoT-системи моніторингу екологічного стану необхідно застосувати програмне забезпечення для кожного з рівнів архітектури: мікроконтролерного, серверного та клієнтського.

Програмування мікроконтролера ESP32 здійснюється мовою C++ із використанням середовища Arduino IDE. Такий підхід забезпечує гнучкість у роботі з апаратними портами, підключенням сенсорів та реалізацією логіки передачі даних. Arduino-платформа підтримує численні бібліотеки для роботи з датчиками MQ-135, DHT22, GP2Y1010AU0F, а також для налаштування Wi-Fi-модуля, що значно спрощує розробку [12].

Для обробки даних на серверному боці використовується мова Python разом із мікрофреймворком Flask, який забезпечує обробку HTTP-запитів, приймання даних від ESP32, збереження їх у базі даних та підготовку відповідей для клієнтської частини. Flask вирізняється простою архітектурою, низькими вимогами до ресурсів та хорошою масштабованістю, що робить його оптимальним вибором для невеликих систем IoT.

Веб-інтерфейс розроблено з використанням HTML, CSS та JavaScript, що дозволяє створити інтерактивний інтерфейс для перегляду екологічних показників. Для побудови графіків або динамічного оновлення даних можуть додатково використовуватися JavaScript-бібліотеки, такі як Chart.js або AJAX. Такий підхід забезпечує зручність доступу до системи з будь-якого пристрою — як з ПК, так і зі смартфона.

Загальна структура програмного забезпечення є клієнт-серверною та дозволяє реалізувати модульну, масштабовану й ефективну IoT-систему для моніторингу стану довкілля.

2 ПРОЄКТУВАННЯ IoT-СИСТЕМИ МОНІТОРИНГУ

2.1 Сучасні мікроконтролери для керування пристроями

Мікроконтролер (МК) – це компактний інтегральний пристрій, який містить процесорне ядро, пам’ять (ОЗП, ПЗП), інтерфейси введення/виведення (I/O) та периферійні модулі (таймери, АЦП, UART, SPI), об’єднані на одному кристалі [13]. Мікроконтролери використовуються для автоматизованого керування електронними пристроями, обробки сигналів від сенсорів, а також передачі даних.

У контексті Інтернету речей (IoT) мікроконтролери є базовими обчислювальними елементами, що забезпечують:

– зчитування даних із фізичних сенсорів (температури, вологості, руху тощо);

– прийняття рішень на основі заданої логіки або зовнішніх команд;

– керування виконавчими механізмами (реле, двигуни, освітлення);

– передавання даних у хмарні сервіси або локальні сервери через протоколи Wi-Fi, Bluetooth, LoRa, ZigBee, MQTT, HTTP тощо.

Роль мікроконтролерів в IoT надзвичайно важлива, адже вони:

– забезпечують енергоефективність, необхідну для автономної роботи пристроїв;

– дозволяють розробляти розподілені системи зі збиранням і обробкою даних без постійного втручання користувача;

– є дешевшими та компактнішими у порівнянні з повноцінними комп’ютерами, що робить їх ідеальними для масового розгортання.

Типовими прикладами використання мікроконтролерів в IoT є: "розумні будинки", системи поливу, відеоспостереження, автоматизація освітлення, контроль навколишнього середовища, охоронні комплекси та побутові пристрої.

ESP-32 – це високопродуктивний мікроконтролер виробництва компанії Espressif Systems, який поєднує двоядерний процесор (Xtensa LX6) із частотою до 240 МГц, підтримку Wi-Fi і Bluetooth, а також велику кількість цифрових

інтерфейсів (SPI, I2C, UART, PWM тощо). Пристрій вирізняється низьким енергоспоживанням і широкими комунікаційними можливостями, що робить його ідеальним для IoT-проєктів [14]. Завдяки відкритим бібліотекам, сумісності з Arduino IDE, а також численним прикладам і документації, ESP-32 активно використовується як у побутових, так і в промислових розробках (рис. 2.1).

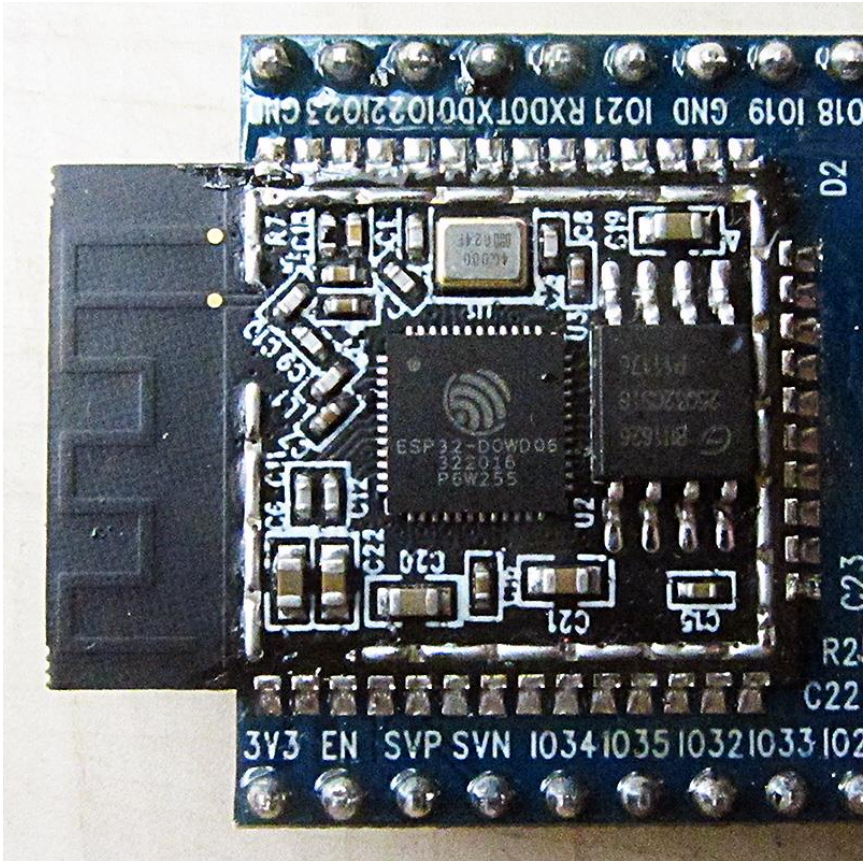


Рисунок 2.1 – Зовнішній вигляд модуля ESP-32 з центральним чіпом ESP32-D0WDQ6

Arduino Uno – один із найвідоміших мікроконтролерів на базі чіпа ATmega328P. Він має 14 цифрових і 6 аналогових входів/виходів, працює на частоті 16 МГц, підтримує інтерфейси UART, SPI, I2C. Arduino Uno широко застосовується в навчальних і аматорських проєктах завдяки простоті програмування та великій спільноті користувачів (рис. 2.2). Проте він має обмежену продуктивність і не підтримує бездротові протоколи "з коробки", тому для IoT-застосувань потребує додаткових модулів (наприклад, Wi-Fi-модуля ESP8266) [15].

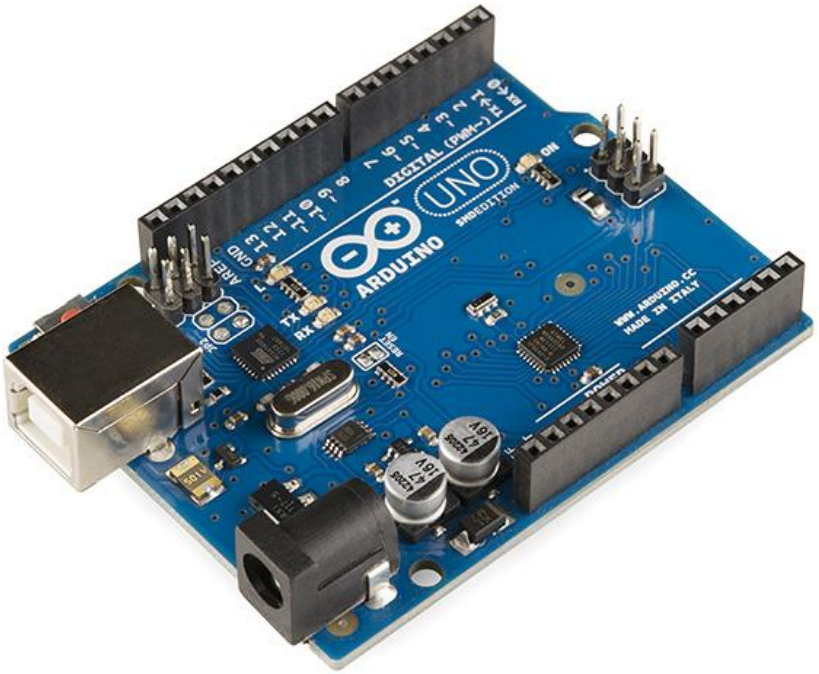


Рисунок 2.2 – Плата Arduino Uno – популярний мікроконтролер на базі чіпа ATmega328P

STM32 – сімейство потужних 32-бітних мікроконтролерів на базі ARM Cortex-M, вироблених компанією STMicroelectronics (рис. 2.3). Моделі цієї серії охоплюють широкий діапазон – від бюджетних рішень до високопродуктивних варіантів із підтримкою графіки, Ethernet, USB і бездротових модулів. STM32 відзначаються високою енергоефективністю, гнучкістю конфігурації, можливістю роботи в реальному часі (RTOS) [16]. Їх частіше використовують у промисловості, автомобільній електроніці та медичних пристроях. Однак поріг входу для новачків вищий через складнішу екосистему.

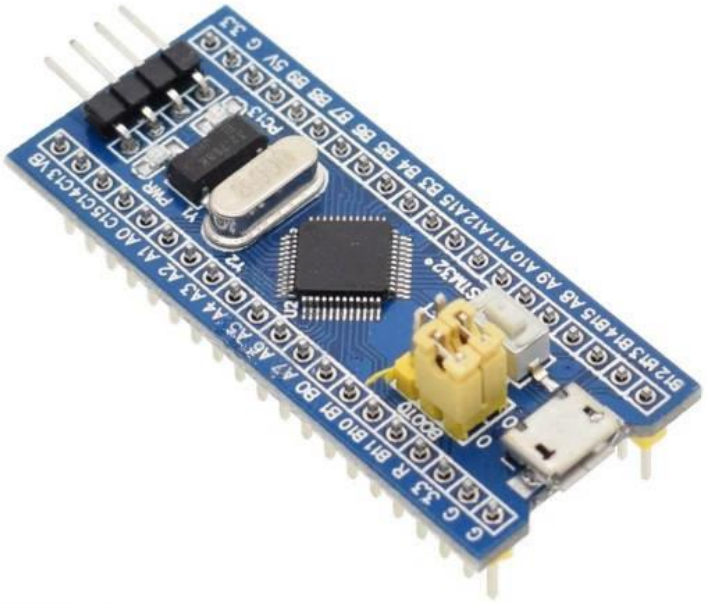


Рисунок 2.3 – STM32F103C8T6 («Blue Pill») – мікроконтролерна плата на базі ARM Cortex-M3

Raspberry Pi Pico – це недорогий мікроконтролер на базі чіпа RP2040 від Raspberry Pi Foundation (рис. 2.4). Він має два ядра ARM Cortex-M0+, працює на частоті до 133 МГц, підтримує численні GPIO та інтерфейси SPI, I2C, UART, PWM. Pico програмується як на C/C++, так і на MicroPython, що робить його зручним для освітніх і швидких прототипів [17]. Однак на відміну від ESP-32, він не має вбудованого Wi-Fi чи Bluetooth, тому для IoT-проєктів вимагає додаткових модулів.

Рисунок 2.4 – Плата Raspberry Pi Pico W на базі чіпа RP2040 з вбудованим Wi-Fi-модулем

Таблиця 2.1 – Порівняльна таблиця популярних мікроконтролерів для IoT-систем

Критерій	**ESP-32**	**Arduino Uno**	**STM32 (наприклад, STM32F103)**	**Raspberry Pi Pico**
Вартість	$3–6	$8–12	$5–15 (залежно від моделі)	$4–6
Енергоефективність	Висока, є режими сну	Середня	Висока, з глибоким сном	Висока
Обчислювальні ресурси	2 ядра, 240 МГц, до 520 КБ SRAM	1 ядро, 16 МГц, 2 КБ SRAM	1 ядро, 72 МГц, 20–64 КБ SRAM	2 ядра, 133 МГц, 264 КБ SRAM
Інтерфейси зв’язку	Wi-Fi, Bluetooth, UART, SPI, I2C	UART, SPI, I2C	UART, SPI, I2C, CAN, USB	UART, SPI, I2C, USB
Підтримка бездротового зв'язку	Вбудовано Wi-Fi, BLE	Потребує окремих модулів	Потребує зовнішніх модулів	Потребує зовнішніх модулів
Програмування	Arduino IDE, ESP-IDF, MicroPython	Arduino IDE	STM32CubeIDE, Keil, PlatformIO	C/C++, MicroPython
Підтримка спільноти	Дуже велика, багато бібліотек	Дуже велика	Велика, технічно орієнтована	Середня, швидко зростає
Призначення	IoT, smart home, автоматизація	Освітні цілі, прототипування	Промислові застосування	Навчання, прототипування

Аналіз показує, що ESP-32 перевершує інші розглянуті мікроконтролери за кількома критичними параметрами, які є визначальними для систем віддаленого керування в контексті IoT:

– має вбудовані модулі Wi-Fi та Bluetooth, що значно спрощує реалізацію бездротової комунікації без додаткових апаратних модулів;

– володіє потужною обчислювальною базою (двоядерний процесор до 240 МГц);

– підтримує енергоощадні режими, що важливо для пристроїв, які працюють автономно;

– активно підтримується спільнотою розробників, має відкриті бібліотеки, документацію та приклади;

– програмується у звичних середовищах, таких як Arduino IDE або ESP-IDF.

ESP-32 є оптимальним рішенням для реалізації системи моніторингу екологічного стану міста, оскільки поєднує в собі функціональність, гнучкість, доступність та зручність розробки, що повністю відповідає вимогам обраного проєкту.

2.2 Середовище проєктування апаратного забезпечення

Аналіз існуючих аналогів систем та вивчення профільної літератури дозволили визначити найбільш ефективний та швидкий шлях освоєння навичок роботи з мікроконтролером ESP32, а також розуміння його інфраструктури та принципів проєктування систем. Одним із таких шляхів є застосування віртуальних інструментів для моделювання роботи мікроконтролерів, сенсорів та іншої периферії, яка використовується під час створення IoT-систем.

Віртуальні середовища моделювання мають ряд суттєвих переваг:

– Підвищення безпеки. Робота з фізичними компонентами завжди супроводжується ризиком коротких замикань, перегріву або пошкодження апаратури. Використання віртуальної симуляції дає можливість вільно експериментувати з кодом та схемами без ризику виходу з ладу обладнання.

– Економія часу та ресурсів. Немає потреби витрачати кошти на придбання фізичних модулів та їхнє налаштування. Завдяки віртуальним платформам можливо швидко конфігурувати і тестувати системи без значних матеріальних витрат.

Серед найпоширеніших рішень для віртуальної симуляції виділяється сервіс Wokwi. Під час роботи користувач має можливість миттєво спостерігати, як внесені зміни в код впливають на поведінку підключених віртуальних пристроїв. Такий

підхід значно полегшує розуміння логіки виконання програмного коду, а також сприяє своєчасному виявленню та усуненню помилок.

Розширена бібліотека електронних компонентів, дозволяє працювати з різними типами сенсорів, модулів зв'язку, дисплеїв, джерел живлення тощо без потреби у їх фізичному придбанні. Це відкриває широкі можливості для експериментів і тестування різноманітних схем і конфігурацій.

Крім того, платформа має активну спільноту користувачів і розробників, яка регулярно оновлює бібліотеки, ділиться прикладами та допомагає новачкам у вирішенні типових проблем. Така підтримка сприяє пришвидшеному навчанню та глибшому розумінню роботи з мікроконтролерами.

Архітектура сервісу Wokwi передбачає наявність кількох ключових модулів, які разом утворюють потужне середовище для створення та симуляції IoT-проєктів:

– Інтерактивне середовище для редагування коду (рис. 2.5) включає в себе підтримку мов програмування C/C++ для Arduino-платформ, а також MicroPython — для мікроконтролерів ESP32 та Raspberry Pi Pico. Середовище оснащене інструментами підсвічування синтаксису, автозавершення та виводу логів у режимі реального часу, що значно покращує процес написання та відлагодження програм [18].

– Апаратний симулятор (рис. 2.6) дозволяє моделювати роботу різних мікроконтролерів (таких як ESP32, ESP8266, Arduino Uno), а також широкого спектра периферійних пристроїв — дисплеїв, сенсорів, світлодіодів, кнопок тощо. Цей модуль наочно демонструє взаємодію коду з «обладнанням», забезпечуючи повне розуміння принципів роботи системи.

– Бібліотека віртуальних компонентів (рис. 2.7) містить набір популярних елементів, які можна легко додавати до проєкту. Це дозволяє максимально гнучко будувати власні конфігурації систем та оперативно перевіряти працездатність запрограмованої логіки.

Wokwi виступає ефективним інструментом для вивчення, тестування та візуалізації IoT-рішень, особливо на етапі навчання або розробки прототипів.

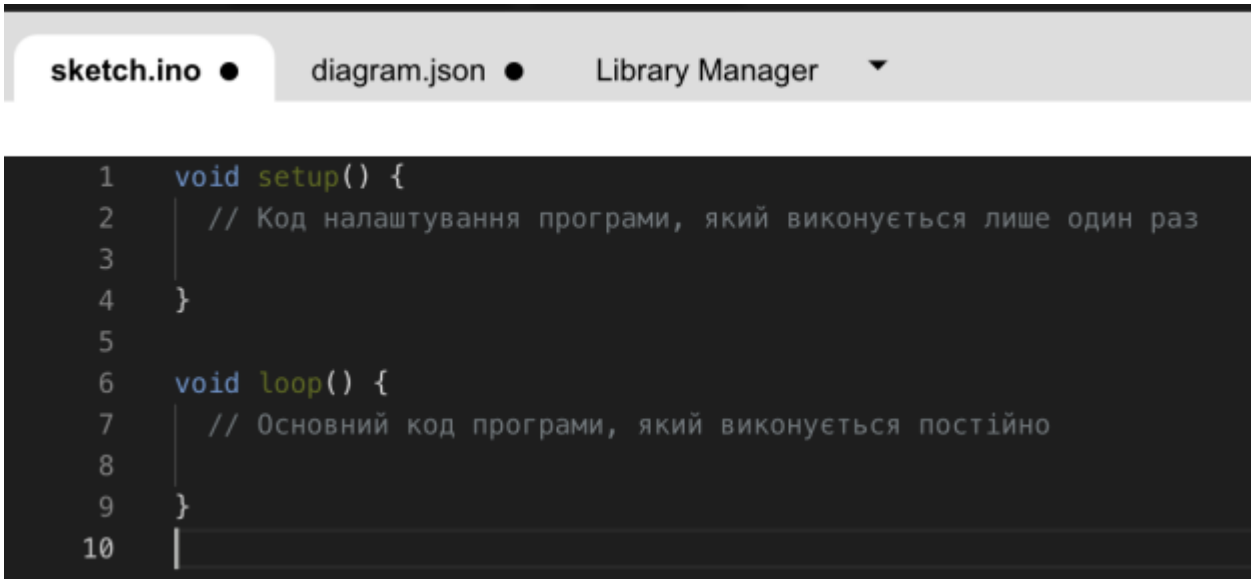


Рисунок 2.5 – Вікно середовища програмування Wokwi з початковою структурою скетчу Arduino

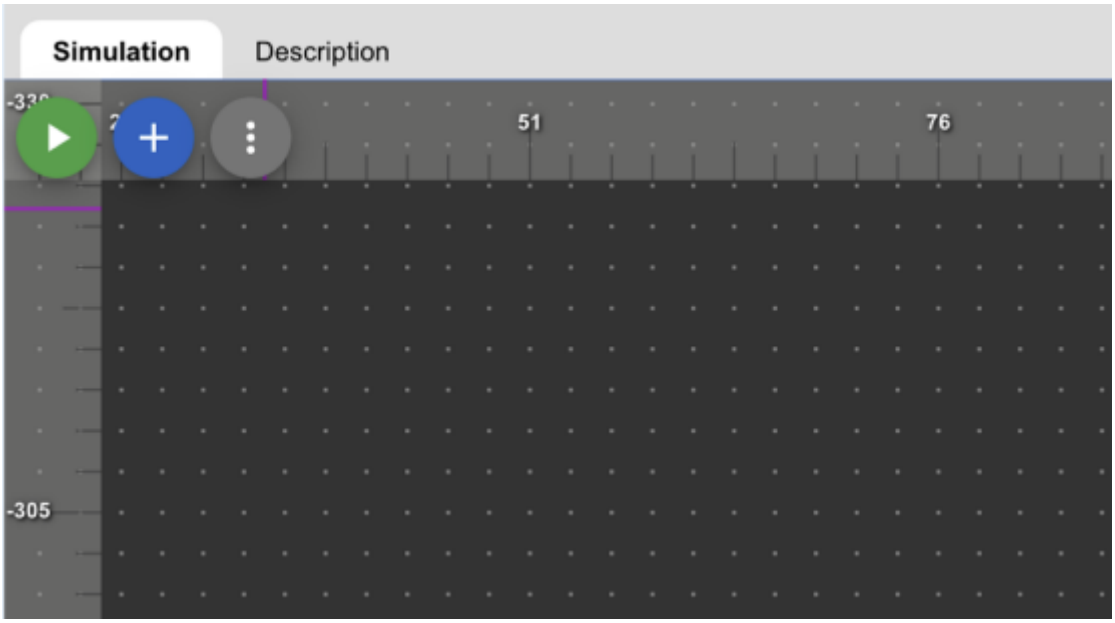


Рисунок 2.6 – Область побудови апаратної схеми у середовищі симуляції Wokwi

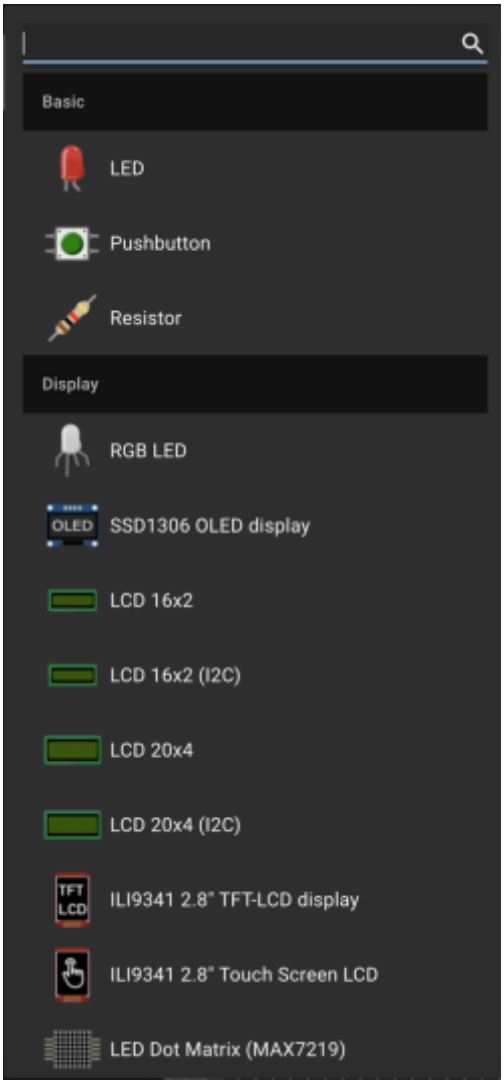


Рисунок 2.7 – Приклад бібліотеки доступних електронних компонентів у Wokwi для побудови IoT-схем

На основі попереднього вибору апаратних компонентів здійснено проєктування відповідного обладнання для IoT-системи. На початковому етапі з бібліотеки елементів середовища Wokwi було обрано необхідні компоненти, які розміщено на робочому полі симулятора. Після цього виконано логічне з’єднання між пристроями з урахуванням вимог до функціонування системи. В результаті сформовано повну електричну схему пристрою, яка наведена на рисунку 2.8.

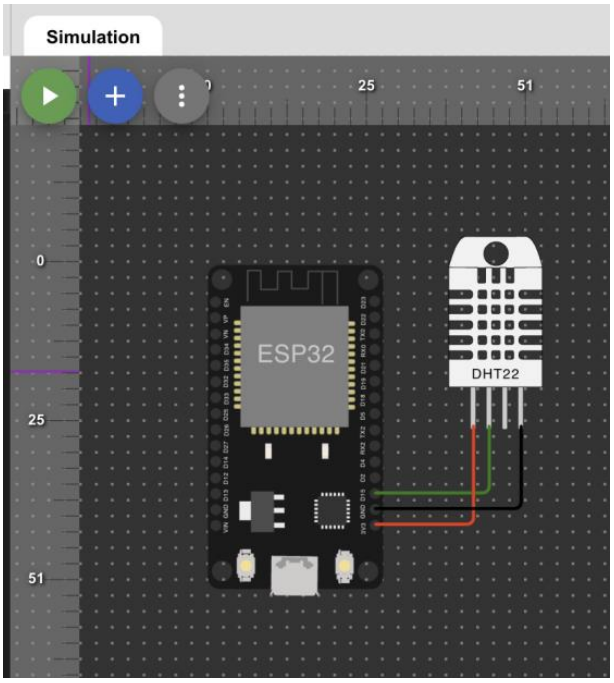


Рисунок 2.7 – Схема з'єднання мікроконтролера ESP32 з датчиком температури та вологості DHT22 у середовищі Wokwi

2.3 Архітектура програмного забезпечення та принципи проєктування

Архітектура програмного забезпечення є ключовим концептуальним уявленням системи, що визначає її загальну організацію, структуру компонентів, принципи взаємодії та поведінку на всіх рівнях. Вона формує базу для проєктування і реалізації системи, забезпечуючи цілісне бачення її функціонування.

Добре спроєктована архітектура, адаптована до специфіки майбутньої системи, дозволяє створити ефективне, стабільне та масштабоване програмне забезпечення. Вона сприяє узгодженій роботі всіх модулів, визначає чіткий розподіл обов’язків між компонентами, а також регламентує способи обробки і зберігання даних.

Крім того, архітектура програмного забезпечення охоплює механізми взаємодії з кінцевими користувачами, зовнішніми системами та сервісами. Саме в ній закладаються технічні рішення, що мають вирішальний вплив на загальну надійність, продуктивність, безпеку та можливість масштабування проєкту.

Важливість архітектурного підходу полягає також у зниженні ризиків, які супроводжують процес розробки. Чітке архітектурне бачення дає змогу зменшити кількість помилок, уникнути неузгодженостей між розробниками, оптимізувати витрати часу і ресурсів, а також забезпечити відповідність функціоналу очікуванням користувачів. Відсутність структурованої архітектури, навпаки, може призвести до технічних труднощів, перевищення бюджету, затримок у розробці або навіть до повного припинення реалізації проєкту.

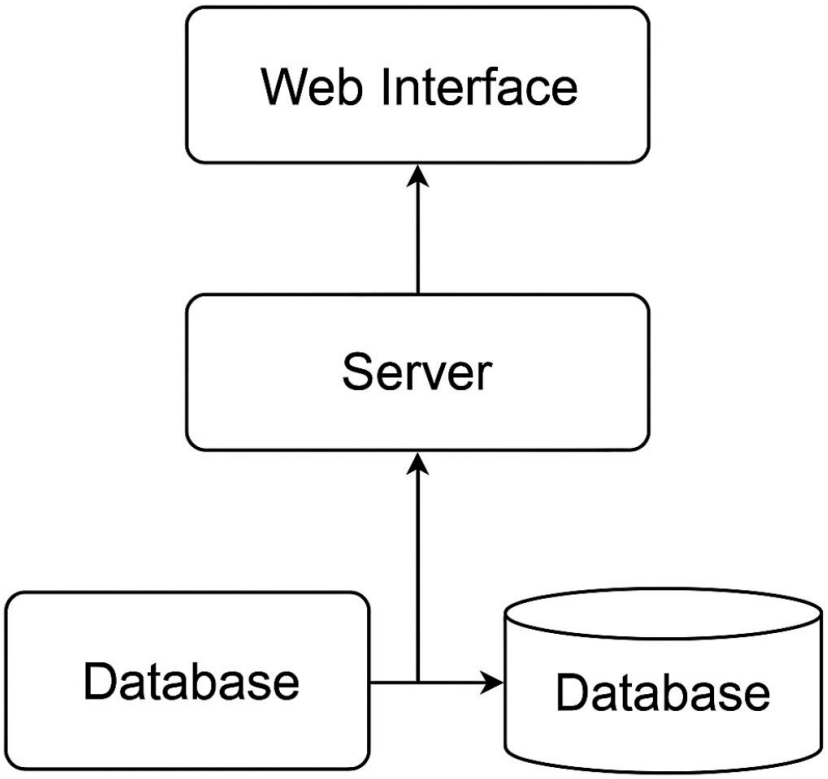


Рисунок 2.8 – Архітектура програмного забезпечення IoT-системи

Слід зазначити, що архітектура програмного забезпечення не є фіксованою та незмінною. Вона повинна постійно адаптуватися до змін як у функціональних вимогах, так і в технологічному середовищі. Гнучкість архітектури, її здатність трансформуватися у відповідь на нові виклики є важливою ознакою її якісного проєктування. Ефективна архітектура здатна не лише підтримувати поточну функціональність, але й забезпечувати її розширення без суттєвих ризиків чи ускладнень [19].

До основних характеристик архітектури сучасних інформаційних систем, зокрема IoT-рішень, належать:

Масштабованість — здатність системи стабільно працювати при зростанні навантаження, що може проявлятися через збільшення кількості пристроїв, обсягу переданих даних або одночасних запитів користувачів. У контексті систем екологічного моніторингу це особливо важливо, оскільки кількість сенсорів у різних географічних точках може динамічно зростати. Для забезпечення масштабованості зазвичай використовують такі підходи, як горизонтальне масштабування серверів, розподілену обробку даних, використання хмарних обчислень та високопродуктивних баз даних. Окрім збільшення обсягу даних, масштабованість також стосується можливості додавання нової функціональності: система має легко адаптуватися до змін вимог без втрати стабільності.

Відмовостійкість — одна з ключових вимог до критичних систем, яка передбачає здатність до автоматичного відновлення після збоїв без втрати інформації та з мінімальним впливом на працездатність. Для IoT-рішень, де дані передаються з багатьох сенсорів у реальному часі, це особливо актуально. У випадку втрати зв’язку з одним із пристроїв система повинна продовжувати функціонувати, а також мати механізми швидкого відновлення або переключення на резервне джерело даних. Це забезпечується через регулярне резервне копіювання, використання дублюючих каналів зв’язку та реалізацію механізмів самодіагностики.

Безпека — критичний аспект архітектури, особливо для систем, що оперують конфіденційною, персональною або стратегічною інформацією. В умовах IoT-середовища, де велика кількість пристроїв підключена до мережі, питання захисту даних стає надзвичайно важливим. Архітектурне рішення має включати механізми шифрування переданих даних, автентифікацію користувачів, контроль доступу до сервісів, а також виявлення та запобігання несанкціонованому втручанню. Приділяючи достатню увагу інформаційній безпеці на етапі архітектурного проєктування, можна мінімізувати ризики, пов’язані з кібератаками або витоками інформації.

Модульність — архітектурний підхід, що передбачає поділ системи на незалежні логічні блоки або модулі, кожен з яких виконує чітко визначене завдання. Така структура спрощує супровід, оновлення та масштабування системи. Наприклад, у рішенні для моніторингу навколишнього середовища можуть бути окремі модулі для збору даних із сенсорів, їх обробки та аналізу, збереження в базі даних та візуалізації для користувача. Завдяки цьому можна незалежно тестувати, вдосконалювати або замінювати окремі частини системи без впливу на її загальну працездатність.

Щоб забезпечити гнучкість та стабільність програмного рішення, взаємодія між окремими модулями системи має здійснюватися відповідно до заздалегідь визначених чітких інтерфейсів або контрактів. Це дозволяє зменшити ступінь залежності між компонентами, підвищуючи їхню автономність. Такий підхід є ефективним не лише в контексті створення програмного забезпечення, а й у проєктуванні складних технічних систем, що складаються з багатьох підсистем. Завдяки цьому спрощується внесення змін до окремих частин системи, зменшується ризик виникнення помилок під час модифікацій і зростає швидкість розробки, адже над різними модулями можна працювати паралельно.

Модульний підхід сприяє кращому розумінню архітектури системи. Кожна функціональна одиниця зосереджена в межах окремого модуля, що дозволяє розробникам не заглиблюватися в деталі реалізації – достатньо ознайомитися з інтерфейсом, який визначає основні функції та правила використання модуля. Таким чином, контракти між модулями відіграють роль узагальнених описів функціональності, спрощуючи супровід і розширення системи.

Ще однією важливою вимогою до архітектури є ефективність, що полягає у раціональному використанні наявних апаратних та програмних ресурсів. Ідеться про оптимізацію обчислювальних процесів, оперативної пам’яті, пропускної здатності мережі та сховищ даних. Наприклад, в умовах IoT-рішення, яке обробляє значні обсяги інформації з великої кількості сенсорів, ефективна архітектура повинна передбачати застосування високопродуктивних алгоритмів обробки

даних, мінімізацію навантаження на процесор, оптимізацію обміну даними в мережі для зменшення трафіку [20].

Таким чином, дотримання принципів модульності та ефективності дозволяє забезпечити надійність, масштабованість і стабільну роботу програмного забезпечення навіть за умов інтенсивного навантаження та розширення функціональних можливостей системи.

3 РЕАЛІЗАЦІЯ СИСТЕМИ ТА ОЦІНКА ЇЇ ЕФЕКТИВНОСТІ

3.1 Вибір стеку технологій для розробки IoT-системи

Однією з важливих технологій, яка використовується у розробці IoT-систем та інтеграції їх з веб-інтерфейсами, є мова програмування ***TypeScript***. Вона була створена компанією Microsoft як надбудова над JavaScript із розширеною підтримкою типізації. Фактично, TypeScript компілюється у звичайний JavaScript, що забезпечує повну сумісність двох мов: валідний JavaScript-код є коректним і для TypeScript, а скомпільовані TypeScript-програми можуть використовуватися в будь-якому сучасному веб-проєкті [21].

Основною перевагою використання TypeScript у проєктуванні IoT-систем є можливість застосування статичної типізації. Це дозволяє ще на етапі розробки виявляти помилки, пов'язані з неправильним використанням типів даних, що підвищує надійність і якість програмного забезпечення. Під час оголошення змінних у TypeScript чітко визначається їх тип, що унеможливлює зміну типу в процесі виконання програми. Це особливо важливо для створення надійних клієнтських та серверних частин IoT-системи, де передача даних повинна бути контрольованою та безпечною.

Крім того, підтримка складних типів, таких як інтерфейси, кортежі, перелічення та використання дженериків, дає можливість створювати гнучкі, розширювані компоненти системи, що зберігають типову безпеку. Розробка із застосуванням TypeScript значно спрощується завдяки інтеграції з сучасними середовищами розробки, такими як Visual Studio Code або WebStorm, які надають підказки щодо типів прямо під час написання коду. У контексті створення IoT-рішень для моніторингу та аналізу даних така перевага сприяє підвищенню ефективності розробки, покращенню стабільності коду та оптимізації часу розробки.

TypeScript також надає широкі можливості для об'єктно-орієнтованого програмування (ООП), включаючи підтримку класів, інтерфейсів та модульної

структури, що значно спрощує створення великих і складних програмних систем на основі JavaScript [22]. Завдяки цьому TypeScript стає ефективним інструментом для розробки архітектурно складних IoT-рішень, де важлива чітка організація коду та розмежування відповідальностей між компонентами системи.

Однією з переваг TypeScript є надання розробникам доступу до функцій, які ще не були офіційно впроваджені у стандарт JavaScript, але плануються до включення у майбутніх версіях. Зокрема, підтримка декораторів дає можливість модифікувати або розширювати функціональність класів, методів чи властивостей, що особливо актуально при використанні сучасних фреймворків, таких як Angular. Окрім цього, TypeScript має вбудовану підтримку синтаксису JSX, що робить його зручним для роботи з такими бібліотеками, як React, де написання HTML-подібного коду безпосередньо в середовищі JavaScript спрощує розробку інтерактивних інтерфейсів для IoT-платформ [23].

Попри значні переваги, TypeScript має і певні недоліки. Зокрема, необхідність компіляції коду в JavaScript може збільшувати час розробки, особливо при створенні масштабних систем. Крім того, хоч TypeScript і сумісний із JavaScript-бібліотеками, інколи виникають труднощі через відсутність готових типових описів для деяких сторонніх бібліотек, що ускладнює інтеграцію. Проте загалом використання TypeScript у проєктах моніторингу та аналітики даних в рамках IoT-рішень значно підвищує надійність, підтримуваність і швидкість розвитку системи, полегшує залучення нових розробників та забезпечує більшу прозорість архітектури проєкту.

Однією з ключових технологій для серверної частини розробки сучасних IoT-рішень є ***Node.js*** — відкрите середовище виконання JavaScript, побудоване на високопродуктивному двигуні V8, що використовується у браузері Google Chrome [24]. Основною метою створення Node.js було забезпечення можливості обробки великої кількості одночасних підключень із мінімальними витратами ресурсів. Завдяки подійно-орієнтованій моделі JavaScript та асинхронному виконанню коду, Node.js виявився ідеальною платформою для розробки високонавантажених систем реального часу, таких як системи моніторингу, обробки даних із сенсорів, або

відеоспостереження з підключенням до хмарних сервісів.

У межах реалізації IoT-системи відеоспостереження Node.js дозволяє створювати серверну частину, яка відповідає за прийом, обробку та маршрутизацію запитів між пристроями, базою даних і клієнтським інтерфейсом. Завдяки тому, що і клієнтський, і серверний код можуть бути написані однією мовою — JavaScript/TypeScript — забезпечується цілісність проєкту, полегшується підтримка коду та спрощується інтеграція нових функцій у майбутньому. Це особливо актуально для комплексних IoT-рішень, де важливо зберігати архітектурну узгодженість між усіма рівнями взаємодії пристроїв, серверів і інтерфейсів.

Завдяки своїй гнучкості, масштабованості та активній екосистемі модулів, Node.js широко використовується в IoT-сценаріях, де необхідна обробка великих обсягів даних у режимі реального часу, зокрема для передачі аналітики з відеопотоку або даних із сенсорів на панелі моніторингу чи хмарні сервіси.

Основою роботи Node.js є концепція подійно-орієнтованого програмування, що є критично важливою для розробки високопродуктивних IoT-систем. Код у Node.js виконується асинхронно, тобто операції введення-виведення (I/O), такі як доступ до файлової системи, запити до бази даних або відправлення HTTP-запитів, не блокують виконання інших частин програми [25]. Замість очікування завершення таких операцій використовуються функції зворотного виклику (callback), які обробляють результати, щойно вони стають доступними. Це дозволяє Node.js ефективно обробляти велику кількість одночасних підключень без створення великої кількості потоків, що зменшує навантаження на систему і збільшує її масштабованість.

У розробленій IoT-системі для відеоспостереження з AI-асистентом така модель дозволяє швидко обробляти запити з відеомодулів, сенсорів і клієнтських інтерфейсів, забезпечуючи мінімальні затримки при передачі даних і високу стабільність роботи навіть при великій кількості пристроїв.

На рисунку 3.1 зображено спрощену модель роботи Node.js. Вхідні запити спочатку потрапляють до черги подій, де вони очікують обробки. Подальше

виконання запитів організовано через нескінченний цикл подій (event loop), який управляє обробкою операцій без блокування основного потоку виконання. Якщо цикл натрапляє на операцію введення-виведення, вона делегується у фонові потоки операційної системи. Поки задача виконується у робочому потоці ОС, цикл подій продовжує обробляти нові запити. Після завершення задачі результат знову потрапляє у чергу подій для подальшої обробки відповідною функцією зворотного виклику.

Рисунок 3.1 – Схема обробки запитів у подійно-орієнтованій архітектурі Node.js

Незважаючи на ефективність подійно-орієнтованої моделі, в роботі Node.js існують певні обмеження, пов’язані із виконанням ресурсомістких завдань, які можуть блокувати цикл обробки подій. Такі завдання, на відміну від операцій вводу-виводу, не можуть бути передані до стандартних робочих потоків операційної системи. Для вирішення цієї проблеми Node.js має можливість використовувати власні внутрішні треди, що дозволяє переміщувати складні обчислення в окремі потоки і тим самим уникати блокування основного циклу подій.

Важливим компонентом екосистеми Node.js є менеджер пакетів NPM (Node Package Manager), який забезпечує легкий доступ до великої кількості модулів для розширення функціональності застосунку. Завдяки NPM розробники можуть інтегрувати готові рішення для роботи з мережевими протоколами, файловими системами, базами даних, автентифікацією, тестуванням тощо. Це значно прискорює процес розробки IoT-систем, оскільки дозволяє сконцентруватися на реалізації прикладної логіки без необхідності створювати базові модулі з нуля [26].

Окремо варто зазначити, що хоча Node.js використовує JavaScript як основну мову програмування, середовище має низку власних розширень і особливостей. Зокрема, Node.js впроваджує модульну систему CommonJS для організації коду та

взаємодії між файлами, а також надає спеціалізовані API для роботи з файловими системами, мережами та потоками. Завдяки цим особливостям Node.js активно використовується не тільки для створення веб-серверів та API, але й для розробки інструментів командного рядка та складних серверних застосунків, що ідеально відповідає вимогам побудови масштабованої IoT-системи відеоспостереження.

Підсумовуючи, Node.js є потужним і універсальним середовищем виконання, яке забезпечує ефективну розробку серверних застосунків на основі JavaScript. Його подійно-орієнтована модель програмування, розвинена екосистема готових модулів та природна інтеграція з клієнтською частиною проєкту роблять Node.js оптимальним вибором для створення масштабованих IoT-систем. Завдяки своїй гнучкості та високій продуктивності, Node.js широко використовується не лише у веб-розробці, але й у побудові інтелектуальних платформ моніторингу, обробки даних та автоматизації, що повністю відповідає вимогам розробки сучасних систем відеоспостереження.

PostgreSQL є однією з найпоширеніших об'єктно-реляційних систем управління базами даних, яка відзначається високою надійністю, масштабованістю та гнучкістю. В основі PostgreSQL лежить класична реляційна модель даних із суворим дотриманням принципів ACID (Atomicity, Consistency, Isolation, Durability), що гарантує стабільність транзакцій, цілісність даних та безпечність операцій. Завдяки цим характеристикам PostgreSQL ідеально підходить для створення великих і навантажених застосунків, включно з IoT-системами, які потребують швидкого та надійного зберігання великих обсягів інформації [27].

Важливою особливістю PostgreSQL є підтримка об'єктно-орієнтованих можливостей — розробники можуть створювати власні типи даних, оператори, функції та індексатори, що полегшує роботу зі складними структурами даних. Така гнучкість дозволяє ефективно адаптувати систему управління базами даних до специфічних вимог IoT-проєктів, наприклад для зберігання метаданих відеопотоків або логів роботи сенсорних пристроїв.

Серед додаткових переваг PostgreSQL – розвинена система розширень, яка дозволяє встановлювати у базу даних додаткові функціональні модулі без

необхідності змінювати основний код СУБД. Завдяки підтримці вкладених транзакцій, складних SQL-запитів, тригерів та збережених процедур PostgreSQL забезпечує потужний інструментарій для складної обробки даних. Це робить його оптимальним вибором для серверної частини IoT-системи відеоспостереження з AI-аналітикою, де необхідно забезпечити надійне зберігання результатів аналізу, журналювання подій та швидкий доступ до історичних даних.

PostgreSQL підтримує кілька підходів до масштабування системи, що дозволяє гнучко адаптувати її до зростаючих навантажень у процесі розвитку IoT-проєкту. Основними методами масштабування є вертикальне та горизонтальне масштабування.

Вертикальне масштабування полягає у збільшенні обчислювальних можливостей одного сервера шляхом додавання додаткових процесорів, оперативної пам’яті, дискового простору або вдосконалення мережевої інфраструктури. Цей підхід дозволяє зберігати простоту архітектури бази даних без потреби реалізації складної логіки розподілу даних. Однак вертикальне масштабування має свої межі, обумовлені фізичними характеристиками серверів і зростанням витрат на їх обслуговування при досягненні високих параметрів.

Горизонтальне масштабування реалізується шляхом додавання нових серверів у пул ресурсів. У випадку баз даних це може бути реалізовано через шардінг (розподіл даних між різними серверами) або реплікацію (копіювання даних для забезпечення високої доступності). Горизонтальне масштабування забезпечує гнучке нарощування ресурсів відповідно до обсягів даних та навантаження, однак потребує ретельного планування, налаштування розподілу даних і управління балансуванням запитів.

PostgreSQL має високу стійкість до відмов завдяки використанню журналу відновлення та підтримці реплікації даних, що дозволяє забезпечити збереження і відновлення інформації навіть у разі аварійних ситуацій.

Серед певних недоліків PostgreSQL варто зазначити складність початкового налаштування й адміністрування системи, що може стати викликом для недосвідчених користувачів [28]. Також ця СУБД є досить ресурсоємною

порівняно з легшими рішеннями, такими як MySQL, що слід враховувати при розгортанні IoT-систем у середовищах з обмеженими обчислювальними можливостями.

Загалом, PostgreSQL є ефективною, надійною та масштабованою системою управління базами даних, яка повністю відповідає вимогам сучасних застосунків, зокрема складних IoT-рішень для моніторингу та аналітики даних.

React — це популярна бібліотека для JavaScript, створена компанією Meta, яка призначена для розробки сучасних, інтерактивних користувацьких інтерфейсів. React активно застосовується у проєктах, де важлива висока швидкість взаємодії з користувачем і оновлення вмісту сторінки без повного перезавантаження, що робить його особливо ефективним у побудові клієнтських частин IoT-систем та панелей моніторингу даних [29].

Однією з основних концепцій React є компонентний підхід до побудови інтерфейсу. Кожен компонент у React є автономною частиною коду, яка містить у собі як HTML-розмітку, так і логіку взаємодії. Компоненти можуть бути вкладеними один в одного, утворюючи складні ієрархічні структури, що забезпечує модульність і повторне використання елементів інтерфейсу. Такий підхід значно спрощує підтримку, масштабування та розвиток проєкту, оскільки дозволяє розробникам концентруватися на окремих функціональних одиницях без необхідності заново будувати всю систему.

Завдяки можливості декомпозиції коду React знижує ментальне навантаження на розробників, підвищує зрозумілість і читабельність коду, що особливо важливо при створенні великих і довготривалих IoT-проєктів, де інтерфейси взаємодіють з великою кількістю сенсорних даних та аналітичної інформації.

Однією з ключових концепцій, що забезпечує високу ефективність роботи React, є використання віртуального DOM. Віртуальний DOM представляє собою легку копію реального DOM-дерева браузера і дозволяє оптимізувати процес оновлення інтерфейсу. Завдяки цій технології зміни у відображенні відбуваються набагато швидше, оскільки React перерендерює лише ті частини інтерфейсу, які

зазнали змін, а не всю сторінку цілком.

Алгоритм роботи віртуального DOM виглядає наступним чином: після взаємодії користувача із системою змінюється лише віртуальна копія DOM. Потім React за допомогою внутрішніх алгоритмів порівнює нову версію віртуального DOM із попередньою, знаходить відмінності та оновлює тільки змінені елементи у реальному DOM [30]. Цей підхід значно знижує навантаження на браузер і підвищує швидкість роботи веб-застосунку, що особливо важливо для IoT-платформ із великою кількістю динамічних елементів.

Для підвищення ефективності порівняння елементів віртуального DOM React використовує низку оптимізацій. Основними правилами є: якщо два елементи мають різні типи, вони вважаються повністю різними структурами, а також використання атрибуту key, який допомагає React ідентифікувати стабільні елементи списків. Якщо система виявляє суттєві зміни в структурі дерева компонентів, відповідні частини інтерфейсу демонтуються та створюються заново у реальному DOM. На рисунку 3.2 представлено загальну схему роботи віртуального DOM у React.

Ще однією важливою особливістю React є використання JSX — спеціального синтаксису, який дозволяє вбудовувати HTML-подібні конструкції безпосередньо в код JavaScript. Це суттєво спрощує процес створення компонентів, адже розробники можуть описувати структуру інтерфейсу у звичному для них форматі HTML із можливістю інтеграції виразів JavaScript прямо в розмітку. Завдяки JSX розробка клієнтської частини IoT-систем стає швидшою, а код — більш наочним і зручним для підтримки.

Керування даними в React реалізується через концепти стану (state) та властивостей (props). Стан представляє собою об'єкт, який зберігає дані, що можуть змінюватися протягом життєвого циклу компонента. Будь-які зміни стану автоматично викликають перерендерювання відповідної частини інтерфейсу. Властивості ж використовуються для передачі даних від батьківських компонентів до дочірніх, що дозволяє будувати гнучкі та добре структуровані взаємозв'язки між компонентами.

Важливою особливістю React є однонаправлений потік даних: інформація завжди передається від батьківських компонентів до дочірніх через властивості. Це забезпечує прозорість змін даних та дозволяє краще контролювати їхню обробку. Для передачі загальних даних через кілька рівнів ієрархії компонентів React пропонує механізм контексту (Context), що дозволяє уникати надлишкового передавання властивостей на кожному рівні [31]. Також React активно використовує контрольовані компоненти для роботи з формами, коли значення полів вводяться під повним контролем стану компонентів, що дозволяє точніше обробляти введені користувачами дані та уникати неочікуваних змін.

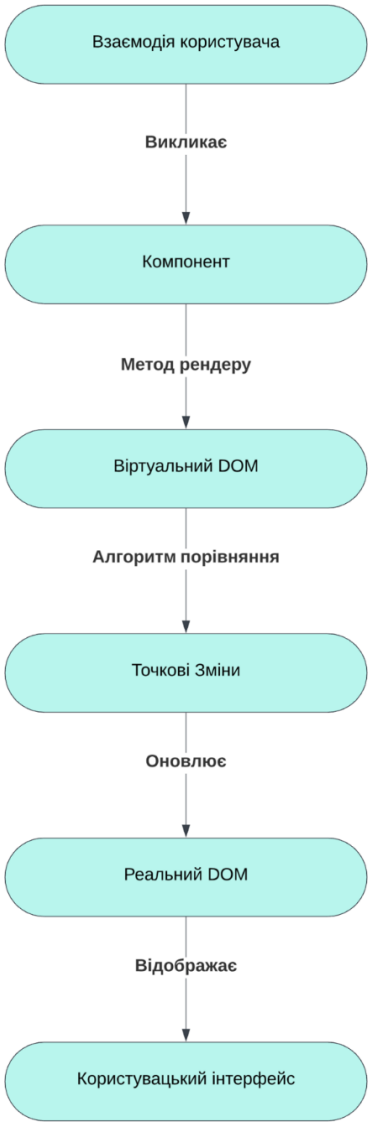


Рисунок 3.2 – Принцип оновлення інтерфейсу за допомогою віртуального DOM у React

3.2 Реалізація серверної частини додатку

Для програмування мікроконтролера ESP32 та підключеного до нього датчика DHT22 використовується спрощений варіант мови C++, відомий як Arduino C++ або просто Arduino Language. Ця мова спеціально адаптована для розробки програм під мікроконтролери й має власні особливості [32].

Найголовніше — програма в Arduino завжди має дві основні частини:

– setup() — функція, яка виконується один раз під час запуску. Саме тут ініціалізуються змінні, налаштовуються порти вводу/виводу, починається зв'язок із сенсорами або іншими пристроями.

– loop() — основний цикл, який виконується безперервно. У ньому описується основна логіка роботи пристрою, що повторюється знову і знову.

Такий підхід дозволяє дуже просто створювати керовану, стабільну програму для пристроїв на основі ESP32.

На рисунку 3.3 представлений приклад простого скетча для ESP32 з підключеним датчиком DHT22. У цьому прикладі показано, як саме реалізується робота обох функцій: setup() налаштовує зв'язок із датчиком, а loop() — щосекунди зчитує і виводить актуальні значення температури й вологості.

```
#include "DHTesp.h"

const int DHT_PIN = 15;
DHTesp dhtSensor;

void setup() {
  Serial.begin(115200);
  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);
}

void loop() {
  TempAndHumidity data = dhtSensor.getTempAndHumidity();
  Serial.println("Температура: " + String(data.temperature, 2) + "°C");
  Serial.println("Вологість: " + String(data.humidity, 1) + "%");
  Serial.println("--");
  delay(1000);
}
```

Рисунок 3.3 – Приклад зчитування та виведення температури й вологості за допомогою ESP32 та датчика DHT22

На рисунку 3.4 представлено приклад результату виконання наведеного вище скрипта. Як видно з виводу у серійному моніторі, система успішно працює: протягом 10 секунд виводяться актуальні показники температури та вологості, які оновлюються щосекунди.

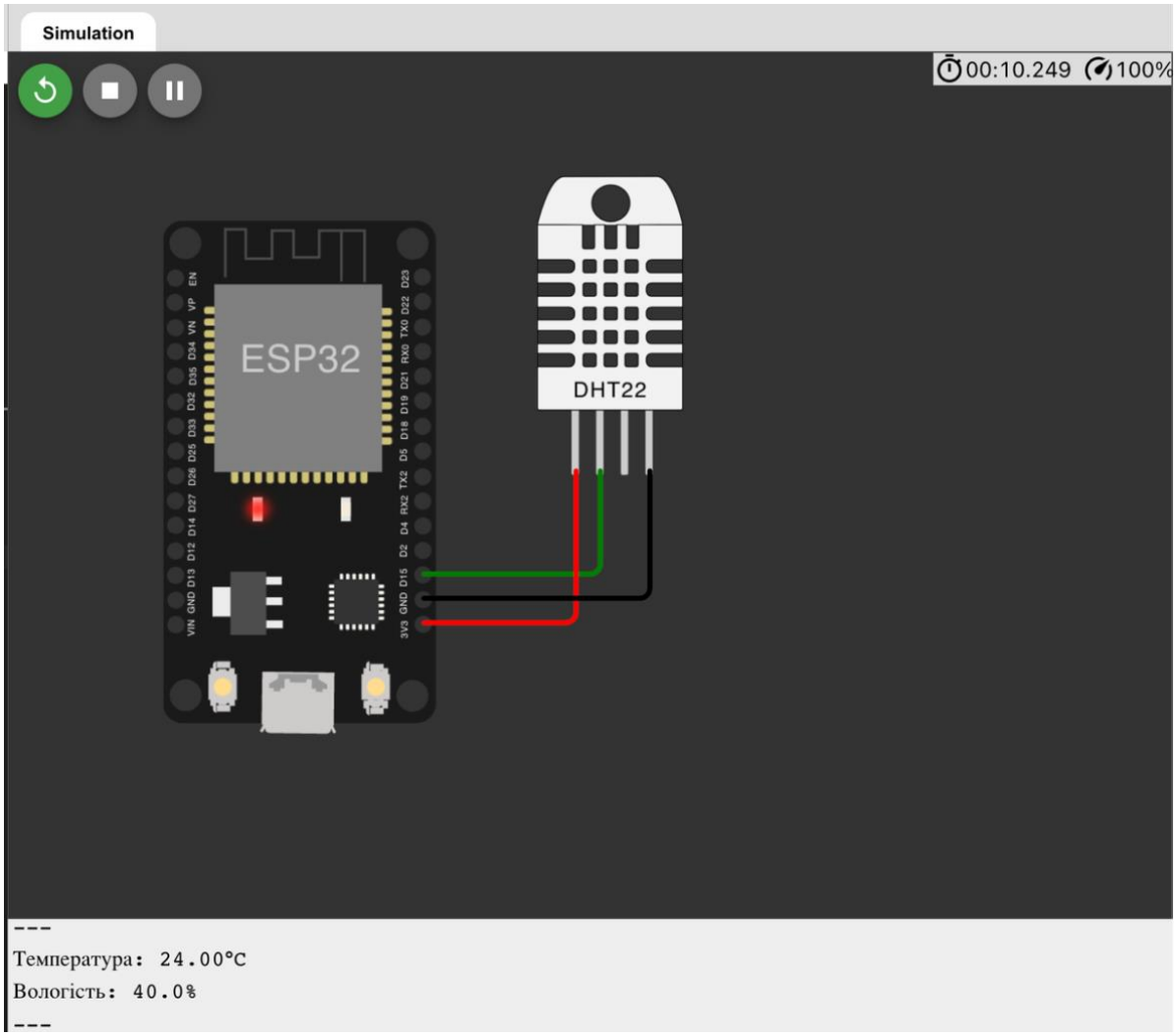


Рисунок 3.4 – Серійний монітор із результатами зчитування температури та вологості з датчика DHT22

Для повноцінної роботи IoT-системи моніторингу необхідно реалізувати скрипт, який щосекунди надсилатиме актуальні дані про стан навколишнього середовища на сервер. Отримана інформація зберігатиметься, оброблятиметься та передаватиметься до клієнтської частини додатку для візуалізації [33].

Першим кроком у створенні такого скрипта є імпорт необхідних бібліотек, які забезпечують зв’язок із сенсорами, обробку часу та передачу даних на сервер.

На рисунку 3.5 наведено приклад відповідного блоку імпорту.

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

Рисунок 3.5 – Імпорт бібліотек для організації зв’язку, зчитування показників та передачі в MQTT-брокер

Наступним кроком є визначення порту мікроконтролера, до якого підключено сенсор DHT22, а також вказання типу сенсора для його подальшої ініціалізації. На рисунку 3.6 наведено відповідний фрагмент коду.

```
#define DHTPIN 4
#define DHTTYPE DHT22
```

Рисунок 3.6 – Ініціалізація змінних для налаштування DHT22 у коді ESP32

Після задання параметрів підключення виконується ініціалізація програмного забезпечення для роботи з датчиком DHT22. На рисунку 3.7 наведено відповідний фрагмент коду.

```
DHT dht(DHTPIN, DHTTYPE);
```

Рисунок 3.7 – Оголошення екземпляра класу DHT для подальшої роботи з сенсором

На наступному етапі визначаються параметри з’єднання з мережею: SSID та пароль Wi-Fi, а також адреса сервера та порт для підключення через протокол MQTT. Відповідний фрагмент коду наведено на рисунку 3.8.

```
const char* ssid       = "your_ssid";
const char* password   = "your_password";
const char* mqttServer = "your-server.com";
const int   mqttPort   = 1883;
```

Рисунок 3.8 – Задання реквізитів підключення до Wi-Fi та MQTT-сервера

На наступному етапі здійснюється ініціалізація протоколу MQTT, який забезпечує з'єднання мікроконтролера з сервером та обмін даними між пристроєм і хмарною інфраструктурою. Приклад ініціалізації наведено на рисунку 3.9.

```
WiFiClient espClient;
PubSubClient client(espClient);
```

Рисунок 3.9 – Ініціалізація клієнта MQTT на основі Wi-Fi-з'єднання

У функції setup() виконується ініціалізація сенсора DHT22, встановлення з'єднання з Wi-Fi мережею та підключення до MQTT-сервера. Відповідний фрагмент програми наведено на рисунку 3.10.

```
void setup() {
  Serial.begin(115200);
  delay(1000);

  dht.begin();

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to the WiFi network");

  client.setServer(mqttServer, mqttPort);
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
```

Рисунок 3.10 – Ініціалізація та налаштування системи

Функція loop() відповідає за обробку всіх подій WebSocket, зчитування даних із сенсора DHT22, формування JSON-об'єкта з актуальними показниками навколишнього середовища та надсилання цього об'єкта клієнту через WebSocket-

з’єднання. На рисунку 3.11 представлено відповідний фрагмент реалізації.

```
void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

    DynamicJsonDocument doc(1024);
    doc["temperature"] = t;
    doc["humidity"] = h;
    String jsonString;
    serializeJson(doc, jsonString);

    client.publish("esp32/data", jsonString.c_str());
  } else {
    Serial.println("WiFi Disconnected");
  }

  delay(1000);
}
```

Рисунок 3.11 – Основний цикл роботи IoT-системи з публікацією даних через MQTT

Серверна частина програмної системи була реалізована з використанням платформи Node.js у поєднанні з мовою TypeScript. Завдяки мультипарадигменності обраних технологій, у розробці було вирішено дотримуватись поєднання кількох підходів — функціонального, об'єктно-орієнтованого та реактивного — з метою повноцінного використання переваг екосистеми JavaScript/TypeScript [34].

Для структуризації серверного коду застосовано принципи гексагональної архітектури та предметно-орієнтованого проєктування (DDD). Такий підхід дозволяє чітко відокремити доменну логіку від інфраструктури, спрощуючи масштабування та підтримку системи.

На рисунку 3.12 показано структуру проєктних папок, яка реалізує ці архітектурні принципи.

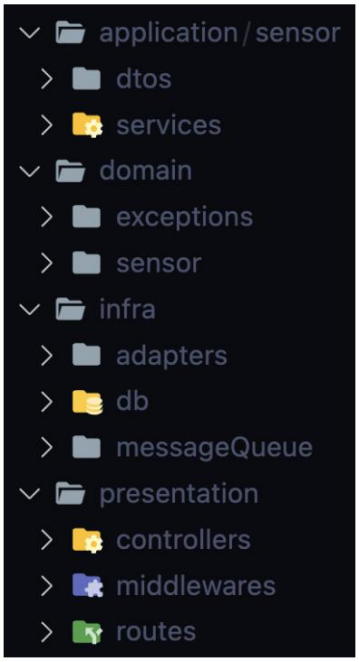


Рисунок 3.12 – Структура серверної частини проєкту згідно з принципами гексагональної архітектури

На рисунку 3.12 представлено основні структурні компоненти серверної частини системи, побудовані відповідно до принципів гексагональної архітектури.

Архітектура поділяється на три ключові шари:

– application — цей шар відповідає за координацію бізнес-логіки. У ньому обробляються вхідні запити, здійснюється трансформація даних у доменні об’єкти, викликаються відповідні методи доменного шару, а також організовується збереження результатів і формування вихідних відповідей;

– domain — центральний елемент архітектури, що містить доменні сутності, їхні властивості, методи, а також правила взаємодії. Саме тут визначається бізнес-логіка, незалежна від інфраструктури та зовнішніх залежностей; infrastructure — шар, який реалізує технічні аспекти системи: роботу з базою даних, мережевими з’єднаннями, API, модулями логування та іншими зовнішніми службами.

– presentation — шар, що відповідає за взаємодію з користувачем. Саме тут організовується відображення даних, отриманих із системи, а також обробка введення користувача. Presentation-шар реалізує API або WebSocket-з'єднання,

через які користувач отримує доступ до функціональності IoT-системи в реальному часі.

Для спрощення розгортання серверної частини додатку було використано бібліотеку Express, яка є одним із найпопулярніших інструментів для створення веб-серверів в екосистемі Node.js. Express забезпечує розвинену систему маршрутизації, що дозволяє легко визначати обробники HTTP-запитів різних типів, а також має вбудовані засоби для формування відповідей у форматі JSON, HTML або текстових файлів та для встановлення відповідних статус-кодів [35].

Серверна частина додатку відповідає за прийом даних від IoT-пристроїв і передачу їх у клієнтську частину через протокол WebSocket. Завдяки WebSocket забезпечується постійне двостороннє з’єднання між клієнтом і сервером, що дозволяє передавати оновлення в режимі реального часу без необхідності періодичних запитів.

Збір даних від пристроїв здійснюється через протокол MQTT. Для взаємодії з брокером MQTT сервер ініціалізує MQTT-клієнта за допомогою бібліотеки mqtt. Архітектура взаємодії побудована за моделлю "публікація–підписка": мікроконтролер виступає в ролі MQTT-клієнта, який відправляє повідомлення з даними про температуру та вологість на MQTT-брокер, у якості якого використовується безкоштовний сервіс EMQ X [35].

Зі свого боку, сервер також виступає MQTT-клієнтом і підписується на відповідні теми повідомлень через брокер. Таким чином, коли пристрій надсилає дані, брокер автоматично передає їх усім клієнтам, що підписані на обрану тему, включно із сервером. Такий механізм забезпечує гнучку та масштабовану обробку повідомлень і дозволяє легко інтегрувати нові пристрої в систему без необхідності значних змін у її архітектурі.

3.3 Організація бази даних та зберігання екологічних даних

На основі аналізу предметної області та розроблених вимог до функціонування системи було сформовано структуру бази даних, яка включає

наступні основні таблиці:

USER (Користувач) — таблиця зберігає інформацію про користувачів системи моніторингу: ім’я, електронну пошту та основний ідентифікатор (user_id). Кожен користувач пов'язаний зі своїм користувацьким профілем.

USER_PROFILE (Профіль користувача) — містить налаштування та вподобання користувачів. Таблиця має унікальний ідентифікатор (profile_id) та зовнішній ключ (user_id) для зв’язку з таблицею користувачів.

DEVICE (Пристрій) — таблиця для обліку IoT-пристроїв, що використовуються у системі. Містить ідентифікатор пристрою, місцезнаходження, тип пристрою (DEVICE_TYPE) та зв’язок із користувачем через зовнішній ключ (user_id) [36].

DEVICE_TYPE (Тип пристрою) — містить перелік можливих типів пристроїв із відповідним унікальним ідентифікатором (device_type_id), назвою та описом.

SENSOR (Датчик) — таблиця з інформацією про сенсори, встановлені на пристроях. Зберігає ідентифікатор датчика, його статус, тип (SENSOR_TYPE) та зв’язок із пристроєм через зовнішній ключ (device_id).

SENSOR_TYPE (Тип датчика) — визначає типи сенсорів, які можуть бути використані у системі. Для кожного типу зберігаються унікальний ідентифікатор (sensor_type_id), назва та опис.

DATA (Дані) — таблиця для зберігання показників, отриманих із датчиків. Містить часову мітку, значення даних, унікальний ідентифікатор (data_id) та посилання на сенсор через зовнішній ключ (sensor_id).

ALERT (Тривога) — таблиця для фіксації повідомлень про тривожні події. Містить унікальний ідентифікатор (alert_id), рівень тривоги, тип тривоги (ALERT_TYPE) та зв’язок із даними через зовнішній ключ (data_id).

ALERT_TYPE (Тип тривоги) — таблиця для класифікації типів тривог, зберігає унікальний ідентифікатор (alert_type_id), назву та опис [37].

3.4 Тестування та оцінка ефективності клієнтської частини розробленого додатку

Клієнтська частина додатку була реалізована за допомогою бібліотеки React, яка забезпечує створення сучасних інтерактивних інтерфейсів. Для організації та управління станом застосунку використано популярну бібліотеку Redux, яка дозволяє централізовано зберігати та обробляти дані. Попри те, що Redux є незалежним від конкретного стеку технологій і може бути використаний у будь-яких JavaScript-проєктах, найбільш поширеним є його використання саме в комбінації з React. Redux особливо ефективний у складних додатках із великою кількістю станів, які потребують синхронного керування та оновлення. Основна концепція Redux базується на тому, що всі дані застосунку зберігаються в єдиному глобальному сховищі (store). Це сховище є єдиною точкою істини для стану додатку, що дозволяє уникати проблем із некоректною синхронізацією даних між різними компонентами [37].

Архітектура Redux базується на трьох основних принципах:

Об'єкт стану доступний лише для читання. Зміни в стані здійснюються лише шляхом створення спеціальних об'єктів (actions), що описують події або зміни, які відбулися. Це гарантує консистентність даних.

Зміна стану виконується за допомогою чистих функцій. Такі функції (reducers) приймають поточний стан і дію, після чого повертають новий стан, не змінюючи вхідні параметри або глобальні об'єкти. Це дозволяє легко передбачити поведінку системи і спростити тестування.

Використання зв'язки React + Redux дозволяє побудувати стабільний, масштабований інтерфейс користувача, який ефективно реагує на зміни даних у реальному часі.

Головна сторінка клієнтського додатку містить кілька ключових віджетів, за допомогою яких користувач взаємодіє із системою моніторингу. Кожен віджет виконує окрему функцію управління або відображення даних у реальному часі.

На рисунку 3.14 показано приклад віджетів для перемикання режимів моніторингу параметрів температури та вологості. Користувач може активувати або деактивувати відображення відповідних показників у залежності від своїх потреб.

Рисунок 3.13 – Інтерактивні віджети для активації моніторингу температури та вологості

На рисунку 3.14 представлено віджет, що відображає список повідомлень, які система надсилає користувачу у випадку погіршення або нормалізації стану навколишнього середовища. Цей елемент інтерфейсу дозволяє оперативно отримувати інформацію про зміну екологічних параметрів у режимі реального часу.

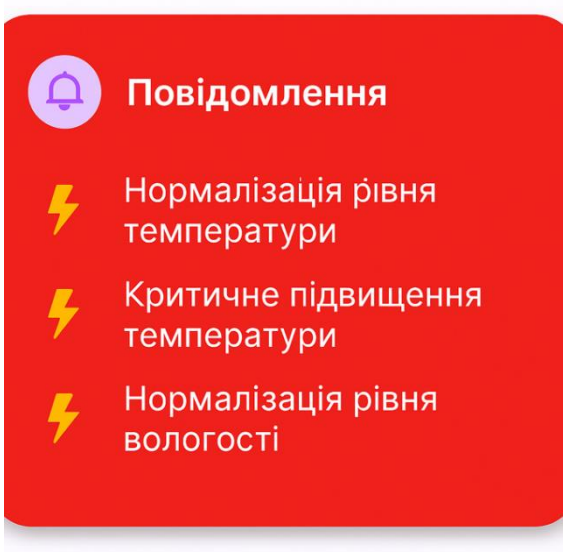


Рисунок 3.14 – Віджет системних сповіщень про стан навколишнього середовища

На рисунку 3.15 зображено віджети, що відображають актуальні значення параметрів навколишнього середовища, які надходять від сенсорів у режимі реального часу. Передача даних здійснюється безпосередньо з датчиків на сервер, де виконується перевірка: у разі зміни показників порівняно з попередніми, система надсилає оновлення до клієнтської частини [38]. Інтерфейс користувача динамічно реагує на ці зміни та миттєво відображає нові значення на відповідних віджетах.

Рисунок 3.15 – Віджети для візуалізації поточних показників сенсорів у режимі реального часу

На рисунку 3.16 представлено загальний вигляд сторінки користувача з інтерактивними віджетами, які забезпечують доступ до основної функціональності системи моніторингу в режимі реального часу.

Рисунок 3.16 – Основна сторінка системи моніторингу з віджетами управління та відображення даних

На екрані налаштувань користувач має можливість встановлювати граничні значення параметрів навколишнього середовища. У разі перевищення встановлених меж система автоматично надсилає відповідне повідомлення користувачу. Приклад інтерфейсу для налаштування цих параметрів наведено на рисунку 3.17.

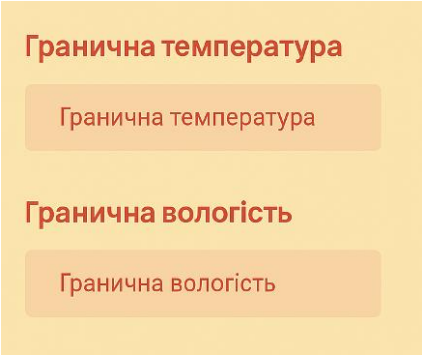


Рисунок 3.17 – Інтерфейс встановлення граничних значень для моніторингу параметрів середовища

На рисунку 3.18 представлено інтерфейс сторінки налаштувань, який дозволяє користувачу змінювати параметри моніторингу та персоналізувати роботу системи відповідно до власних вимог.

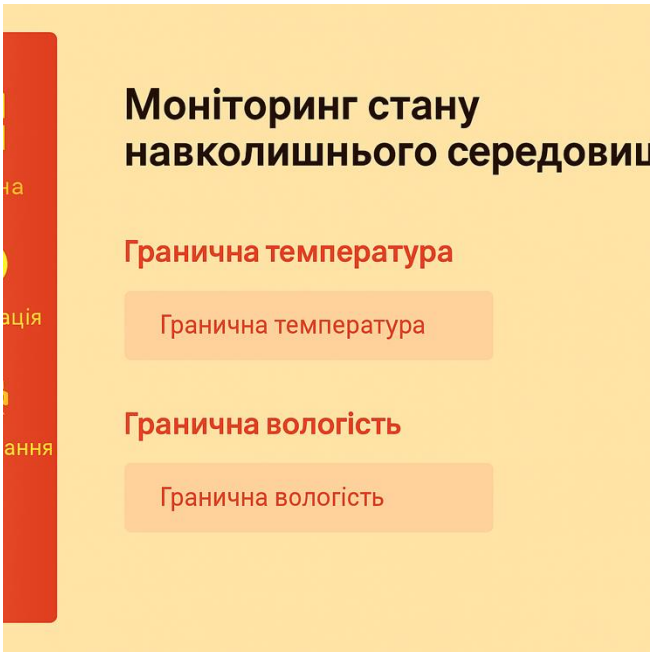


Рисунок 3.18 – Інтерфейс сторінки налаштувань користувача в системі моніторингу

ВИСНОВКИ

У результаті виконаної дипломної роботи було комплексно досліджено та реалізовано систему моніторингу екологічного стану на базі мікроконтролера ESP32 з використанням технологій Інтернету речей (IoT). У першому розділі проведено теоретичний аналіз сучасних систем екологічного моніторингу, здійснено порівняння існуючих IoT-рішень та визначено вимоги до створюваної системи. Обґрунтовано вибір апаратного й програмного забезпечення, що забезпечує надійність, енергоефективність та відповідність поставленим задачам.

У другому розділі було спроєктовано архітектуру IoT-системи, включно з побудовою загальної схеми взаємодії компонентів, вибором сенсорів (зокрема DHT22) та обґрунтуванням використання мікроконтролера ESP32. Особливу увагу приділено проєктуванню програмної архітектури та вимогам до інтерфейсу користувача, що забезпечують зручність у використанні системи в реальних умовах.

У третьому розділі реалізовано повноцінну систему моніторингу, що включає серверну частину на базі Node.js з використанням протоколу MQTT, організовано зберігання екологічних даних у структурованій базі даних, а також створено клієнтський інтерфейс із застосуванням React та Redux. Система успішно реалізує двосторонню взаємодію між пристроями, сервером та користувачем у режимі реального часу.

Було проведено тестування функціональних модулів системи, перевірено ефективність збору, обробки та візуалізації даних. Аналіз результатів підтвердив стабільну роботу пристроїв, точність зчитування показників та надійність передачі інформації між компонентами системи. Візуальні віджети та повідомлення про перевищення допустимих значень екологічних параметрів забезпечують користувача своєчасною інформацією.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бондаренко О. І., Бутко М. М. Інтернет речей: концепції, технології, рішення. — Київ : Наукова думка, 2021. — 288 с.

2. Гуржій А. М., Дудкін М. О. Інформаційні технології: навч. посіб. — Київ : КНЕУ, 2020. — 204 с.

3. Мельник А. О. Інформаційні системи в управлінні. — Львів : ЛНУ ім. І. Франка, 2021. — 264 с.

4. Олійник А. М., Пархоменко О. О. Мікроконтролери в системах автоматизації: навч. посіб. — Харків : ХНУРЕ, 2020. — 170 с.

5. Слєпцов А. В., Черняк М. Ю. Програмування мікроконтролерів на мові С: навч. посіб. — Дніпро : ДНУ, 2021. — 192 с.

6. Шевченко С. А. Основи проєктування комп’ютерних систем : підруч. — Одеса : ОНУ ім. І.І. Мечникова, 2022. — 304 с.

7. Стратегія кібербезпеки України на 2021–2025 роки [Електронний ресурс]. — Режим доступу: https://www.rnbo.gov.ua (дата звернення: 15.04.2025).

8. Державна служба спеціального зв’язку та захисту інформації України. Національна стратегія розвитку ІТ-інфраструктури до 2030 року [Електронний ресурс]. — Режим доступу: https://cip.gov.ua (дата звернення: 18.04.2025).

9. Методичні рекомендації з побудови інформаційних систем управління [Електронний ресурс]. — Режим доступу: https://it-edu.gov.ua/docs (дата звернення: 20.04.2025).

10. Закон України «Про основні засади забезпечення кібербезпеки України». — № 2163-VIII від 05.10.2017 р. — Відомості Верховної Ради України. — 2017. — № 45. — Ст. 408.

11. Ashton K. That ‘Internet of Things’ Thing // RFID Journal. — 2009. — [Electronic resource]. — Available at: https://www.rfidjournal.com/articles/view?4986 (accessed: 14.04.2025).

12. McEwen A., Cassimally H. Designing the Internet of Things. — Chichester : Wiley, 2014. — 256 p.

13. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications // IEEE Communications Surveys & Tutorials. — 2015. — Vol. 17, No. 4. — P. 2347–2376.

14. Sharma V. K., Chaurasiya R. C., Jain S. Architecting scalable IoT systems: a review on methodologies, challenges and open research issues // Journal of Systems Architecture. — 2020. — Vol. 107. — Article 101730.

15. Tan L., Wang N. Future Internet: The Internet of Things // 3rd International Conference on Advanced Computer Theory and Engineering. — IEEE, 2010. — P. 376–380.

16. Ray P. A survey on Internet of Things architectures // Journal of King Saud University — Computer and Information Sciences. — 2016. — Vol. 30, No. 3. — P. 291–319.

17. Banerjee A., Sheth A. The Internet of Things: A Survey from the Data-Centric Perspective // Springer Journal of Data Intelligence. — 2016. — Vol. 2. — P. 115–130.

18. NodeMCU Documentation [Electronic resource]. — Available at: https://nodemcu.readthedocs.io (accessed: 17.04.2025).

19. Espressif Systems. ESP32 Series Datasheet [Electronic resource]. — Available at: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (accessed: 16.04.2025).

20. OpenWeatherMap API Documentation [Electronic resource]. — Available at: https://openweathermap.org/api (accessed: 19.04.2025).

21. IoT Analytics [Електронний ресурс]. – Режим доступу до ресурсу: https://iot-analytics.com/

22. IoT For All [Електронний ресурс]. – Режим доступу до ресурсу: https://www.iotforall.com/

23. Arduino [Електронний ресурс]. – Режим доступу до ресурсу: https://www.arduino.cc/

24. Raspberry Pi [Електронний ресурс]. – Режим доступу до ресурсу: https://www.raspberrypi.org/

25. LoRa Alliance [Електронний ресурс]. – Режим доступу до ресурсу: https://lora-alliance.org/

26. IEEE Xplore Digital Library [Електронний ресурс]. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/

27. Elsevier [Електронний ресурс]. – Режим доступу до ресурсу: https://www.elsevier.com/

28. Coursera. Internet of Things Specialization [Електронний ресурс]. – Режим доступу до ресурсу: https://www.coursera.org/specializations/internet-of-things

29. Random Nerd Tutorials. ESP8266 NodeMCU with Arduino IDE [Електронний ресурс]. – Режим доступу до ресурсу: https://randomnerdtutorials.com/getting-started-with-esp8266-wifitransceiver-review/

30. GreatScott! Arduino Tutorial - How to use a Buzzer [Відео]. YouTube, 19 березня 2014 р

31. Rui Santos. How to Use the ESP8266-01 WiFi Module with the Arduino UNO [Електронний ресурс]. Random Nerd Tutorials, 21 липня 2016 р

32. Chaudhryl V. Arduair: Air Quality Monitoring. International Journal of Environmental Engineering and Management. 2013. P. 639-646.

33. Zhanga H., Wangb Sh., Haob J., Wangc X., Wanga Sh., Chaia F., Lid M. Air pollution and control action in Beijing. Journal of Cleaner Production. Vol. 112. Part 2. 2016. P. 1519-1527.

34. Okokpujie K., Noma-Osaghae E., Modupe O., John S., Oluwatosin O. A smart air pollution monitoring system. International Journal of Civil Engineering and Technology. Vol. 9. 2018. P. 799–809.

35. Parmar G., Lakhani S., Chattopadhyay M. An IoT based low cost air pollution monitoring system. International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE). 2017. P. 524-528.

36. Raspberry Pi Downloads – Software for the Raspberry Pi. https://www.raspberrypi.org/downloads.../ (дата звернення: 24.04.2025).

37. Ларіоник Р.В., Луцик Н.С. Комп’ютерна система для дистанційного контролю якості атмосферного повітря. Актуальні задачі сучасних технологій : збірник тез доповідей X міжнародної науково-технічної конференції молодих учених та студентів. Тернопіль: ФОП Паляниця В. А. 2021. С. 100.

38. Vasylkivskyi I., Ishchenko V., Pohrebennyk V., Palamar M., Palamar A. System of water objects pollution monitoring. International Multidisciplinary Scientific GeoConference Surveying Geology and Mining Ecology Management (SGEM 2017). 2017. Vol. 17, No. 33. P. 355-362.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Державний університет інформаційно-комунікаційних технологій

Кафедра інформаційних систем та технологій

Кваліфікаційна робота на тему:

«РОЗРОБКА IOT СИСТЕМИ МОНІТОРИНГУ ЕКОЛОГІЧНОГО СТАНУ МІСТА З ВИКОРИСТАННЯМ МОДУЛЮ ESP-32»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: ІЛЮК Андрій ІСД-42
Науковий керівник роботи: СЕНЬКОВ Олег

КИЇВ - 2025

Актуальність дослідження. Сучасні міста стикаються з численними екологічними викликами: забруднення повітря, зміни мікроклімату, підвищення рівня вуглекислого газу тощо. Ефективний моніторинг стану довкілля є важливою умовою забезпечення якості життя населення та своєчасного реагування на потенційні загрози. Водночас традиційні системи контролю часто є дорогими, маломасштабними та не забезпечують оперативного збору даних у реальному часі.

Мета роботи – розробити та реалізувати ефективну IoT-систему для моніторингу екологічного стану міста, що дозволяє автоматизовано збирати, зберігати та візуалізувати дані про параметри довкілля, а також провести оцінку її ефективності та працездатності.

Об'єкт дослідження – процеси моніторингу екологічного стану міських територій з використанням IoT-технологій.

Предмет дослідження – архітектура, принципи побудови та методи реалізації IoT-систем для збору, обробки та відображення екологічних даних на базі мікроконтролера ESP32.

Завдання:

- проаналізувати сучасні підходи до екологічного моніторингу із використанням IoT-технологій та дослідити можливості ESP32 і сумісних сенсорів;
- сформувати вимоги до архітектури IoT-системи для екологічного моніторингу в міських умовах;
- розробити апаратну та серверну частину системи з використанням ESP32 і реалізувати збирання, обробку та збереження екологічних даних;
- створити інтерфейс користувача для візуалізації даних, провести тестування системи та оцінити її ефективність;

Рисунок 1.1 – Загальна схема роботи системи моніторингу довкілля

Системи моніторингу довкілля
Призначення:
•Виявлення небезпечних змін у стані атмосфери, води, ґрунтів;
•Попередження екологічних катастроф;
•Інформування влади, підприємств і громадськості;
•Формування баз для екологічного аналізу та прогнозування;
•Підтримка сталого розвитку.
Принципи роботи:
1.Збір даних — сенсори фіксують температуру, вологість, рівень забруднень тощо;
2.Передача — дані надсилаються через Wi-Fi, GSM, LoRa тощо;
3.Обробка — фільтрація, аналіз, виявлення відхилень;
4.Збереження — дані зберігаються в БД;
5.Візуалізація — графіки, карти, звіти;
6.Реагування — сповіщення та запуск протоколів дій.

4

Порівняння існуючих IoT-рішень для екологічного моніторингу

Рисунок 1.2 – Пристрій для моніторингу AirVisual Node

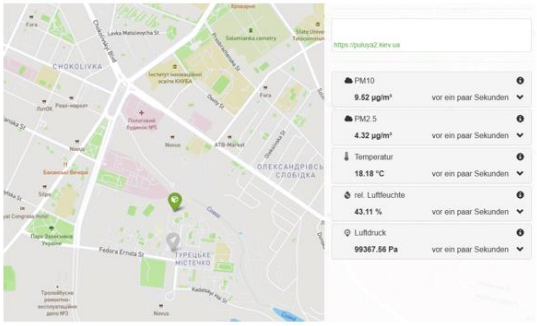


Рисунок 1.3 – OpenSenseMap відкрита онлайн-платформа

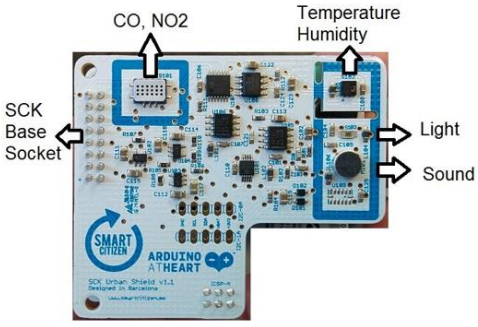


Рисунок 1.4 – Плата Smart Citizen Kit

Постановка задачі та визначення вимог до системи

5

Створення системи для збору, передачі, збереження та візуалізації даних про стан довкілля на базі ESP32 та сенсорів.

Функції системи:

•Збір даних: CO_2, температура, вологість, запиленість;

•Передача на сервер у реальному часі;

•Збереження в БД, формування звітів;

•Веб-інтерфейс з графіками й таблицями;

•Повідомлення при перевищенні екопорогів.

Вимоги до системи:

•**Автономність** (енергоощадність, режим сну ESP32);

•**Точність** сенсорів згідно зі стандартами;

•**Стабільна передача** (Wi-Fi/LoRa, буферизація);

•**Захист даних** (ідентифікатор, API-ключ);

•**Зручний інтерфейс** (адаптація під ПК/мобільні).

Підбір апаратних засобів для вимірювання екологічних параметрів

6

Рисунок 1.5 – Модуль MQ-135 якості повітря

Рисунок 1.6 – Датчик вологи та температури DHT22

Рисунок 1.7 – Датчик пилу GP2Y1010AU0F

Гнучкість:
Архітектура має адаптуватися до змін вимог і технологій без втрати стабільності.
Ключові характеристики:
•**Масштабованість** – стабільна робота при збільшенні кількості сенсорів та обсягу даних.
•**Відмовостійкість** – автоматичне відновлення після збоїв, резервування.
•**Безпека** – шифрування, автентифікація, захист від атак.
•**Модульність** – поділ системи на незалежні блоки з чіткими інтерфейсами.
•**Ефективність** – оптимізація ресурсів: CPU, пам'яті, мережі, БД.

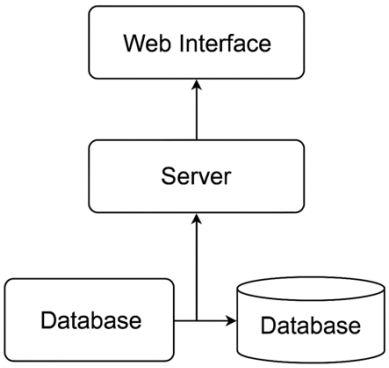


Рисунок 2.8 – Архітектура програмного забезпечення IoT-системи

Вибір стеку технологій для розробки IoT-системи

Рисунок 3.1 – Схема обробки запитів у подійно-орієнтованій архітектурі Node.js

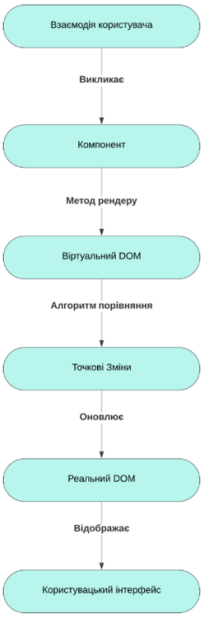


Рисунок 3.2 – Принцип оновлення інтерфейсу за допомогою віртуального DOM у React

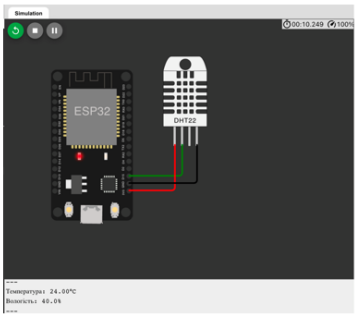
Реалізація серверної частини додатку

```
#include "DHTesp.h"

const int DHT_PIN = 15;
DHTesp dhtSensor;

void setup() {
  Serial.begin(115200);
  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);
}

void loop() {
  TempAndHumidity data = dhtSensor.getTempAndHumidity();
  Serial.println("Температура: " + String(data.temperature, 2) + "°C");
  Serial.println("Вологість: " + String(data.humidity, 1) + "%");
  Serial.println("---");
  delay(1000);
}
```



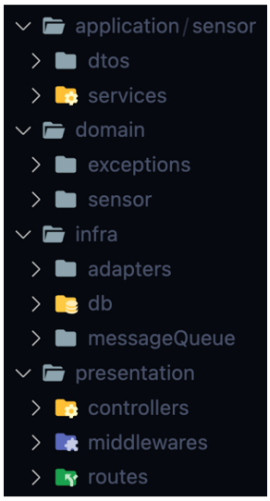

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <DHT.h>
```

```
DHT dht(DHTPIN, DHTTYPE);
```

```
const char* ssid       = "your_ssid";
const char* password   = "your_password";
const char* mqttServer = "your-server.com";
const int   mqttPort   = 1883;
```

```
WiFiClient espClient;
PubSubClient client(espClient);
```

```
#define DHTPIN 4
#define DHTTYPE DHT22
```




```
void setup() {
  Serial.begin(115200);
  delay(1000);

  dht.begin();

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to the WiFi network");

  client.setServer(mqttServer, mqttPort);
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
}
```

```
void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

    DynamicJsonDocument doc(1024);
    doc["temperature"] = t;
    doc["humidity"] = h;
    String jsonString;
    serializeJson(doc, jsonString);

    client.publish("esp32/data", jsonString.c_str());
  } else {
    Serial.println("WiFi Disconnected");
  }

  delay(1000);
}
```

Тестування та оцінка ефективності клієнтської частини розробленого додатку

Рисунок 3.13 – Інтерактивні віджети для активації моніторингу температури та вологості

Рисунок 3.17 – Інтерфейс встановлення граничних значень для моніторингу параметрів середовища

Рисунок 3.15 – Віджети для візуалізації поточних показників сенсорів у режимі реального часу

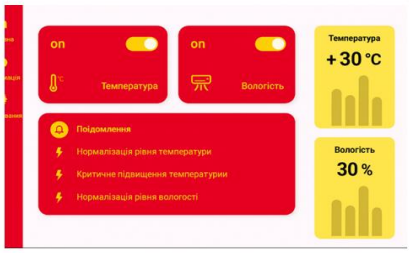


Рисунок 3.16 – Основна сторінка системи моніторингу з віджетами управління та відображення даних

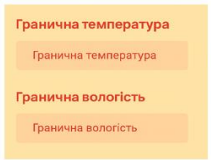


Рисунок 3.14 – Віджет системних сповіщень про стан навколишнього середовища

Рисунок 3.18 – Інтерфейс сторінки налаштувань користувача в системі моніторингу

ВИСНОВКИ

11

- Проведено аналіз існуючих IoT-рішень для екологічного моніторингу та визначено вимоги до системи.
- Обґрунтовано вибір апаратного (ESP32, DHT22) та програмного забезпечення з урахуванням надійності й енергоефективності.
- Спроєктовано архітектуру IoT-системи: сенсори, мікроконтролер, сервер, клієнт.
- Реалізовано серверну частину на Node.js з використанням MQTT та бази даних.
- Розроблено адаптивний веб-інтерфейс на React + Redux для перегляду екологічних показників.
- Проведено тестування системи: перевірено точність сенсорів, стабільність передачі даних і візуалізацію.
- Система успішно функціонує в реальному часі, надаючи користувачеві сповіщення про критичні зміни в довкіллі.

Апробація

Основні положення і результати бакалаврської роботи
доповідались на:
- VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», ДУІКТ, 2025.

Дякую за увагу!
Доповідь закінчено