НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних систем, мереж та кібербезпеки

_____ Касаткін Д.Ю., к.пед.н., доц.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2024 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: « Розробка підсистеми накопичення даних у системі моніторингу парникових газів »

Спеціальність 123 «Комп'ютерна інженерія»

Гарант освітньої програми: _____ / Нікітенко Є. В. /

підпис ПІБ

Керівник дипломного проекту: _____ / Смолій В.В. /

підпис ПІБ

Виконав: _____ / Єжик А. О. /

підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.п.н., доц. /
підпис ПІБ, вчене звання і ступінь
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТУ

Єжик Артем Олегович
(прізвище, ім'я, по батькові)
Спеціальність (напрям підготовки): комп'ютерна інженерія

Тема кваліфікаційної бакалаврської роботи: «Розробка підсистеми накопичення даних у системі моніторингу парникових газів»

затверджена наказом ректора НУБіП України від "16" __12__2024р. №2251 "С"

Термін подання завершеної роботи на кафедру

Вихідні дані до кваліфікаційної бакалаврської роботи
LoRa/ESP32 як комунікаційне середовище
PostgreSQL як основа для централізованої бази даних
Система моніторингу парникових газів

Перелік питань, що підлягають розробці:

1. Аналіз вимог до підсистеми накопичення даних в умовах автономного моніторингу
2. Розробка структури централізованого сховища даних на основі PostgreSQL.
3. Розробка інтерфейсу для перегляду даних та їх аналізу

Перелік графічного матеріалу (за потреби)

Дата видачі завдання "_16_" ____грудня_____2024 р.

Керівник бакалаврської роботи _____ _Смолій В.В., д.т.н., професор_
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання _____ _Єжик А. О._____
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	26.04.2025 р.	Виконано
2	Проектування системи	03.05.2025 р.	Виконано
3	Реалізація системи	10.05.2025 р.	Виконано
4	Тестування системи	20.05.2025 р.	Виконано
5	Оформлення пояснювальної записки	26.05.2025 р.	Виконано
6	Оформлення графічного матеріалу	27.05.2025 р.	Виконано

Студент _____ **А. О. Єжик**

(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____ **В. В. Смолій**

(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 65 сторінок, 15 рисунків, 1 таблиця, 2 лістинга, 1 додаток, 14 джерел.

КОМП’ЮТЕРНА СИСТЕМА, ПІДСИСТЕМА НАКОПИЧЕННЯ ДАНИХ, POSTGRESQL

Об’єкт аналізу – процес процес збирання, буферизації, передавання та збереження екологічних даних у розподіленій інформаційній системі.

Мета роботи – розроблення підсистеми накопичення даних у системі моніторингу парникових газів.

Проект складається з чотирьох розділів.

Перший розділ присвячено аналізу предметної області, екологічних задач та сучасних технологій моніторингу навколишнього середовища.

У другому розділі розглянуто вибір апаратного і програмного забезпечення, описано структуру підсистеми та обґрунтовано архітектурні рішення.

У третьому розділі реалізовано серверну частину на Flask, створено REST API та базу даних PostgreSQL, реалізовано збереження даних на SD-карту.

У четвертому розділі проведено тестування системи, оцінено її ефективність, надійність та можливості масштабування.

У результаті було розроблено працездатну, автономну, масштабовану підсистему накопичення даних, придатну до використання в реальних умовах екологічного моніторингу.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

CO_2 — вуглекислий газ

CH_4 — метан

ESP32 — мікроконтролер компанії Espressif Systems

Wi-Fi — Wireless Fidelity (бездротова мережа стандарту IEEE 802.11)

LoRa — Long Range (технологія бездротової передачі даних)

LoRaWAN — Long Range Wide Area Network (протокол, що працює поверх LoRa)

UART — універсальний асинхронний приймач-передавач

I2C — Inter-Integrated Circuit (послідовний інтерфейс для підключення периферійних пристроїв)

SPI — Serial Peripheral Interface (послідовний периферійний інтерфейс)

OTA — Over-The-Air (оновлення прошивки через бездротовий канал)

REST API — Representational State Transfer Application Programming Interface (інтерфейс програмування, що дотримується REST-принципів)

SD — Secure Digital (формат картки пам’яті)

PostgreSQL — об’єктно-реляційна система керування базами даних

СУБД — система управління базами даних

API — Application Programming Interface (інтерфейс програмування застосунків)

IoT — Internet of Things (Інтернет речей)

5G / LTE — покоління мобільного зв’язку (п’яте покоління / Long-Term Evolution)

Flask — мікрофреймворк для веб-розробки мовою Python

ВСТУП

Однією з найактуальніших глобальних екологічних проблем сьогодення є зміна клімату, зумовлена підвищенням концентрації парникових газів у атмосфері. Найбільшу частку серед основних парникових газів займає CO_2, джерелами якого є спалювання викопного палива, промислове виробництво, транспорт та інші антропогенні процеси.

Для ефективного впровадження заходів щодо зменшення викидів парникових газів необхідне точне та систематичне їх вимірювання. Такі системи повинні бути здатні до тривалої автономної роботи, збору та накопичення великих обсягів екологічних даних з різних географічних регіонів, з подальшою передачею до центральних баз даних для аналізу та прийняття рішень.

У структурі такої системи важливу роль відіграє підсистема накопичення даних, яка забезпечує буферизацію, збереження та передачу отриманих значень у разі наявності зв’язку. В умовах нестабільного покриття або повної його відсутності критично важливою є здатність системи до локального зберігання інформації, перевірки її цілісності та гарантованого доставлення у централізоване сховище після відновлення з’єднання.

Актуальність обраної теми полягає в необхідності створення надійних, масштабованих та енергоефективних рішень для забезпечення збереження екологічних даних в умовах автономної роботи сенсорних систем.

Метою даної дипломної роботи є проєктування та реалізація підсистеми накопичення даних у складі системи моніторингу парникових газів, яка забезпечує збирання, зберігання, перевірку та передачу екологічних даних з сенсорної мережі до централізованої бази даних.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Загальна характеристика системи моніторингу парникових газів

Система моніторингу парникових газів — це комплекс технічних і програмних засобів, призначених для автоматизованого збору, накопичення, передавання та аналізу екологічних параметрів, пов’язаних з викидами парникових газів, зокрема CO_2 та CH_4. Такі системи дозволяють здійснювати спостереження за концентрацією шкідливих газів у повітрі в реальному часі, що є важливою умовою для оцінки поточного стану атмосфери та прийняття ефективних управлінських рішень у сфері екології.

До основних компонентів системи входять:

- Сенсорна мережа, яка включає різноманітні датчики (CO_2, температури, вологості, тиску, вітру тощо), розміщені у визначених географічних точках.
- Підсистема накопичення даних, яка зберігає дані локально у разі відсутності зв’язку, а при його наявності — передає дані до центрального сховища.
- Комунікаційна підсистема, яка забезпечує передачу даних до серверів через мережі LoRa та Wi-Fi
- Централізоване сховище де зберігаються всі отримані дані для подальшого аналізу, я буду використовувати PostgreSQL.
- WEB-інтерфейс, яке забезпечує візуалізацію даних, аналіз трендів, створення звітів та доступ до історичних записів.

Такий підхід дозволяє реалізувати масштабовану, енергоефективну та автономну систему моніторингу, здатну працювати в різноманітних кліматичних умовах та забезпечувати достовірність зібраних даних навіть при перебоях у зв’язку чи електроживленні.

1.2 Аналіз параметрів, що підлягають моніторингу

У процесі моніторингу парникових газів надзвичайно важливо забезпечити не лише вимірювання концентрації вуглекислого газу (CO_2), але й комплексний облік супутніх параметрів навколишнього середовища, які можуть істотно впливати на точність отриманих результатів та подальший аналіз. Одним із ключових аспектів є геолокаційні дані, зокрема географічна широта, довгота та висота над рівнем моря. Вони дозволяють точно визначити місце збору даних, враховувати особливості рельєфу, кліматичні умови та інші просторові фактори, які впливають на поширення газів в атмосфері. Висота над рівнем моря, наприклад, може суттєво впливати на атмосферний тиск і, відповідно, на поведінку газів, зокрема їхню концентрацію та щільність.

Не менш важливим є фіксація точної дати та часу проведення вимірювання, що дає змогу здійснювати порівняння даних у динаміці, аналізувати добові, сезонні та річні зміни, а також синхронізувати показники з іншими джерелами інформації. Температура повітря має значний вплив на фізико-хімічні властивості газів: зі зміною температури змінюється і об’єм, і щільність CO_2, що потребує корекції значень, отриманих від сенсорів. Відносна вологість також відіграє важливу роль, оскільки вона може впливати на стабільність та точність роботи сенсорів, особливо при використанні їх в умовах відкритого середовища або в регіонах із підвищеною вологістю.

Окрему увагу слід приділяти таким параметрам, як швидкість і напрямок вітру. Ці показники дозволяють оцінити переміщення повітряних мас, розрахувати ймовірні джерела викидів та спрогнозувати подальше поширення парникових газів у просторі. Напрямок вітру дає змогу з’ясувати, з якого боку надходять повітряні потоки, а швидкість дозволяє визначити, з якою інтенсивністю відбувається змішування повітря та розсіювання CO_2.

Основним вимірюваним показником є концентрація вуглекислого газу, яка фіксується спеціалізованими сенсорами, такими як MH-Z19C або аналогічними. Цей параметр є критично важливим для оцінки рівня парникового ефекту та розрахунку екологічного навантаження на певну територію. Однак сам по собі рівень CO_2 не дає повної картини, якщо не враховуються умови, в яких він вимірювався. Саме тому необхідно інтегрувати всі вищезазначені фактори в єдину систему збору даних.

Для досягнення високої точності, надійності та енергоефективності системи моніторингу використовується технологія LoRa — малопотужний, але далекобійний бездротовий зв’язок, що дає змогу передавати дані з важкодоступних або віддалених територій (наприклад, поля, лісові ділянки, сільськогосподарські об'єкти). LoRa дозволяє надсилати короткі пакети інформації, що містять усі необхідні параметри, із мінімальними витратами енергії, що особливо важливо для систем, які працюють автономно та живляться від акумуляторів або сонячних панелей.

Таким чином, ефективний моніторинг стану атмосфери передбачає створення інтегрованої системи, яка здатна забезпечити регулярне, синхронне, багатофакторне вимірювання параметрів довкілля. Це вимагає точного каліброваного обладнання, стабільного каналу передачі даних, механізмів синхронізації часу, а також спеціалізованого програмного забезпечення для обробки, зберігання та візуалізації отриманої інформації. Лише комплексний підхід дає змогу формувати повну картину екологічного стану певної місцевості, виявляти потенційні загрози та ухвалювати обґрунтовані управлінські рішення щодо захисту довкілля.

1.3 Аналіз вимог до апаратної частини сенсорної системи

Апаратна частина сенсорної системи моніторингу парникових газів є критично важливою для забезпечення точного, стабільного та безперебійного збору даних в умовах тривалого автономного функціонування. Вона має відповідати ряду вимог, зумовлених як технічними, так і експлуатаційними особливостями:

1. Захищеність та придатність до зовнішнього використання

- Виконання у польовому форм-факторі з відповідним захистом від пилу, дощу, снігу та ультрафіолету.
- Мінімальний клас захисту — IP65.
- Температурна стійкість в межах від -40°C до +125°C, що дозволяє експлуатувати обладнання у широкому кліматичному діапазоні.

2. Сенсори

- Датчик CO_2 — високоточний, з компенсацією температури та вологості - MH-Z19C, який працює в діапазоні 0–5000 ppm.
- Датчики температури та вологості — SHT31, з вбудованою цифровою передачею даних.
- Датчик швидкості та напрямку вітру — ультразвукового типу.
- Модуль GPS — для точної прив’язки до координат.

3. Обчислювальний модуль (контролер)

- ESP32 — оптимальний варіант: має вбудований Wi-Fi, Bluetooth, підтримує mesh-мережі, низьке енергоспоживання.
- Модуль RTC (Real Time Clock) для точної синхронізації часу.

4. Живлення

- Li-Po акумулятори з контролером заряду.
- Сонячна панель для підзарядки батареї у денний час, розрахована на добове споживання + запас.
- Наявність контролера живлення для переходу у сплячий режим та енергоефективної роботи.

5. Засоби зберігання даних

- Карта пам’яті (microSD) для збереження даних у разі втрати зв’язку.

- Буфер накопичення повинен забезпечувати збереження щонайменше 7 діб даних без втрат.

6. Засоби зв’язку

- Mesh-зв’язок на базі Wi-Fi та LoRa — дозволяє створювати розгалужену мережу з передачею даних через проміжні вузли.
- Bluetooth — для локального конфігурування та початкового налаштування.

З огляду на ці вимоги, система повинна бути надійною, масштабованою та здатною до тривалої автономної роботи з гарантованою точністю збирання екологічних даних.

Таблиця 1.3.1 - Дані які збирає один девайс та відповідні поля для данин у БД

№	Назва параметра	Поле в БД	Тип даних	Джерело / сенсор	Примітки
1	Device ID (Ідентифікатор вузла)	sensor_id	INTEGER (FK)	Прошивка	Для ідентифікації пристрою
2	Географічна широта	lat	DOUBLE PRECISION	GPS-модуль	У десятковому форматі
3	Географічна довгота	lon	DOUBLE PRECISION	GPS-модуль	У десятковому форматі
4	Висота над рівнем моря	altitude	REAL	GPS-модуль	Точність до ±10 м
5	Дата і час вимірювання	timestamp	TIMESTAMPTZ	RTC-модуль	Формат ISO 8601, UTC
7	Температура повітря	temperature	REAL	SHT31	З температурною компенсацією
8	Відносна вологість	humidity	REAL	SHT31	Точність ±2% RH
9	Концентрація CO_2	co2_ppm	REAL	MH-Z19C	Оптичний NDIR сенсор

10	Швидкість вітру	wind_speed	REAL	Анемометр	Ультразвуковий або обертовий

Продовження таблиці 1.3.1

№	Назва параметра	Поле в БД	Тип даних	Джерело / сенсор	Примітки
11	Напрямок вітру	wind_dir	REAL	Флюгер	За азимутом, 0° — Північ

1.4 Вимоги до накопичення та збереження даних

Програмне забезпечення підсистеми накопичення даних виконує критичну роль посередника між сенсорною частиною системи та центральним сховищем, забезпечуючи фіксацію, тимчасове зберігання, обробку, перевірку цілісності та передачу екологічних даних. Воно повинно забезпечувати безперервне зчитування даних з CO_2-сенсорів, а також з датчиків температури, вологості, швидкості й напрямку вітру та геопозиціювання. Всі зібрані записи мають бути точно синхронізовані з часом і координатами місцевості завдяки інтеграції з модулями RTC і GPS. У разі втрати зв’язку система зобов’язана тимчасово буферизувати дані, зберігаючи їх локально на microSD-карті до моменту відновлення зв’язку з центральним сховищем, після чого має автоматично здійснювати передачу накопичених пакетів. Формат переданих даних повинен бути уніфікованим, зокрема у форматі JSON із можливістю серіалізації, щоб забезпечити сумісність з подальшою обробкою. Програмне забезпечення має гарантувати стабільність роботи, включно з обробкою помилок при розривах з’єднання, пошкоджених файлів або порушенні послідовності записів. У випадках критичних збоїв має бути реалізований механізм автоматичного перезапуску пристрою або сервісу, а всі події повинні фіксуватись у журналі активності

— від моменту ввімкнення до процесів з’єднання і передачі даних. З міркувань безпеки необхідно реалізувати шифрування як збережених, так і переданих даних, а також забезпечити аутентифікацію вузлів при спробі з’єднання з сервером. Доступ до налаштувань повинен бути обмежений — наприклад, за допомогою Bluetooth лише при наявності пароля чи цифрового ключа. У перспективі важливо, щоб система була масштабованою та здатною до оновлення: додавання нових типів сенсорів не повинно потребувати переписування усього коду, а можливість віддаленого оновлення прошивки (OTA) повинна бути доступна для всіх компонентів системи. Для ефективної інтеграції з основною СУБД структура і формат даних мають бути повністю сумісними з PostgreSQL, що дозволяє автоматизувати подальший аналіз, візуалізацію, агрегацію та довготривале збереження інформації. Усі ці вимоги закладають основу для створення надійного, захищеного та гнучкого програмного забезпечення, яке здатне забезпечити ефективну роботу розподіленої системи екологічного моніторингу в умовах автономного функціонування.

1.5 Постановка завдань дипломної роботи

Метою цієї дипломної роботи є розробка підсистеми накопичення даних у складі системи моніторингу парникових газів, що дозволяє здійснювати автоматизований збір, збереження та передачу екологічно важливої інформації — передусім про концентрацію вуглекислого газу (CO_2), температуру, вологість повітря, напрямок і швидкість вітру, а також географічне положення сенсорів. Така система має підтримувати автономність роботи, надійність передачі даних, масштабованість та інтеграцію з центральним сховищем на основі СУБД.

У процесі реалізації даного проєкту необхідно вирішити низку практичних завдань. Передусім — це розробка архітектури системи накопичення, яка включає локальні сенсорні вузли з обмеженими обчислювальними ресурсами та централізований сервер або сховище для зберігання й обробки великих обсягів даних. Система повинна працювати у складних кліматичних умовах, що зумовлює вибір відповідного обладнання з промисловими характеристиками.

Одним з основних завдань є вибір надійного способу збереження даних на рівні сенсора, враховуючи можливі перерви в зв'язку. Локальне буферизування даних із наступною синхронізацією при відновленні з'єднання має бути реалізовано максимально просто і надійно. Також критично важливим є вибір комунікаційної технології, яка дозволить передавати інформацію на значні відстані без значного енергоспоживання — з урахуванням порівняння варіантів було прийнято рішення на користь використання LoRa у комбінації з мікроконтролерами ESP32.

Наступним завданням є проєктування структури централізованого сховища даних. Вибір зроблено на користь реляційної бази даних PostgreSQL, яка дозволяє ефективно зберігати, індексувати й опрацьовувати великі обсяги інформації з сенсорів у вигляді часових рядів. Таке сховище також забезпечує можливість інтеграції з веб-інтерфейсом для візуалізації даних і формування звітів.

Важливою складовою є також забезпечення захисту даних та відмовостійкості: необхідно реалізувати механізми повторної передачі, перевірки цілісності, обробки збоїв під час запису і передачі, а також ведення журналів подій. Система повинна бути здатна виявляти та фіксувати нестандартну поведінку сенсорів, невідповідність очікуваним параметрам, або втрату з'єднання з вузлами.

Таким чином, завданням дипломної роботи є повний цикл проєктування та часткової реалізації підсистеми накопичення даних: від вибору архітектури та компонентів до проєктування способів зберігання і

передачі інформації, із забезпеченням надійності, масштабованості та відповідності екологічним вимогам.

2 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

2.1 Вибір технічних засобів контролю та збору даних

Одним з ключових аспектів побудови ефективної підсистеми накопичення даних у системі моніторингу парникових газів є вибір надійної та енергоефективної технології для передачі даних з польових сенсорних вузлів до центрального сховища. Від правильного вибору комунікаційної технології залежить стабільність зв’язку, тривалість автономної роботи пристроїв, відстань між модулями, здатність до масштабування та загальна вартість впровадження.

Серед основних варіантів для передавання даних у подібних розподілених екологічних системах виділяють наступні: 5G/LTE-зв’язок, LoRa/LoRaWAN, а також Wi-Fi та ESP-NOW (на базі ESP32).

Сучасні мобільні мережі, такі як LTE та 5G, забезпечують високу пропускну здатність і стабільний зв’язок на великих територіях за умови наявності покриття. Основною перевагою цих рішень є здатність працювати без проміжних вузлів (ретрансляторів) і пряме підключення до інтернету. Проте такі технології мають і суттєві недоліки: по-перше, вартість модемів та SIM-зв’язку для кожного сенсорного вузла є доволі високою; по-друге, енергоспоживання модулів LTE/5G значне, що критично при використанні автономного живлення. Крім того, в сільській місцевості або в гірських регіонах покриття може бути нестабільним або відсутнім.

Іншим поширеним підходом у системах з низьким енергоспоживанням є використання технології LoRa (Long Range) або її мережевого розширення LoRaWAN. LoRa дозволяє передавати дані на відстань до 10-15 км у сільській місцевості, при цьому споживаючи дуже мало енергії. Основною перевагою є можливість побудови власної приватної мережі, що не залежить від мобільного оператора. Сенсорні вузли можуть об’єднуватися у "mesh" або зіркоподібну топологію навколо базової станції, яка вже передає дані до

центрального сервера через інтернет. Недоліками є обмежена пропускна здатність (що, однак, не є критичним для періодичного передавання невеликих пакетів екологічних даних) та потреба у налаштуванні власної інфраструктури — принаймні однієї LoRa-базової станції для кожної зони покриття.

Ще один варіант — використання ESP32 з Wi-Fi або ESP-NOW. Такий підхід може бути ефективним у випадках, коли система моніторингу встановлюється в межах об’єкту, який вже має стабільне Wi-Fi-покриття (наприклад, у промислових умовах, теплицях, лабораторіях). ESP32 має вбудовані засоби Wi-Fi, Bluetooth та можливість реалізації зв’язку ESP-NOW — протоколу прямої передачі даних між ESP-пристроями без доступу до роутера. Це дає змогу створити локальну бездротову мережу з низьким енергоспоживанням і швидкою реакцією. Основним обмеженням є коротша відстань передачі (до 100 м на відкритому просторі), тому така технологія не підходить для розподілених польових систем великої площі.

Після порівняльного аналізу варіантів було прийнято рішення на користь поєднання ESP32 як базової апаратної платформи з комунікаційним модулем LoRa. Такий вибір обґрунтовано кількома причинами. По-перше, ESP32 — це потужний і доступний мікроконтролер з низьким енергоспоживанням, великою спільнотою підтримки та можливістю гнучкого програмного налаштування. По-друге, комбінація з LoRa-модулем забезпечує надійний бездротовий зв’язок на великих відстанях, що є критичним для польових сенсорів. При цьому забезпечується тривала автономна робота вузлів — до кількох тижнів або навіть місяців на одному заряді, особливо у поєднанні з сонячними панелями та енергозберігаючими режимами.

Щодо зберігання даних, на кожному сенсорному вузлі передбачається наявність локального сховища — microSD-карти для збереження даних при втраті зв’язку. Це дозволяє уникнути втрати даних у разі збою мережі або енергопостачання. На рівні центрального сервера буде використовуватися централізована база даних PostgreSQL, яка добре підходить для роботи з

великими обсягами часових даних, має потужні засоби фільтрації, агрегації та інтеграції з аналітичними інструментами та візуалізацією.

Таким чином, обрана архітектура зв’язку та зберігання даних оптимально поєднує низьке енергоспоживання, достатню дальність передачі, гнучкість конфігурації та масштабованість, що робить її придатною для розгортання в реальних екологічних умовах.

2.2 Вибір способів збереження та передачі даних

У підсистемі накопичення даних особливу увагу необхідно приділити не лише збору інформації від сенсорних пристроїв, але й способам її ефективного збереження та подальшої передачі. З огляду на вимоги до надійності, автономності та відмовостійкості, доцільно передбачити дворівневу архітектуру збереження — локальну (на самому сенсорному вузлі) та централізовану (на віддаленому сервері або в хмарі). Передача даних, у свою чергу, має здійснюватися таким чином, щоб мінімізувати енергоспоживання, забезпечити захист від втрати інформації, та підтримувати масштабованість системи.

На першому рівні, тобто на сенсорному вузлі, необхідно реалізувати локальне тимчасове збереження даних. Це дозволяє забезпечити буферизацію інформації у випадку відсутності зв’язку з мережею або центральним вузлом, уникнувши втрат у даних. Найбільш поширеними варіантами є використання флеш-пам’яті мікроконтролера, EEPROM або зовнішніх носіїв, таких як microSD-карти. Перевагою microSD є велика ємність за невисоку ціну та простота реалізації файлової системи, однак для надійної роботи потрібне контролювання енергоживлення та обробка збоїв при записі. Вбудована флеш-пам’ять ESP32 є меншою за обсягом, але дозволяє швидко і стабільно зберігати критичні дані, які згодом будуть передані. У випадку обмеженого

обсягу локального сховища, застосовується циклічний запис із видаленням найстаріших записів після підтвердження отримання нових даних сервером.

На другому рівні — централізованому, дані накопичуються у серверному сховищі, що дозволяє здійснювати довготривале зберігання, аналіз, фільтрацію та візуалізацію. Як основну платформу для організації централізованого збереження обрано реляційну систему керування базами даних PostgreSQL. PostgreSQL має потужну підтримку часових рядів, добре масштабується та дозволяє створювати ефективні індекси, що критично для обробки великої кількості точок даних від різних сенсорів у часовій послідовності. Крім того, вона має широкі можливості інтеграції з веб-інтерфейсами та аналітичними інструментами Python/NumPy.

Для забезпечення стабільної та безпечної передачі даних з сенсорів до централізованої бази використовуються LoRa-з'єднання між сенсорними вузлами та шлюзами, а далі — передача через інтернет (Wi-Fi) до серверного середовища. LoRa дозволяє передавати дані на великі відстані, навіть у складних географічних умовах, що є ключовою перевагою в екологічних системах моніторингу. Кожен сенсорний вузол періодично надсилає інформацію про концентрацію CO_2, температуру, вологість, координати та час зйому даних. За відсутності зв’язку, дані залишаються у локальному буфері, звідки вони автоматично передаються до шлюзу, щойно з’єднання буде відновлено.

На стороні шлюзу реалізується черга обробки отриманих пакетів, фільтрація на повтори та валідація цілісності. Для забезпечення захисту даних при передачі можна використовувати прості методи шифрування або обмеження доступу через аутентифікацію пристроїв.

Для відображення стану системи, а також для забезпечення зручного доступу до накопичених показників, планується реалізація веб-інтерфейсу, який працюватиме у зв’язці з серверною базою даних. Це дозволяє не лише здійснювати візуальний контроль даних, але й управляти системою —

наприклад, надсилати запити на повторне зчитування даних, виконання тесту сенсорів або очищення буфера.

У перспективі можливо додатково реалізувати резервне копіювання центральної бази даних у хмарне сховище, а також організувати передачу зведених звітів або сповіщень (наприклад, за допомогою електронної пошти або Telegram-ботів) у разі перевищення допустимих порогів концентрації CO_2.

Таким чином, побудована система збереження та передачі даних базується на принципах відмовостійкості, енергоефективності та гнучкості масштабування, що дозволяє їй ефективно функціонувати в умовах розподіленої сенсорної мережі, працювати автономно при перебоях зв'язку та зберігати критично важливу екологічну інформацію у довготривалому форматі.

2.3 Розробка логічної структури підсистеми

Логічна структура підсистеми накопичення даних у системі моніторингу парникових газів визначає спосіб взаємодії основних її компонентів, а також потік даних — від моменту їх отримання сенсорами до надсилання у центральне сховище. Правильно спроєктована логіка дозволяє забезпечити стабільність, надійність і масштабованість системи, а також спрощує її подальший супровід і модернізацію.

У центрі логічної моделі — сенсорний вузол. Кожен з таких вузлів включає мікроконтролер ESP32 підключений до сенсорів вимірювання парникових газів (зокрема CO_2, температури, вологості, вітру та ін.). Мікроконтролер виконує періодичне опитування датчиків відповідно до заданого графіку (кожну годину), формує структурований запис даних з додаванням мітки часу та географічних координат. Також контролер веде

локальний буфер пам’яті, де зберігає зібрані дані у разі втрати з’єднання з мережею.

Рисунок 2.3.1 - ESP32

Для комунікації між сенсорними вузлами та базовою станцією використовується LoRa — технологія, що дозволяє передавати невеликі обсяги даних на великі відстані з мінімальним енергоспоживанням. За потреби вузли можуть утворювати mesh-мережу, пересилаючи дані один одному, доки вони не досягнуть базової станції з доступом до централізованої інфраструктури.

Базова станція — це проміжна ланка, яка приймає дані з багатьох сенсорних вузлів, виконує їх перевірку на цілісність, можливе попереднє агрегування, а також забезпечує їх пересилання до центрального серверу за допомогою високошвидкісних технологій зв’язку - Wi-Fi. Саме тут реалізуються механізми повторної передачі у разі збоїв, а також первинний журнал подій.

Центральний сервер функціонує як головне сховище даних та логічне ядро всієї системи. Він базується на системі керування базами даних PostgreSQL, яка дозволяє організовувати збереження даних у вигляді часових рядів, пов’язаних з конкретними сенсорними вузлами, мітками часу, просторовими координатами. На сервері також розгортається бекенд для API-запитів, а також веб-інтерфейс, через який користувачі можуть переглядати, аналізувати або експортувати дані.

Усі частини системи пов’язані логічно: сенсорний вузол → LoRa-мережа → базова станція → інтернет-з’єднання → центральний сервер → інтерфейс користувача. Така логіка дозволяє підтримувати як точкове встановлення (одиничні сенсори), так і великомасштабні розгортання з десятками або сотнями вузлів. Кожен компонент працює незалежно, але узгоджено з іншими завдяки синхронізації часу, уніфікованому формату даних та стандартам взаємодії.

Особливістю цієї структури є стійкість до відмов: у разі втрати з’єднання дані тимчасово зберігаються локально, а після відновлення — автоматично передаються без втрати цілісності. Крім того, структура дозволяє адаптуватися до змін у кількості сенсорів чи умов роботи без необхідності суттєвого перепроєктування.

Загалом, логічна структура підсистеми є ключовим елементом, який забезпечує ефективне функціонування всієї системи моніторингу, об'єднуючи в єдине ціле сенсори, канали зв’язку, обчислювальні ресурси та інтерфейси користувача.

2.4 Визначення структури централізованого сховища

Централізоване сховище є ключовим компонентом підсистеми накопичення даних, оскільки саме воно забезпечує довгострокове, структуроване та захищене збереження всієї інформації, яка надходить із сенсорних вузлів. Його правильне проєктування дозволяє ефективно працювати з великими обсягами даних, швидко здійснювати пошук, аналіз і візуалізацію інформації, а також гарантувати цілісність і збереження важливих екологічних показників.

З урахуванням характеру даних (переважно часові ряди, отримані з великої кількості вузлів) та необхідності в просторовому аналізі, доцільним є

використання СУБД PostgreSQL з розширеннями PostGIS (для просторових даних) та TimescaleDB (для оптимізації роботи з часовими рядами).

Основною одиницею зберігання є запис сенсорних даних, який містить такі поля (див. Таблиця 1.3.1):

- sensor_id — ідентифікатор сенсорного вузла, з якого надійшли дані;
- timestamp — мітка часу, що фіксує момент збору даних;
- co2 — концентрація CO_2 (в ppm);
- temperature — температура повітря (в градусах Цельсія);
- humidity — відносна вологість (%);
- wind_speed — швидкість вітру (м/с);
- wind_dir — напрям вітру (в градусах);
- lat та lon — координати місця вимірювання;
- altitude — висота над рівнем моря (м);

Дані передбачається зберігати постійно, з можливістю архівування за допомогою політик розділення за часовими періодами (наприклад, партиціювання по місяцях або роках), що підтримується в TimescaleDB.

Додатковою перевагою структури є готовність до інтеграції з візуалізаційними платформами — такими як **Grafana** або спеціалізовані веб-інтерфейси, що дозволяють аналізувати просторово-часові закономірності розподілу CO_2 та інших факторів.

Така структура централізованого сховища забезпечує гнучкість, масштабованість, високу продуктивність і зручність в адмініструванні системи збору екологічних даних.

2.5 Розробка структури бази даних для зберігання результатів моніторингу

Для забезпечення ефективного зберігання, пошуку та обробки даних, що надходять у процесі моніторингу парникових газів, необхідно створити реляційну базу даних, структура якої логічно відображає процес збору інформації з сенсорних вузлів. Під час проєктування цієї структури враховуються ключові вимоги до системи, зокрема необхідність підтримки високої швидкості запису та читання даних у режимі реального часу, що критично важливо для стабільної роботи у середовищі з великою кількістю сенсорів. База даних повинна бути легко масштабованою, аби забезпечити адаптацію до зростання кількості пристроїв та обсягу інформації. Також важливою є можливість виконання просторово-часового аналізу — кожен запис повинен бути прив’язаний до точних географічних координат та часу вимірювання, що дозволяє здійснювати глибокий аналіз змін у розподілі концентрацій парникових газів у просторі та динаміці. Система має бути сумісною з сучасними інструментами для візуалізації та аналітики, що спрощує інтерпретацію даних, створення звітів і прийняття рішень. Окрім того, передбачається стійкість до втрат зв’язку з окремими вузлами: база даних має забезпечити цілісність і послідовність даних, навіть якщо передача здійснюється з часовим запізненням після відновлення зв’язку. Усі ці аспекти є критично важливими для побудови надійної інфраструктури збирання й аналізу екологічної інформації.

Для реалізації бази даних у межах системи було обрано PostgreSQL — одну з найпотужніших, надійних і масштабованих систем управління базами даних з відкритим вихідним кодом, яка забезпечує високу продуктивність, підтримку складних запитів, розширену функціональність для роботи з просторовими та часовими даними, а також активну спільноту розробників і широку сумісність із сучасними інструментами аналітики та візуалізації.

Вона підтримує розширення, які критично важливі в контексті даного проєкту.

Базова схема БД містить ключову таблицю measurements яка зберігає основні результати моніторингу, див. Таблиця 1.3.1.

Завдяки використанню TimescaleDB, таблиця measurements може бути розділена на гіпертаблиці, що дозволяє автоматично партиціювати дані за часовими або іншими параметрами, що особливо корисно при обробці великих масивів даних з високою частотою збору.

Такий підхід до структурування бази даних дозволяє ефективно управляти інформацією, пов’язаною з моніторингом парникових газів, забезпечує надійність, масштабованість і підтримку для подальшого розвитку системи, включаючи інтеграцію з аналітичними та візуалізаційними платформами.

3 РЕАЛІЗАЦІЯ ПІДСИСТЕМИ НАКОПИЧЕННЯ ДАНИХ

3.1 Реалізація логіки збирання, буферизації та передачі даних

Після створення сенсорної системи, яка забезпечує фіксацію екологічних параметрів, критично важливим є розроблення програмної логіки, яка керує процесами зчитування, обробки, тимчасового збереження (буферизації) та подальшої передачі даних до централізованого сховища. Це є основним функціональним ядром підсистеми накопичення даних, від якого залежить її надійність, точність і стійкість до збоїв у зв'язку.

На цьому етапі було визначено, що основна обробка даних виконуватиметься на мікроконтролері ESP32, який забезпечує високу обчислювальну здатність, має розширені можливості для підключення периферійних пристроїв (через інтерфейси I2C, UART, SPI), та підтримує бездротові технології зв’язку (Wi-Fi, Bluetooth, LoRa). Його використання дозволяє одночасно виконувати зчитування сенсорів, обробку отриманих значень, ведення буфера даних та ініціацію передачі інформації в залежності від стану мережі.

Збір екологічних даних у системі реалізовано шляхом періодичного опитування сенсорних вузлів із заданим інтервалом часу. Після кожного циклу зчитування формується уніфікована структура даних, яка містить ключову інформацію про стан навколишнього середовища. До цієї структури входять такі параметри: унікальний ідентифікатор пристрою (Device ID), географічні координати місця проведення вимірювання (широта та довгота), висота над рівнем моря, а також точна дата і час фіксації. Крім цього, збираються значення температури повітря, відносної вологості, концентрації вуглекислого газу (CO_2), швидкості вітру та його напрямку. Всі ці показники інтегруються в єдиний об’єкт і перетворюються у формат JSON, що забезпечує зручність подальшої передачі даних на сервер, збереження у базі або обробку на стороні клієнта. Такий підхід дозволяє підтримувати гнучку,

масштабовану та стандартизовану систему збору і обміну даними між різними компонентами моніторингової інфраструктури.

Буферизація є важливою функцією для забезпечення стійкості до збоїв. Якщо пристрій втрачає з’єднання з мережею (наприклад, при слабкому сигналі або відключенні базової станції), зібрані дані не губляться, а тимчасово зберігаються у внутрішній пам’яті ESP32 (наприклад, в оперативній пам’яті або у файловій системі LittleFS/EEPROM).

Коли зв’язок відновлюється, пристрій автоматично ініціює передачу накопичених даних на сервер. Це дозволяє гарантувати повноту записів та зменшити ризики втрати інформації в умовах нестабільної інфраструктури або при використанні малопотужних каналів зв’язку (LoRa).

Для передачі даних у проєкті використовується технологія LoRa, яка забезпечує енергоефективний зв’язок на великих відстанях. Такий підхід дозволяє охоплювати значні території, зокрема сільськогосподарські об’єкти або віддалені станції, передаючи короткі пакети з необхідною інформацією.

Цей формат забезпечує зручність збереження у базі даних та подальшу обробку для візуалізації, аналітики або інтеграції з іншими підсистемами.

Таким чином, розроблена логіка забезпечує гнучкий, надійний та масштабований підхід до збору і передачі екологічних даних з віддалених сенсорних пристроїв. Буферизація та умовна передача дозволяють мінімізувати втрати інформації та адаптувати систему до будь-яких сценаріїв розгортання, включаючи роботу в польових умовах. У наступному розділі буде описано реалізацію збереження даних у випадку відсутності зв'язку.

3.2 Реалізація збереження даних у випадку відсутності зв'язку

Для забезпечення безперервного збору та збереження даних в умовах можливих збоїв зв’язку у проєктованій системі реалізовано багаторівневий механізм збереження інформації на локальних носіях. Загальна архітектура системи складається з масиву сенсорних вузлів на базі мікроконтролера ESP32, кожен з яких підключений до набору сенсорів для вимірювання уже згаданих вище фізичних параметрів. Зібрані дані на кожному вузлі формуються у структурований пакет у форматі JSON, що дозволяє уніфікувати обмін даними між вузлами системи. Після формування JSON-пакету вузол здійснює спробу передати його на централізований приймальний вузол за допомогою радіозв’язку LoRa, який забезпечує бездротову передачу даних на великі відстані при низькому енергоспоживанні. У разі успішної передачі пакет даних надсилається далі, а сенсорний вузол переходить у режим очікування до наступного циклу вимірювання. Якщо ж передача неможлива через втрату зв’язку або збої у роботі LoRa-модуля, сформований JSON-пакет записується у файл на SD-карті, підключеній до ESP32. Завдяки цьому забезпечується локальне збереження інформації безпосередньо на сенсорному вузлі, що дозволяє уникнути втрати даних навіть у разі тривалого відключення зв’язку. Централізований приймальний вузол також побудований на основі ESP32 і виконує функції шлюзу між LoRa-мережею та сервером через Wi-Fi-з’єднання. Він постійно очікує надходження пакетів даних від сенсорних вузлів через LoRa, приймає ці пакети у форматі JSON та виконує спробу їх передачі на сервер за допомогою HTTP POST-запиту на визначену адресу API. У разі наявності активного мережевого підключення і успішного підтвердження прийому даних сервером вузол переходить до очікування наступного пакету. Якщо ж Wi-Fi-з’єднання відсутнє або сервер не приймає дані (наприклад, у разі відсутності відповіді чи помилки HTTP-запиту), прийнятий JSON-пакет зберігається на SD-карті приймального вузла. Таким чином, на обох рівнях системи реалізовано резервне збереження даних на SD-карті у випадку втрати зв’язку, що дозволяє уникнути втрат інформації

навіть в умовах нестабільного мережевого середовища. Застосування такого механізму дозволяє підвищити стійкість та надійність роботи системи, а також забезпечити повне збереження усіх отриманих показників для подальшої синхронізації після відновлення зв’язку. Додатковою перевагою є використання єдиного формату JSON для пакування даних, що значно спрощує їх подальшу обробку, архівування та інтеграцію з серверними додатками, базами даних та іншими інформаційними системами. Запропоноване рішення є універсальним, легко масштабується шляхом додавання нових сенсорних вузлів без необхідності зміни програмної логіки або серверної частини, а також дозволяє адаптувати систему до роботи з різними типами сенсорів та фізичних параметрів, залишаючи структуру обміну та збереження даних сталою.

Для реалізації механізму збереження даних у випадку відсутності зв’язку у програмному коді сенсорного вузла передбачено кілька основних функціональних блоків. Після зчитування показників з сенсорів температура та вологість зберігаються у відповідні змінні.

Далі ці значення упаковуються у формат JSON за допомогою бібліотеки ArduinoJson, що дозволяє сформувати структурований пакет даних для подальшої передачі або збереження. Формування пакету виглядає наступним чином:

```
StaticJsonDocument<200> doc;
doc["temperature"] = temperature;
doc["humidity"] = humidity;
doc["timestamp"] = millis();
String jsonData;
serializeJson(doc, jsonData);
```

Після формування JSON-пакету здійснюється спроба передати його через модуль LoRa на централізований вузол. Передача відбувається за допомогою функцій:

```
LoRa.beginPacket();
```

```
LoRa.print(jsonData);
LoRa.endPacket();
```

Після надсилання перевіряється статус зв'язку або наявність підтвердження прийому. У разі, якщо передача неможлива (наприклад, у випадку відсутності LoRa-з’єднання), дані записуються на SD-карту. Для цього у коді використовується бібліотека SD.h, яка дозволяє працювати з файловою системою FAT. Перед початком роботи ініціалізується SD-карта:

```
if (!SD.begin(5)) {
  Serial.println("Card Mount Failed");
  return;
}
```

У випадку відсутності зв’язку з LoRa, сформований JSON-пакет записується у файл на SD-карті за допомогою наступного блоку коду:

```
File file = SD.open("/data_log.txt", FILE_APPEND);
if (file) {
  file.println(jsonData);
  file.close();
} else {
  Serial.println("Failed to open file for writing");
}
```

Аналогічний механізм реалізовано і на централізованому приймальному вузлі. Після отримання пакету через LoRa:

```
String receivedData = LoRa.readString();
```

Здійснюється спроба передати його на сервер за допомогою Wi-Fi та HTTP POST-запиту. Для цього використовується бібліотека HTTPClient.h, яка дозволяє надсилати запити за вказаною адресою:

```
HTTPClient http;
http.begin("http://server-address/api/data");
http.addHeader("Content-Type", "application/json");
int httpResponseCode = http.POST(receivedData);
```

У разі, якщо HTTP-запит не виконується успішно, у разі відсутності мережевого підключення чи помилки на сервері, отримані дані також зберігаються на SD-карті централізованого вузла за допомогою аналогічного блоку:

```
if (httpResponseCode <= 0) {
  File file = SD.open("/data_log.txt", FILE_APPEND);
  if (file) {
    file.println(receivedData);
    file.close();
  }
}
```

Таким чином, у програмному коді передбачено два незалежні механізми збереження даних: на сенсорному вузлі — у випадку невдалої передачі через LoRa, та на приймальному вузлі — у випадку неможливості відправки на сервер через Wi-Fi. В обох випадках дані зберігаються у текстовому файлі на SD-карті у форматі JSON, що дозволяє легко їх обробити або передати повторно після відновлення зв’язку. Це дозволяє гарантувати збереження усієї зібраної інформації в умовах нестабільного або відсутнього зв’язку на будь-якому з етапів роботи системи.

Підключення SD-карти до ESP-32 показано на рисунку 3.2.1. Для такого підключення було використано онлайн сервіс Wokwi

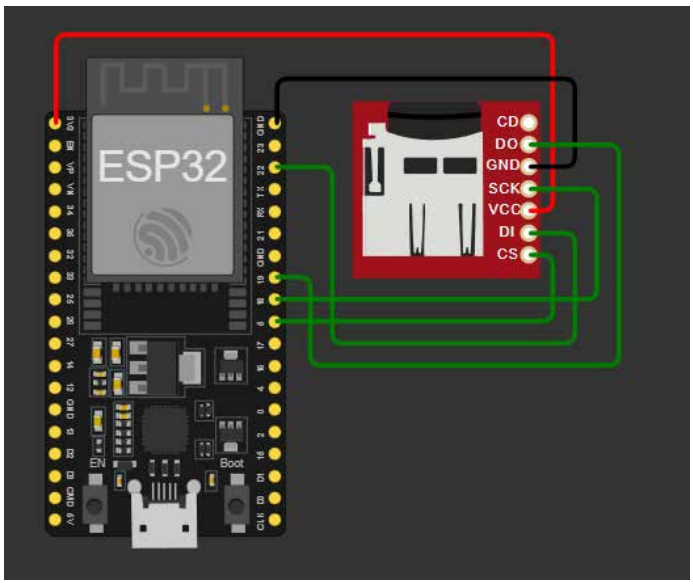


Рисунок 3.2.1 - Підключення SD-карти до ESP-32

Застосований модуль SD-карти, підключений до мікроконтролера ESP32 за допомогою SPI-інтерфейсу. Модуль SD-карти має сім основних контактів: CD (Card Detect) — контакт виявлення наявності карти, DO (Data Out) або MISO — лінія передачі даних від SD-карти до мікроконтролера, GND (Ground) — загальний контакт (земля), SCK (Serial Clock) — тактова лінія, VCC — лінія живлення, DI (Data In) або MOSI — лінія передачі даних від мікроконтролера до SD-карти, та CS (Chip Select) — лінія вибору пристрою.

У даній реалізації використовуються лише необхідні для роботи SPI-інтерфейсу контакти: VCC, GND, DO, DI, SCK та CS. Контакт CD залишено непідключеним, оскільки в цій системі виявлення наявності карти не використовується.

Живлення модуля SD-карти здійснюється від контакту **3.3V** плати ESP32, який з’єднано з контактом VCC модуля. Загальний контакт GND підключено до відповідного контакту GND на ESP32. Передача даних реалізована за допомогою наступних з’єднань: контакт CS модуля SD-карти підключено до GPIO5 (D5) ESP32, контакт SCK — до GPIO18 (D18), контакт DI (MOSI) — до GPIO23 (D23), а контакт DO (MISO) — до GPIO19 (D19).

Таке підключення дозволяє ESP32 взаємодіяти з SD-картою за стандартним SPI-протоколом. Ініціалізація карти в програмному коді

виконується за допомогою бібліотеки SD.h, де під час виклику функції SD.begin() вказується номер GPIO-порту, до якого підключено лінію CS. Зчитування та запис даних реалізується через файлову систему FAT, що дозволяє створювати, відкривати, дописувати та читати файли у зручному для обробки форматі JSON.

3.3 Реалізація сервера прийому та збереження даних

Одним із ключових етапів реалізації підсистеми накопичення даних є розгортання сервера, який забезпечує прийом інформації від сенсорної системи, її обробку та збереження у базі даних. З огляду на ціль дипломної роботи — створити працездатну архітектуру системи моніторингу парникових газів — серверна частина має бути стабільною, легко масштабованою та відкритою до розширення.

Вибір технологій

Для реалізації серверної частини програмного забезпечення було обрано комбінацію сучасних, перевірених та взаємодоповнюючих технологій, що забезпечують надійність, гнучкість та масштабованість рішення. Нижче наведено детальний огляд обраних інструментів:

- **PostgreSQL** — це потужна, надійна та функціонально багата система керування реляційними базами даних з відкритим вихідним кодом. Вона активно використовується в різноманітних галузях для зберігання та обробки великих обсягів структурованих даних. PostgreSQL підтримує розширену роботу з транзакціями, складні запити, процедури та тригери, що дозволяє ефективно організувати зберігання даних для системи моніторингу. Крім того, PostgreSQL легко

масштабується та забезпечує високу продуктивність навіть за умов активного зростання обсягів інформації.

- **Python** — універсальна, сучасна та гнучка мова програмування, яка має велику кількість бібліотек і фреймворків для вирішення найрізноманітніших задач. Python дозволяє швидко та якісно реалізовувати серверну логіку, взаємодіяти з базами даних, обробляти запити клієнтів та виконувати аналітичні операції над даними. Завдяки зрозумілому синтаксису та активній спільноті, Python є одним із найкращих варіантів для створення веб-сервісів та API.
- **Flask** — популярний мікрофреймворк для розробки веб-додатків і RESTful API на Python. Його перевагами є легкість, простота освоєння, модульність та можливість швидкої розробки серверної частини з мінімальним обсягом коду. Flask дозволяє створювати REST-сервіси для обміну даними між клієнтською та серверною частинами застосунку, а також легко інтегрується з різними бібліотеками та базами даних.
- **psycopg2** — одна з найпопулярніших та найпотужніших клієнтських бібліотек для взаємодії Python із системою керування базами даних PostgreSQL. Вона забезпечує стабільне та швидке з’єднання, підтримує транзакції, підготовлені запити, параметризацію SQL-запитів, а також дозволяє працювати як зі звичайними, так і з розширеними типами даних PostgreSQL. Завдяки цій бібліотеці можна організувати ефективний обмін даними між сервером та базою даних.
- **PyCharm** — професійне інтегроване середовище розробки (IDE) для мови Python, яке надає розробникам зручні інструменти для написання, тестування, налагодження та супроводу коду. PyCharm підтримує роботу з різними фреймворками, має вбудований інтегратор із системами контролю версій, а також потужні засоби для роботи з базами даних та REST API. Завдяки широким можливостям

автоматизації та рефакторингу коду, ця IDE значно спрощує та прискорює процес розробки серверної частини застосунку.

Такий стек дозволяє забезпечити швидку обробку вхідних HTTP-запитів і безперебійну взаємодію з базою даних, у якій зберігаються всі виміряні параметри (температура, вологість, концентрація CO_2).

На етапі створення бази даних у СКБД PostgreSQL була створена нова база з назвою ghg_monitoring. У ній передбачено одну таблицю — measurements, яка містить поля:

- id — унікальний ідентифікатор запису;
- temperature — значення температури в градусах Цельсія;
- humidity — рівень відносної вологості у відсотках;
- co2_ppm — умовна або фактична концентрація CO_2 у ppm;
- timestamp — мітка часу, коли були отримані дані;
- wind_speed — значення швидкості вітру;
- wind_dir — напрямок вітру;
- lat — географічна широта;
- lon — географічна довгота;
- altitude — висота над рівнем моря.

Рисунок 3.3.1 – Структура таблиці measurements у базі даних PostgreSQL

Реалізація REST API на Flask

Для взаємодії з сенсорними пристроями на базі ESP32 був реалізований простий REST API. Сервер на Flask має маршрут /api/data, який приймає POST-запити з JSON-даними. Дані передаються у форматі:

```
{
 "device_id": "esp32-005",
 "timestamp": "2025-05-25T14:40:30Z",
```

```
  "temperature": 25.5,
  "humidity": 53.2,
  "co2_ppm": 440,
  "wind_speed": 6.5,
  "wind_dir": "SE",
  "lat": 49.8411,
  "lon": 24.0311,
  "altitude": 300
}
```

Ці дані витягуються із тіла запиту, після чого зберігаються у базі даних. У випадку успішного запису сервер повертає JSON-відповідь із підтвердженням отримання.

```python
@app.route( rule: "/api/data", methods=["POST"])
def receive_data():
    data = request.json
    if not data:
        return jsonify({"error": "No JSON data received"}), 400

    try:
        with conn.cursor() as cur:
            cur.execute( query: """
                INSERT INTO measurements
                (timestamp, temperature, humidity, co2_ppm, wind_speed, wind_dir, lat,
                lon, sd_status, battery_voltage, device_id)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            """, vars: (
                data.get("timestamp"),
                data.get("temperature"),
                data.get("humidity"),
                data.get("co2_ppm"),
                data.get("wind_speed"),
                data.get("wind_dir"),
                data.get("lat"),
                data.get("lon"),
                data.get("sd_status"),
                data.get("battery_voltage"),
                data.get("device_id")
            ))
```

Рисунок 3.3.2 – Приклад логіки обробки запиту на Flask

Верифікація роботи сервера

Після запуску сервер перевірявся за допомогою утиліт Postman і cURL, що дозволило переконатися в коректності прийому даних. Надалі цей сервер буде використовуватись для прийому реальних або емульованих даних із сенсорної системи.

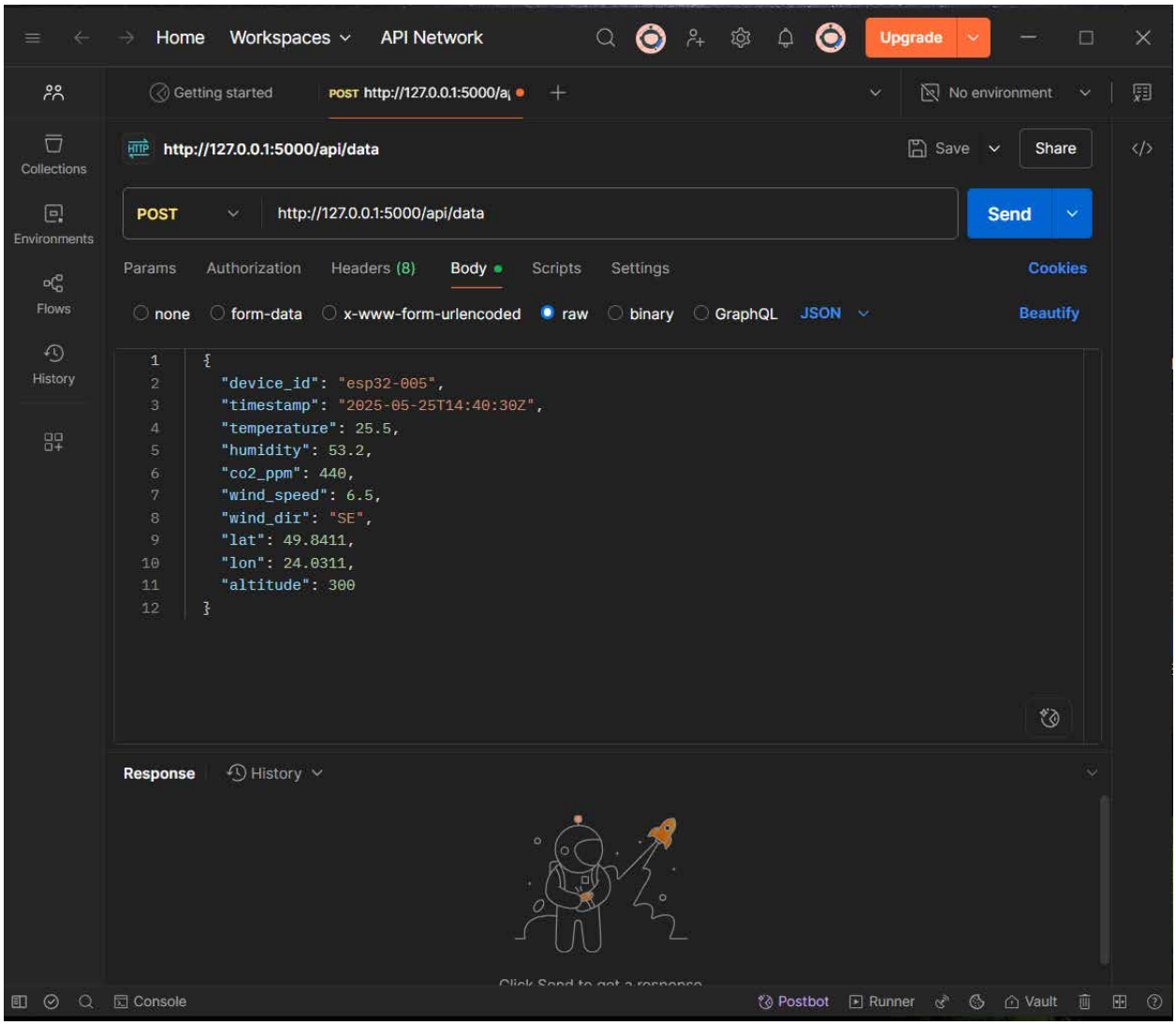


Рисунок 3.3.3 – Тестування API через Postman

Таким чином, серверна частина підсистеми накопичення даних успішно реалізована. Вона забезпечує:

- прийом даних у форматі JSON;
- перевірку вхідної інформації;

- збереження значень у базі даних PostgreSQL;
- простоту інтеграції з іншими модулями системи.

У наступних розділах буде продемонстровано механізми перевірки цілісності даних, а також реалізацію інтерфейсу для перегляду накопиченої інформації користувачем.

3.4 Реалізація контролю цілісності та надійності передачі

У системах моніторингу навколишнього середовища, зокрема парникових газів, надзвичайно важливо забезпечити цілісність та надійність переданих даних, оскільки навіть незначна втрата або спотворення значень температури, вологості чи концентрації CO_2 може призвести до хибних висновків або некоректної побудови аналітики. У цьому підрозділі розглядаються практичні заходи, що були реалізовані для забезпечення достовірності даних при їх збиранні, буферизації та передачі на сервер.

Забезпечення цілісності даних на рівні сенсорної системи

Цілісність починається ще з етапу зчитування даних із сенсорів. Наприклад, модуль MH-Z19C періодично може повертати некоректні значення (NaN) через перешкоди або електричні збої. Для таких випадків у коді ESP32 реалізовано перевірку дійсності зчитаних значень:

```
cpp
if (isnan(data.temperature) || isnan(data.humidity)) {
  Serial.println("Помилка зчитування!");
  return;
}
```

Подібну перевірку можна також реалізувати для MH-Z19C, щоб не передавати нульові або підозріло високі/низькі значення.

Буферизація та повторна передача

Оскільки передача даних у реальному часі через бездротовий зв'язок (наприклад, Wi-Fi або LoRa) не завжди стабільна, було реалізовано буферизацію даних у випадку відсутності з'єднання із сервером.

Система перевіряє наявність підключення до мережі перед передачею:

```
if (WiFi.status() != WL_CONNECTED) {
  // Зберегти дані у локальному буфері (пам’яті)
  // або на SD-карті (за потреби)
  return;
}
```

У разі відновлення зв'язку з сервером, буферизовані дані можуть бути відправлені повторно. Це дозволяє уникнути їх втрати через тимчасові збої в мережі.

Використання контрольних відповідей (acknowledgement)

На стороні сервера, реалізованого з використанням Flask, після отримання та успішного збереження даних у базу PostgreSQL сервер повертає підтвердження:

```
return jsonify({"status": "success"}), 201
```

На ESP32, у свою чергу, необхідно обробляти відповідь сервера, щоб переконатись у тому, що дані були прийняті. Наприклад, якщо відповідь не є 201 або у тілі відповіді немає "status: success", це означає, що слід повторити передачу пізніше:

```
if (httpResponseCode != 201) {
  // Повторити пізніше або записати в лог
}
```

Захист від повторної вставки

Щоб уникнути ситуацій, коли один і той самий пакет даних буде повторно передано кілька разів (наприклад, через непідтверджену відповідь від сервера), можна реалізувати ідентифікатори транзакцій або позначку часу. Таким чином сервер може перевіряти, чи вже був подібний запис, і ігнорувати дублікати.

Додаткові заходи:

1. Контроль діапазонів допустимих значень. На стороні сервера реалізується валідація: температура не може бути нижча за -50°C або вища за +100°C, вологість має бути у межах 0–100%, а концентрація CO_2 не повинна перевищувати допустимі екологічні межі (наприклад, 5000 ppm).
2. Логування помилок. При виникненні помилок — на клієнті або сервері — логуються відповідні повідомлення для подальшого аналізу.
3. Перевірка структури JSON. Сервер перевіряє, чи всі необхідні поля надіслані:

```
required_fields = ["temperature", "humidity", "co2"]
if not all(field in data for field in required_fields):
    return jsonify({"error": "Missing fields"}), 400
```

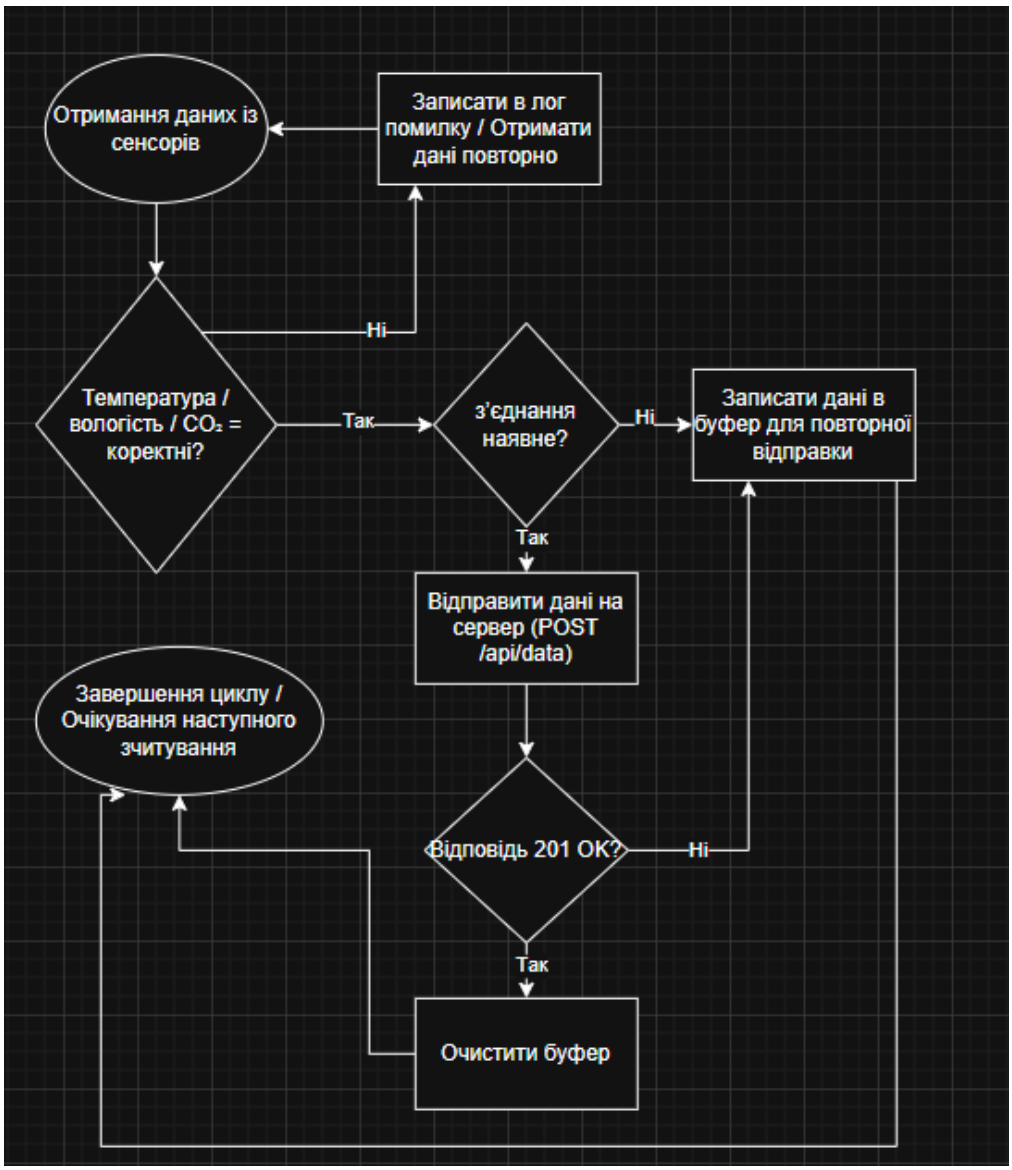


Рисунок 3.4.1 – Блок-схема контролю цілісності та логіки повторної передачі даних у сенсорній системі ESP32

Таким чином, для забезпечення надійної роботи системи моніторингу було реалізовано низку заходів контролю за цілісністю та достовірністю даних, що передаються від сенсорної частини до сервера. Серед них — перевірка зчитаних значень, буферизація, повторна передача, перевірка відповідей від сервера та валідація отриманих значень. Сукупність цих заходів дозволяє гарантувати, що збережені дані є точними, несфальсифікованими та повноцінними — що особливо важливо у задачах, пов'язаних із моніторингом кліматичних параметрів.

3.5 Реалізація інтерфейсу для перегляду накопичених даних

3.5.1 Варіанти реалізації інтерфейсу для перегляду даних

Інтерфейс для перегляду накопичених даних відіграє ключову роль у системі моніторингу парникових газів, оскільки забезпечує доступ користувача до результатів вимірювань у зручній та візуально зрозумілій формі. Його реалізація може здійснюватися різними способами залежно від технічних можливостей, поставлених вимог, обсягу даних і очікуваної частоти доступу до них.

Одним із найпоширеніших варіантів є веб-інтерфейс, який забезпечує доступ до даних через браузер з будь-якого пристрою в локальній мережі або через інтернет. Такий інтерфейс можна реалізувати за допомогою фреймворків, таких як Flask або Django (на Python), Node.js, або сучасних фронтенд-бібліотек і фреймворків — React, Vue.js, Angular. Веб-інтерфейс дозволяє не лише переглядати дані у вигляді таблиць, а й будувати графіки, діаграми, порівняльні звіти тощо. Головною перевагою цього підходу є універсальність та гнучкість в розробці, а також простота доступу з різних пристроїв без потреби встановлення додаткового програмного забезпечення.

Іншим підходом є **десктоп-додатки**, створені, наприклад, за допомогою Python (з використанням бібліотек Tkinter, PyQt або Kivy) або інших мов програмування. Такі рішення більше підходять для закритих середовищ, де система має працювати на одному або кількох комп’ютерах і немає потреби в доступі через браузер. Перевагою десктоп-додатків є більший контроль над інтерфейсом, краща інтеграція з локальними ресурсами та можливість офлайн-роботи. Проте вони вимагають встановлення ПЗ на кожен комп’ютер користувача, що ускладнює розгортання системи.

Також можливий варіант реалізації інтерфейсу на базі мобільного застосунку — це доцільно у випадках, коли користувач повинен мати доступ до даних у польових умовах або в режимі реального часу. Розробка мобільного інтерфейсу можлива на основі Android Studio (Java, Kotlin), Flutter (Dart), React Native (JavaScript) тощо. Проте така реалізація потребує більшого часу на розробку і тестування, а також реєстрації застосунку для зручного поширення серед користувачів.

У деяких випадках, особливо для внутрішнього використання, можна використовувати готові рішення — наприклад, засоби візуалізації, як-от Grafana, що дозволяє швидко створити інтерактивні дашборди для відображення даних з бази даних PostgreSQL. Такий підхід значно скорочує час впровадження інтерфейсу, але обмежує можливості кастомізації.

Загалом, вибір методу реалізації інтерфейсу має базуватися на потребах кінцевих користувачів, технічних можливостях і обсягах даних, що обробляються. Для цілей цієї дипломної роботи доцільним є створення простого веб-інтерфейсу на базі Python Flask, який дозволятиме переглядати показники отримані з сенсорної системи, у вигляді таблиці з можливістю сортування та фільтрації, а також у вигляді графіків змін показників з часом. Такий підхід забезпечить гнучкість, зручність та масштабованість системи.

3.5.2 Реалізація інтерфейсу для перегляду накопичених даних

З метою забезпечення зручного, наочного та функціонально розширеного перегляду накопичених екологічних даних із сенсорної системи, яка здійснює вимірювання температури, вологості, концентрації вуглекислого газу (CO_2) та інших параметрів, було розроблено веб-інтерфейс на основі мікрофреймворку Flask. Інтерфейс призначений для надання користувачу швидкого доступу до актуальної інформації з бази даних

PostgreSQL без необхідності використання командного рядка або сторонніх інструментів адміністрування. Завдяки використанню сучасних веб-технологій, інтерфейс поєднує простоту використання з широкими можливостями для аналізу даних.

Серверна частина реалізована на мові програмування Python із використанням Flask як основного фреймворку для обробки HTTP-запитів, формування HTML-сторінок та інтеграції з базою даних PostgreSQL. Для візуалізації даних застосовано шаблонізатор Jinja2, що дозволяє динамічно передавати дані з сервера до HTML-сторінки та формувати таблиці і графіки відповідно до параметрів вибірки.

Створений інтерфейс від початку орієнтований на розширену функціональність. Він підтримує взаємодію з користувачем, забезпечуючи можливість гнучкої фільтрації даних за часовими межами, ідентифікатором пристрою, діапазоном температури, вологості та концентрації CO_2. Це дозволяє аналітично працювати з інформацією, виділяти необхідні фрагменти, виявляти закономірності або порушення. Усі фільтри інтегровані у вигляді веб-форми, яка розміщується безпосередньо над таблицею даних, що значно покращує зручність використання інтерфейсу та дозволяє отримувати лише релевантну інформацію без перезавантаження або зміни структури запитів вручну.

Табличне подання інформації реалізоване з підтримкою інтерактивного сортування за будь-яким із доступних параметрів: часом вимірювання, температурою, вологістю, концентрацією CO_2, швидкістю та напрямком вітру, координатами розташування, станом SD-карти, напругою живлення та ідентифікатором пристрою. Це надає користувачам широкі аналітичні можливості вже безпосередньо у вікні браузера без потреби експорту даних у зовнішні системи.

Особливу увагу приділено візуалізації динаміки зміни параметрів. У рамках інтерфейсу реалізовано інтерактивний графік на основі бібліотеки

Plotly, який відображає зміну температури та концентрації CO_2 у часі. Графік має дві осі Y, що дозволяє паралельно аналізувати кілька показників, а його інтерактивні можливості (масштабування, переміщення, підказки) роблять роботу з великими обсягами даних інтуїтивно зрозумілою. Графік динамічно оновлюється відповідно до встановлених фільтрів, забезпечуючи синхронізацію із табличним відображенням та полегшуючи аналіз змін екологічної ситуації у часі.

Для зручності користувачів додано функцію експорту даних у форматі CSV. Завдяки цій можливості, користувач може у будь-який момент зберегти вибрані дані на комп'ютері для подальшої обробки у таких програмах як Microsoft Excel, Google Sheets або спеціалізованих системах аналітики. Це значно спрощує підготовку звітності, обчислення середніх значень, побудову зведених таблиць тощо.

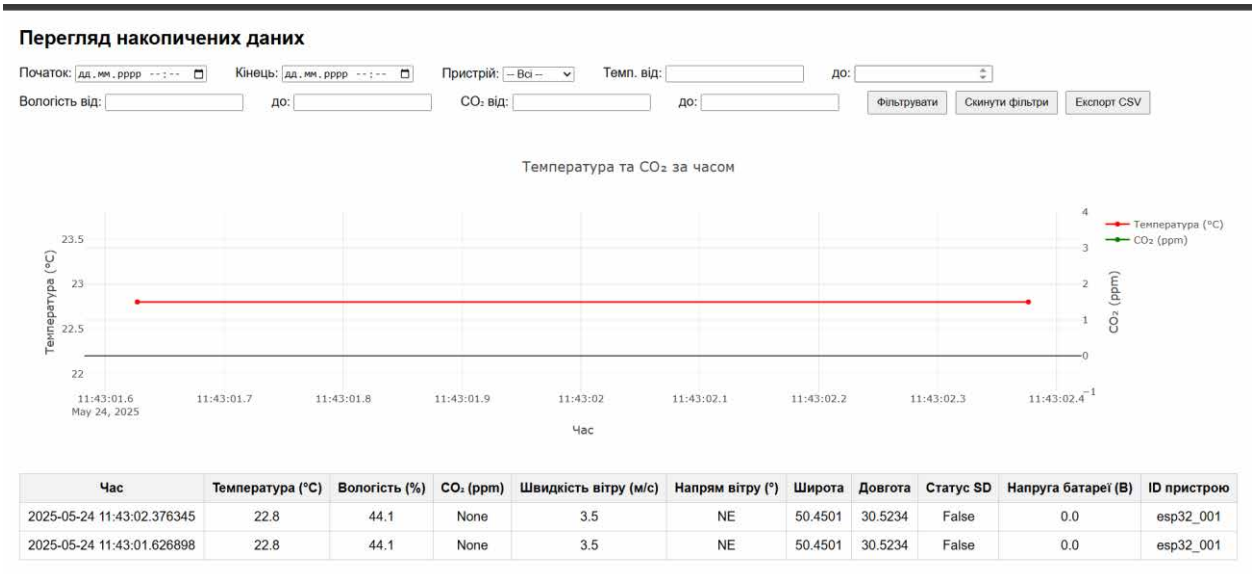
Крім того, веб-інтерфейс включає кнопку ручного опитування сенсорної системи, яка ініціює запит до окремого API-ендпоінту. Цей функціонал дозволяє оперативно отримати найсвіжіші показники з пристроїв у реальному часі, що є важливою опцією для адміністраторів системи чи дослідників під час польових досліджень або налагодження роботи пристроїв.

Окремої уваги заслуговує реалізація адаптивної верстки, що забезпечує коректне відображення інтерфейсу на різних пристроях — від настільних ПК до мобільних телефонів і планшетів. Таким чином, доступ до екологічних даних забезпечено у будь-який момент незалежно від розташування користувача, що суттєво підвищує гнучкість використання системи.

Основні функції реалізованого інтерфейсу:

- Виведення динамічної таблиці з даними сенсорної системи;
- Фільтрація записів за часовим діапазоном, значеннями параметрів, ID пристрою;

- Інтерактивне сортування даних за будь-яким полем;
- Побудова графіка зміни температури та CO_2 із двома шкалами Y;
- Експорт результатів фільтрації у форматі CSV;
- Запит поточних даних з сенсорів через кнопку ручного опитування;
- Адаптивна верстка для зручної роботи на мобільних пристроях.



Час	Температура (°C)	Вологість (%)	CO_2 (ppm)	Швидкість вітру (м/с)	Напрям вітру (°)	Широта	Довгота	Статус SD	Напруга батареї (В)	ID пристрою
2025-05-24 11:43:02.376345	22.8	44.1	None	3.5	NE	50.4501	30.5234	False	0.0	esp32_001
2025-05-24 11:43:01.626898	22.8	44.1	None	3.5	NE	50.4501	30.5234	False	0.0	esp32_001

Рисунок 3.5.2.1 – Вигляд інтерфейсу для перегляду накопичених даних

Таким чином, реалізований веб-інтерфейс є не лише засобом відображення даних, а повноцінним інструментом моніторингу, аналізу та оперативного реагування. Його відкритість до модифікації та використання сучасних технологій створює потенціал для подальшого розвитку: впровадження картографічного представлення розташування сенсорів, системи авторизації та розмежування прав доступу, оновлення даних у реальному часі за допомогою WebSocket, а також автоматичної генерації статистичних звітів за обрані періоди. Усі ці можливості будуть надалі спрямовані на підвищення ефективності екологічного моніторингу та покращення взаємодії з даними системи.

Отже після всіх введеннь код для сервера складається з двох частин server.py (див. лістинг А.1, ДОДАТОК А) та index.html (див. лістинг А.2, ДОДАТОК А).

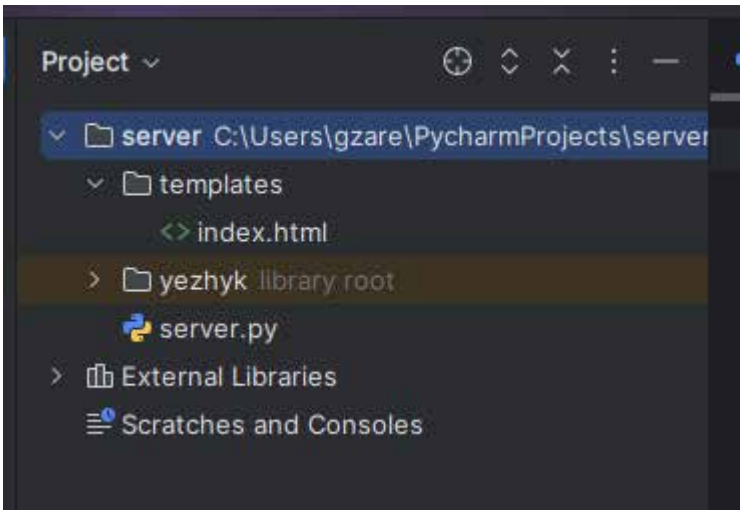


Рисунок 3.5.2.2 – структура проєкту

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Тестування системи та оцінка її ефективності

Для перевірки роботи серверної частини системи було розгорнуто REST API на основі мікрофреймворку **Flask**, який обробляє запити від сенсорної системи. Сервер приймає POST-запити з JSON-структурою, що містить показники датчиків.

У рамках тестування було проведено ручне наповнення бази даних за допомогою утиліти Postman. Було створено шаблон запиту, що містив поля усі дані які повинен надсилати ESP32.

Приклад JSON-запиту:

```
{
  "device_id": "esp32-005",
  "timestamp": "2025-05-25T14:40:30Z",
  "temperature": 25.5,
  "humidity": 53.2,
  "co2_ppm": 440,
  "wind_speed": 6.5,
  "wind_dir": "SE",
  "lat": 49.8411,
  "lon": 24.0311,
  "altitude": 300
}
```

Запити надсилалися на маршрут http://127.0.0.1:5000/api/data, після чого відбувалося збереження даних у базу PostgreSQL.

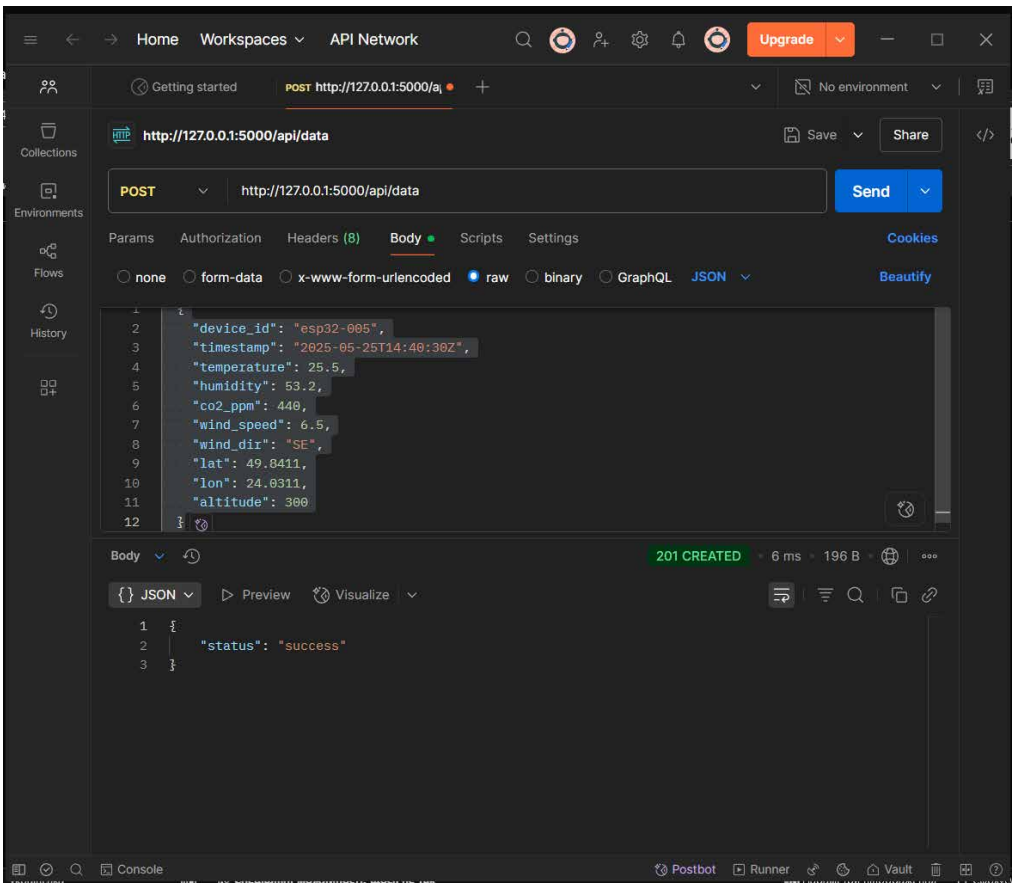


Рисунок 4.1.1 - Postman, який демонструє успішний POST-запит

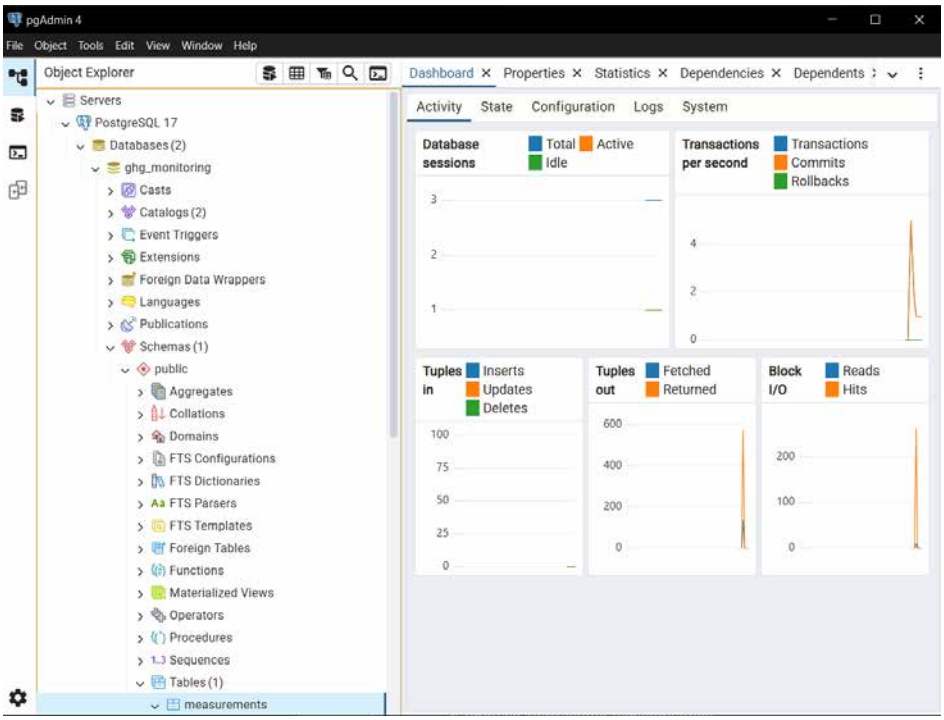


Рисунок 4.1.2 – Вигляд утиліти pgAdmin 4, в якій було створено БД

Після додавання кількох десятків записів у базу, було розпочато тестування веб-інтерфейсу системи, що дозволяє переглядати накопичені дані. Інтерфейс реалізовано як вебсторінку, яка підключається до тієї ж бази даних і відображає записи у вигляді таблиці або графіка.

У процесі тестування перевірялися:

- коректність виводу даних (збіг значень із тими, що передавались через Postman);
- сортування за часом;
- фільтрація за діапазоном температур або часу;
- відсутність дублювання записів;
- стійкість до помилок запиту.

Рисунок 4.1.3 – веб-інтерфейс

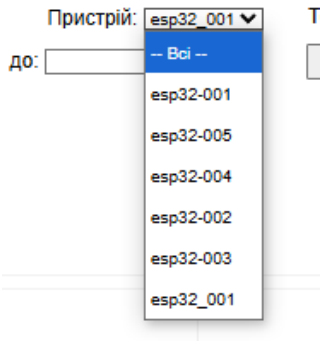
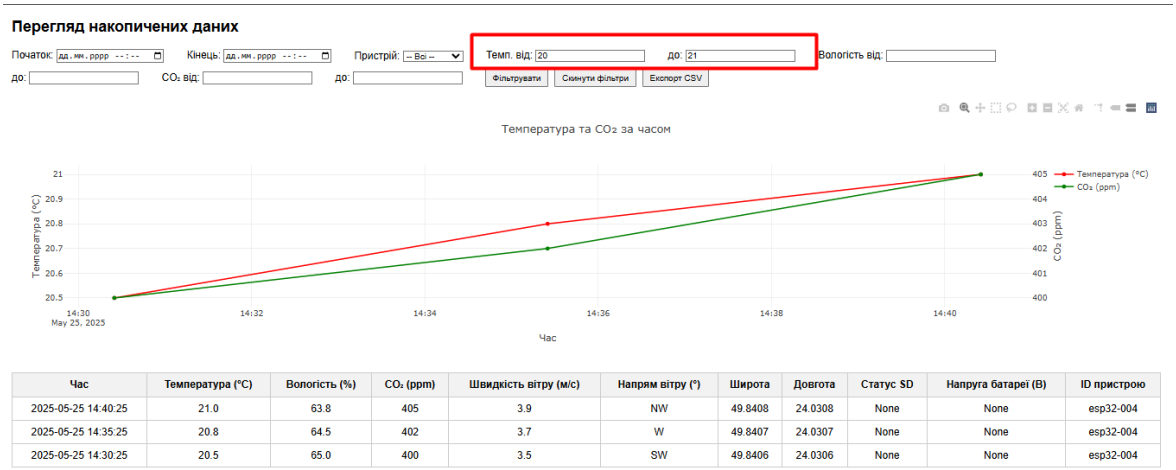


Рисунок 4.1.4 – вибір сортування за пристроєм

Час	Температура (°C)	Вологість (%)	CO_2 (ppm)	Швидкість вітру (м/с)	Напрям вітру (°)	Широта	Довгота	Статус SD	Напруга батареї (В)	ID пристрою
2025-05-25 14:40:30	25.5	53.2	440	6.5	SE	49.8411	24.0311	None	None	esp32-005
2025-05-25 14:35:30	25.2	52.5	438	6.2	E	49.841	24.031	None	None	esp32-005
2025-05-25 14:30:30	25.0	53.0	435	6.0	NE	49.8409	24.0309	None	None	esp32-005
2025-05-25 14:40:20	24.5	55.2	430	5.5	S	49.8405	24.0305	None	None	esp32-003
2025-05-25 14:35:20	24.2	54.5	428	5.2	SE	49.8404	24.0304	None	None	esp32-003
2025-05-25 14:30:20	24.0	55.0	425	5.0	E	49.8403	24.0303	None	None	esp32-003
2025-05-25 14:40:10	23.3	59.0	420	4.9	SE	49.8399	24.0299	None	None	esp32-001
2025-05-25 14:35:10	23.0	57.5	418	4.7	E	49.8398	24.0298	None	None	esp32-001
2025-05-24 11:43:01.626898	22.8	44.1	None	3.5	NE	50.4501	30.5234	False	0.0	esp32_001
2025-05-24 11:43:02.376345	22.8	44.1	None	3.5	NE	50.4501	30.5234	False	0.0	esp32_001
2025-05-25 14:30:10	22.5	58.0	415	4.5	NE	49.8397	24.0297	None	None	esp32-001
2025-05-25 14:40:15	22.4	59.8	414	4.2	W	49.8402	24.0302	None	None	esp32-002
2025-05-25 14:35:15	22.0	61.0	412	4.0	NW	49.8401	24.0301	None	None	esp32-002
2025-05-25 14:30:15	21.7	60.5	410	3.8	N	49.84	24.03	None	None	esp32-002
2025-05-25 14:40:25	21.0	63.8	405	3.9	NW	49.8408	24.0308	None	None	esp32-004
2025-05-25 14:35:25	20.8	64.5	402	3.7	W	49.8407	24.0307	None	None	esp32-004
2025-05-25 14:30:25	20.5	65.0	400	3.5	SW	49.8406	24.0306	None	None	esp32-004

Рисунок 4.1.5 – вигляд таблиці з усіма даними БД

Рисунок 4.1.6 – вигляд графіка температури та CO_2 за часом



Час	Температура (°C)	Вологість (%)	CO_2 (ppm)	Швидкість вітру (м/с)	Напрям вітру (°)	Широта	Довгота	Статус SD	Напруга батареї (В)	ID пристрою
2025-05-25 14:40:25	21.0	63.8	405	3.9	NW	49.8408	24.0308	None	None	esp32-004
2025-05-25 14:35:25	20.8	64.5	402	3.7	W	49.8407	24.0307	None	None	esp32-004
2025-05-25 14:30:25	20.5	65.0	400	3.5	SW	49.8406	24.0306	None	None	esp32-004

Рисунок 4.1.7 – приклад використання фільтра по температурі

Результати тестування показали, що система коректно обробляє вхідні запити, зберігає їх у базі даних та відображає у веб-інтерфейсі без помилок. Усі передані значення були присутні у таблиці measurements, а інтерфейс

працював стабільно та без збоїв навіть при одночасному доступі з кількох вкладок браузера.

4.2 Аналіз результатів та можливості масштабування

У результаті проведеного тестування всіх складових підсистеми накопичення даних було встановлено, що система демонструє високу стабільність, точність та ефективність у виконанні покладених на неї функцій. Реалізована архітектура дозволяє забезпечити безперервний автоматизований збір даних з сенсорної системи, їх локальне буферизування у разі втрати зв’язку, подальшу передачу до серверної частини та збереження у централізованій базі даних. При цьому всі компоненти — як апаратні, так і програмні — функціонують узгоджено, забезпечуючи мінімальні затримки, високу надійність обміну даними та збереження їх цілісності.

Наповнення бази даних за допомогою інструментів тестування (зокрема Postman) дало змогу перевірити роботу серверної частини без участі фізичних пристроїв, що дозволило ізолювати і протестувати окремі програмні модулі. Отримані дані успішно зберігалися в базі PostgreSQL, а їх подальше відображення через веб-інтерфейс підтвердило правильність логіки взаємодії між усіма компонентами системи. Перевірка виконувалася з різними обсягами даних, включно з імітацією великої кількості запитів у короткий проміжок часу, що дозволило виявити та усунути потенційні вузькі місця.

Ключовим висновком тестування є підтвердження того, що система витримує навантаження, стабільно функціонує в умовах перебоїв зв’язку, не втрачає інформацію при накопиченні даних у буфері, а також зберігає повну функціональність після відновлення підключення. Реалізована логіка повторної передачі даних та перевірки їх валідності забезпечує високий

рівень надійності — дані не дублюються, не зникають і не пошкоджуються при передаванні.

Особливу увагу варто звернути на потенціал масштабування розробленої системи. Завдяки використанню мікроконтролера ESP32 та протоколу зв'язку LoRa з'являється можливість розгортати мережу сенсорів на великих територіях, з відстанями до 10–15 км між вузлами. При цьому система не потребує централізованої інфраструктури, що знижує витрати на її впровадження. Усі вузли можуть працювати автономно, живлячись від акумуляторів з підзарядкою від сонячних панелей, що робить систему придатною для польових умов та екологічного моніторингу у віддалених регіонах.

Застосування PostgreSQL у поєднанні з розширенням TimescaleDB забезпечує високу продуктивність при роботі з великим обсягом часових рядів. База даних добре масштабується як вертикально (збільшення потужності одного сервера), так і горизонтально (розподілення навантаження між кількома вузлами), що дозволяє нарощувати кількість сенсорів у системі без суттєвих змін у програмній архітектурі. Окрім цього, структура бази даних уже на етапі проєктування враховує можливість подальшої інтеграції з візуалізаційними платформами на кшталт Grafana, Metabase або власними веб-інтерфейсами.

API-сервер, побудований на Flask, завдяки модульній побудові може бути розширений у майбутньому — наприклад, додаванням нових маршрутів для отримання зведених звітів, фільтрації даних за часовими або географічними параметрами, або підключенням нових клієнтів (наприклад, мобільних додатків або хмарних аналітичних сервісів). Це робить розробку відкритою для масштабування не лише в кількісному, а й у функціональному вимірі.

Загалом, система уже зараз є достатньо зрілою для подальшого розгортання у реальних умовах. Її відкритість, розширюваність та підтримка стандартних протоколів взаємодії роблять її придатною для використання не

лише як окремий проєкт, але й як складову частину більших екологічних платформ, регіональних систем моніторингу або наукових досліджень у сфері кліматології. Завдяки гнучкій архітектурі система може бути адаптована під інші типи сенсорів, наприклад для вимірювання концентрації метану, оксидів азоту, озону або твердих частинок (PM2.5), що відкриває перспективи її використання в інших галузях охорони довкілля.

Таким чином, результати тестування повністю підтверджують доцільність обраної архітектури, правильність технічних рішень та високий потенціал масштабування розробленої підсистеми як у технічному, так і в функціональному плані.

ВИСНОВКИ

Отже, у ході виконання дипломної роботи було реалізовано повний цикл проєктування та створення підсистеми накопичення даних, яка є ключовим елементом в екологічних системах моніторингу парникових газів. У результаті проведеного аналізу технічного завдання було обґрунтовано вибір сенсорів, апаратної платформи та методів передачі й збереження даних.

Було спроєктовано архітектуру сенсорної системи з використанням мікроконтролера ESP32 та потрібних сенсорів. Розроблено логіку збирання даних, буферизації та їх подальшої передачі до серверу. Створено REST API-сервер на базі Flask та базу даних PostgreSQL для збереження екологічних показників у вигляді часових рядів.

Система протестована з використанням інструментів ручного тестування (Postman), що підтвердило її працездатність, надійність та стійкість до перебоїв у зв’язку. Запропонована архітектура є масштабованою, енергоефективною та придатною до розгортання у реальних екологічних умовах.

Таким чином, у роботі досягнуто поставленої мети — створено підсистему накопичення даних, яка здатна забезпечити безперервний моніторинг, локальне збереження, перевірку цілісності та передачу екологічних даних для подальшого аналізу і прийняття рішень у сфері охорони довкілля.

ПЕРЕЛІК ПОСИЛАНЬ

1. Основи побудови систем моніторингу навколишнього середовища [Електронний ресурс] – Режим доступу: https://eco.com.ua/article/system-monitoring-basics
2. Використання IoT у моніторингу повітря [Електронний ресурс] – Режим доступу: https://www.iotforall.com/iot-air-quality-monitoring
3. LoRaWAN Technology Overview [Електронний ресурс] – Режим доступу: https://lora-alliance.org/about-lorawan
4. Інтернет речей (IoT): основи та можливості [Електронний ресурс] – Режим доступу: https://techukraine.org/2021/04/12/iot-in-ukraine/
5. ESP32 Technical Reference Manual [Електронний ресурс] – Режим доступу: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
6. Сенсори для екологічного моніторингу: типи, застосування [Електронний ресурс] – Режим доступу: https://uatom.org/ekologichnyj-monitoring
7. Flask Documentation [Електронний ресурс] – Режим доступу: https://flask.palletsprojects.com/en/2.2.x/
8. PostgreSQL Documentation [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/
9. TimescaleDB: Efficient Time-Series Data [Електронний ресурс] – Режим доступу: https://www.timescale.com/docs
10. Моніторинг парникових газів: український досвід [Електронний ресурс] – Режим доступу: https://mepr.gov.ua/news/37616.html
11. Симулятор IoT-схем Wokwi [Електронний ресурс] – Режим доступу: https://docs.wokwi.com/

12. DHT22 Sensor with ESP32 [Електронний ресурс] – Режим доступу: https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-monitor-arduino/
13. Greenhouse Gas Emissions Explained [Електронний ресурс] – Режим доступу: https://www.epa.gov/ghgemissions
14. Environmental Monitoring with Open Technologies [Електронний ресурс] – Режим доступу: https://opensource.com/article/19/4/environmental-monitoring

ДОДАТОК А

Лістинг А.1 - server.py

```
from flask import Flask, request, jsonify, render_template, send_file, redirect,
url_for
import psycopg2
import csv
import io

app = Flask(__name__)

conn = psycopg2.connect(
    dbname="ghg_monitoring",
    user="postgres",
    password="root123",
    host="localhost",
    port="5432"
)

def fetch_devices():
    with conn.cursor() as cur:
        cur.execute("SELECT DISTINCT device_id FROM measurements")
        return [row[0] for row in cur.fetchall()]

def build_filters(args):
    filters = []
    values = []

    mappings = {
        "start": ("timestamp >= %s", args.get("start")),
        "end": ("timestamp <= %s", args.get("end")),
        "device_id": ("device_id = %s", args.get("device_id")),
        "min_temp": ("temperature >= %s", args.get("min_temp")),
        "max_temp": ("temperature <= %s", args.get("max_temp")),
        "min_humidity": ("humidity >= %s", args.get("min_humidity")),
        "max_humidity": ("humidity <= %s", args.get("max_humidity")),
        "min_co2": ("co2_ppm >= %s", args.get("min_co2")),
        "max_co2": ("co2_ppm <= %s", args.get("max_co2"))
    }

    for key, (clause, value) in mappings.items():
```

```
        if value:
            filters.append(clause)
            values.append(value)

    return filters, values

@app.route("/", methods=["GET"])
def show_data():
    filters, values = build_filters(request.args)

    query = """
        SELECT timestamp, temperature, humidity, co2_ppm, wind_speed, wind_dir,
               lat, lon, sd_status, battery_voltage, device_id
        FROM measurements
        WHERE TRUE
    """

    if filters:
        query += " AND " + " AND ".join(filters)

    query += " ORDER BY timestamp DESC LIMIT 1000"

    with conn.cursor() as cur:
        cur.execute(query, values)
        rows = cur.fetchall()

    devices = fetch_devices()

    return render_template("index.html", data=rows, devices=devices,
filters=request.args)

@app.route("/api/export", methods=["GET"])
def export_csv():
    with conn.cursor() as cur:
        cur.execute("""
            SELECT timestamp, temperature, humidity, co2_ppm, wind_speed,
wind_dir,
                   lat, lon, sd_status, battery_voltage, device_id
            FROM measurements
            ORDER BY timestamp DESC
        """)
        rows = cur.fetchall()

    si = io.StringIO()
    cw = csv.writer(si)
```

```
    cw.writerow(["timestamp", "temperature", "humidity", "co2_ppm",
"wind_speed", "wind_dir",
                 "lat", "lon", "sd_status", "battery_voltage", "device_id"])
    cw.writerows(rows)
    output = io.BytesIO()
    output.write(si.getvalue().encode("utf-8"))
    output.seek(0)

    return send_file(output, mimetype="text/csv", as_attachment=True,
download_name="measurements.csv")

@app.route("/reset_filters", methods=["GET"])
def reset_filters():
    return redirect(url_for("show_data"))

if __name__ == "__main__":
    app.run(debug=True)
```

Лістинг А.2 - index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8" />
    <title>Накопичені дані сенсорів</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        table { border-collapse: collapse; width: 100%; margin-top: 20px; }
        th, td { border: 1px solid #ccc; padding: 8px; text-align: center; }
        th { cursor: pointer; background-color: #f2f2f2; }
        form, .controls { margin-bottom: 20px; }
        label { margin-right: 10px; }
        input, select { margin-right: 15px; }
        button { padding: 5px 10px; }
        .controls { display: flex; flex-wrap: wrap; gap: 10px; align-items: center; }
    </style>
</head>
<body>

<h2>Перегляд накопичених даних</h2>

<form method="get" action="/" class="controls">
```

```
  <label>Початок:
    <input type="datetime-local" name="start" value="{{ filters.start or '' }}">
  </label>
  <label>Кінець:
    <input type="datetime-local" name="end" value="{{ filters.end or '' }}">
  </label>
  <label>Пристрій:
    <select name="device_id">
      <option value="">-- Всі --</option>
      {% for d in devices %}
        <option value="{{ d }}" {% if filters.device_id == d %}selected{%
endif %}>{{ d }}</option>
      {% endfor %}
    </select>
  </label>
  <label>Темп. від:
    <input type="number" step="0.1" name="min_temp" value="{{
filters.min_temp or '' }}">
  </label>
  <label>до:
    <input type="number" step="0.1" name="max_temp" value="{{
filters.max_temp or '' }}">
  </label>
  <label>Вологість від:
    <input type="number" step="0.1" name="min_humidity" value="{{
filters.min_humidity or '' }}">
  </label>
  <label>до:
    <input type="number" step="0.1" name="max_humidity" value="{{
filters.max_humidity or '' }}">
  </label>
  <label>CO₂ від:
    <input type="number" name="min_co2" value="{{ filters.min_co2 or '' }}">
  </label>
  <label>до:
    <input type="number" name="max_co2" value="{{ filters.max_co2 or '' }}">
  </label>
  <button type="submit">Фільтрувати</button>
  <a href="{{ url_for('reset_filters') }}"><button type="button">Скинути
фільтри</button></a>
  <a href="{{ url_for('export_csv') }}"><button type="button">Експорт
CSV</button></a>

</form>
```

```
<div id="chart" style="height: 400px;"></div>

<table id="data-table">
  <thead>
    <tr>
      <th>Час</th>
      <th>Температура (°C)</th>
      <th>Вологість (%)</th>
      <th>CO₂ (ppm)</th>
      <th>Швидкість вітру (м/с)</th>
      <th>Напрям вітру (°)</th>
      <th>Широта</th>
      <th>Довгота</th>
      <th>Статус SD</th>
      <th>Напруга батареї (В)</th>
      <th>ID пристрою</th>
    </tr>
  </thead>
  <tbody>
    {% for row in data %}
    <tr>
      <td>{{ row[0] }}</td>
      <td>{{ row[1] }}</td>
      <td>{{ row[2] }}</td>
      <td>{{ row[3] }}</td>
      <td>{{ row[4] }}</td>
      <td>{{ row[5] }}</td>
      <td>{{ row[6] }}</td>
      <td>{{ row[7] }}</td>
      <td>{{ row[8] }}</td>
      <td>{{ row[9] }}</td>
      <td>{{ row[10] }}</td>
    </tr>
    {% endfor %}
  </tbody>
</table>

<script>
// Сортування таблиці
document.querySelectorAll('th').forEach((th, index) => {
  th.addEventListener('click', () => {
    const table = th.closest('table');
    const tbody = table.querySelector('tbody');
    const rows = Array.from(tbody.querySelectorAll('tr'));
    const asc = !th.classList.contains('asc');
```

```
        rows.sort((a, b) => {
            const aText = a.children[index].textContent.trim();
            const bText = b.children[index].textContent.trim();
            return asc ? aText.localeCompare(bText, undefined, { numeric: true }) :
bText.localeCompare(aText, undefined, { numeric: true });
        });
        rows.forEach(row => tbody.appendChild(row));
        table.querySelectorAll('th').forEach(th2 => th2.classList.remove('asc', 'desc'));
        th.classList.toggle('asc', asc);
        th.classList.toggle('desc', !asc);
    });
});

// Опитування сенсора
function pollSensor() {
    fetch('/api/poll_sensor', {
        method: 'POST'
    })
    .then(response => response.json())
    .then(data => {
        alert("Дані опитування отримано. Оновіть сторінку для перегляду.");
    })
    .catch(err => alert("Помилка опитування: " + err));
}

// Побудова графіку температури і CO₂
document.addEventListener("DOMContentLoaded", function () {
    const rows = document.querySelectorAll("#data-table tbody tr");
    const times = [];
    const temps = [];
    const co2s = [];

    rows.forEach(row => {
        times.push(row.children[0].textContent);
        temps.push(parseFloat(row.children[1].textContent));
        co2s.push(parseInt(row.children[3].textContent));
    });

    const tempTrace = {
        x: times,
        y: temps,
        name: 'Температура (°C)',
        mode: 'lines+markers',
        line: { color: 'red' }
    };
```

```
    const co2Trace = {
        x: times,
        y: co2s,
        name: 'CO₂ (ppm)',
        mode: 'lines+markers',
        line: { color: 'green' },
        yaxis: 'y2'
    };

    const layout = {
        title: 'Температура та CO₂ за часом',
        xaxis: { title: 'Час' },
        yaxis: { title: 'Температура (°C)' },
        yaxis2: {
            title: 'CO₂ (ppm)',
            overlaying: 'y',
            side: 'right'
        }
    };

    Plotly.newPlot('chart', [tempTrace, co2Trace], layout);
});
</script>

</body>
</html>
```