Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Мікроконтролерна система моніторингу компонентів довкілля. Серверна частина.

Назва теми

КвРКІ.200246.20.02.23 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-20-2 _____ М. О. Сойко
Підпис Ініціали, прізвище

Керівник _____ Д. М. Медзатий
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С. М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«19» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень Бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т. О. Говорущенко

" 10 " 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Сойко Максиму Олександровичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Мікроконтролерна система моніторингу компонентів довкілля. Серверна частина.

Керівник проекту (роботи) Медзатий Д.М., к.т.н., доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметної області серверної частини мікроконтролерної системи моніторингу компонентів довкілля

Вибір технологій для розробки серверної частини мікроконтролерної системи моніторингу компонентів довкілля

Опис програмної реалізації серверної частини мікроконтролерної системи моніторингу компонентів довкілля

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура ПЗ проєкту

Діаграми роботи системи

Програмне забезпечення проєкту

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прий
Нормоконтроль	Лисенко С.М., професор кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання «_10_» _____01_____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Приміт
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконан
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконан
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	24.02.2024	виконан
4	Робота над розділом 2 – вибір технологій для розробки серверної частини мікроконтролерної системи моніторингу компонентів довкілля	01.04.2024	виконан
5	Робота над розділом 3 – програмна реалізація серверної частини мікроконтролерної системи моніторингу компонентів довкілля	10.05.2024	виконан
6	Оформлення пояснювальної записки згідно вимог	29.05.2024	виконан
7	Попередній захист ВКР	30.05.2024	виконан
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ М. О. Сойко
Підпис Ініціали, прізвище

Керівник роботи _____ Д. М. Медзатий
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл.-листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ.200246.02.02.23 ПЗ	Пояснювальна записка	67		
			Графічні матеріали			
2		КвРКІ.200246.02.02.23 Е8	Архітектура ПЗ проєкту	1		
3		КвРКІ.200246.02.02.23 Е8	Діаграми роботи системи	1		
4		КвРКІ.200246.02.02.23 Е8	Програмне забезпечення	1		
			проєкту			

КвРКІ.200246.20.02.23 ВП

Зм	Арк	№ докум	Підпис	Дата
Розробив		Сойко		
Перевір.		Медзатий		
Н. контр.		Лисенко		
Затв.		Говорущенко		19.06

Відомість проекту

Літера	Аркуш	Аркушів
У	1	67

ХНУ, КІ2-20-2

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мікроконтролерна система моніторингу компонентів довкілля. Серверна частина».

Автор роботи: Сойко Максим Олександрович.

Керівник роботи: Медзатий Дмитро Миколайович.

Пояснювальна записка: 67 с., 28 рис., 6 табл., 3 дод., 51 джерело.

Графічна частина: 3 креслення.

Серверна частина, Мікроконтролерна система, Архітектура, Моніторинг.

Метою даної дипломної роботи є розробка та дослідження серверної частини мікроконтролерної системи моніторингу компонентів довкілля.

Об'єктом дослідження є система моніторингу компонентів довкілля, яка включає в себе апаратні та програмні засоби для збору, обробки та аналізу екологічних даних.

Предметом дослідження є серверна частина мікроконтролерної системи моніторингу компонентів довкілля, зокрема її архітектура, технології, програмне забезпечення та методи обробки і зберігання даних, а також механізми взаємодії з клієнтськими додатками.

Під час проведення даного дослідження був використаний метод систематичного огляду літератури для вивчення і аналізу предметної області даного дослідження з текстових джерел інформації. Для реалізації програмного забезпечення серверної частини використано методи розробки програмного забезпечення, зокрема методи інкрементної та ітеративної розробки, тестування коду, налагодження та оптимізації.

Підпис студента

04.06.2024

Дата

ЗМІСТ

ВСТУП

Термін «Інтернет речей» (IoT) відноситься до концепції різноманітних пристроїв із сенсорами, які мають здатність збирати, аналізувати та обмінюватися даними за допомогою Інтернету. Це можуть бути різні пристрої, від побутових пристроїв, таких як холодильники або розумні датчики освітлення, до промислового обладнання та транспортних засобів.

Роль IoT у нашому житті стає все більш суттєвою:

– забезпечення комфорту та зручності в побуті стає все більш доступним завдяки розумним пристроям. Вони дозволяють автоматизувати багато процесів, забезпечуючи оптимальні умови для життя. Наприклад, за допомогою смарт-термостатів можна контролювати освітлення та підтримувати комфортну температуру в приміщенні. Такі інновації значно спрощують наше повсякденне життя та роблять його більш зручним;

– оптимізація промислових процесів: використання IoT в промисловості дозволяє збирати та аналізувати дані з обладнання, що сприяє оптимізації виробничих процесів, зниженню витрат та уникненню аварійних ситуацій;

– забезпечення безпеки: IoT можна використовувати для створення систем безпеки вдома, на роботі або в громадських місцях. Це можуть бути системи відеоспостереження, датчики пожежі або системи виявлення витоку газу, які надсилають сповіщення на мобільний додаток власника;

– розумне управління містами та інфраструктурою: IoT може допомогти містам стати більш розвиненими та інтелектуальними, шляхом впровадження систем управління транспортом, водопостачанням, освітленням та іншими, що дозволить ефективніше використовувати ресурси та забезпечити більш комфортне життя мешканців;

– здоров'я та медицина: у медицині IoT використовується для створення портативних пристроїв для моніторингу здоров'я, віддаленого відстеження

показників пацієнтів, а також для підтримки індивідуалізованої медичної допомоги.

Роль Інтернету речей (IoT) в моніторингу довкілля є критично важливою і дуже перспективною. Її суть полягає у створенні мережі розумних сенсорів і пристроїв, які можуть збирати різноманітні дані про стан навколишнього середовища.

Для моніторингу якості повітря можна використовувати повітряні датчики, розташовані в різних місцях міста або регіону, які вимірюють рівень забруднення атмосфери оксидами азоту, сірки, вуглеводнів та іншими шкідливими речовинами. Також можна використовувати радіаційний фон. Особливо це може бути необхідно поблизу атомних станцій. Ці дані можуть надсилатися в реальному часі на центральний сервер для подальшого аналізу.

Щодо води, датчики можуть бути встановлені у водоймах, річках та озерах для вимірювання рівня забруднення хімічними речовинами або мікробіологічними показниками, такими як бактерії. Ці дані допомагають виявляти потенційні джерела забруднення та приймати заходи для його усунення.

В сучасному світі проблеми екології та забруднення довкілля стають все більш актуальними. Для вирішення цих проблем важливо мати ефективні системи моніторингу, які забезпечують збір, аналіз та візуалізацію даних про якість повітря, ґрунту, води та інших параметрів довкілля. Одним з ключових компонентів таких систем є серверна частина, яка відповідає за обробку, зберігання та аналіз зібраних даних в реальному часі.

Система моніторингу компонентів довкілля (СМКД) є комплексною інформаційно-технічною системою, спрямованою на збір, обробку, аналіз та візуалізацію даних про стан навколишнього середовища. Головною метою таких систем є надання оперативної та достовірної інформації про якість повітря, води, ґрунту, рослинний та тваринний світ та інші екологічно важливі параметри.

Елементами системи моніторингу можуть бути різноманітні датчики, які вимірюють рівень забруднення повітря оксидами азоту, сірки, вуглеводнів, рівень

радіаційного фону або рівень забруднення водойми хімічними речовинами. Також, до складу системи моніторингу компонентів довкілля можуть входити датчики для вимірювання рівня шуму, температури, вологості, тиску тощо.

Дані, зібрані датчиками, передаються до центральної системи збору та обробки, де вони аналізуються та візуалізуються за допомогою спеціалізованого програмного забезпечення. Це дозволяє оперативно реагувати на зміни в середовищі та вживати відповідних заходів для захисту навколишнього середовища та здоров'я населення.

Системи моніторингу компонентів довкілля є невід’ємною складовою екологічного управління та регулювання. Вони допомагають забезпечити ефективний контроль за станом довкілля, розробляти та впроваджувати екологічні стратегії та політику, спрямовані на збереження та відновлення екосистем.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ СЕРВЕРНОЇ ЧАСТИНИ МІКРОКОНТРОЛЕРНОЇ СИСТЕМИ МОНІТОРИНГУ КОМПОНЕНТІВ ДОВКІЛЛЯ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Предметна область для серверної частини мікроконтролерної системи моніторингу компонентів довкілля включає в себе широкий спектр аспектів, пов'язаних зі збором, зберіганням, обробкою та аналізом даних про довкілля. Що стосується збору та передачі даних, то цей аспект включає в себе розробку механізмів для збору даних від сенсорів та їх передачу на сервер для подальшої обробки.

Також невід’ємною складовою серверної частини є системи зберігання даних. Ця частина включає в себе розробку систем для зберігання великого обсягу даних про довкілля, зібраних з датчиків. Це може включати в себе використання реляційних та нереляційних баз даних, а також технології хмарного зберігання.

Не менш важливим є обробка та аналіз даних. Обробка та аналіз даний включають в себе розробку алгоритмів та програмного забезпечення для обробки та аналізу даних про довкілля. Це може бути виявлення аномалій, прогнозування змін, ідентифікація тенденцій та інші види аналізу.

Візуалізація та представлення даних включає в себе розробку інтерфейсів користувача для візуалізації та представлення даних про довкілля. Це можуть бути веб-інтерфейси, багатофункціональні веб-сайти, мобільні додатки або інші інтерактивні інтерфейси для взаємодії з користувачами.

Аналіз предметної області серверної частини системи моніторингу компонентів довкілля дозволяє виявити ряд проблем і завдань, які можуть виникнути при проєктуванні та розвитку таких систем.

Однією з ключових проблем є забезпечення сумісності та ефективної взаємодії серверної частини з мікроконтролерами та сенсорами, що використовуються для збору даних про довкілля. Це може вимагати розробки

спеціалізованих протоколів зв'язку та інтерфейсів для забезпечення надійної передачі даних. Для збору даних від датчиків якості повітря та їх передачі на сервер використовуються різноманітні технології IoT. Найбільш з поширених є: Wi-Fi (датчики можуть бути підключені до локальної мережі Wi-Fi, що дозволяє їм передавати дані безпосередньо на сервер через Wi-Fi маршрутизатор), Bluetooth (датчики можуть використовувати технологію Bluetooth для з'єднання зі смартфонами або іншими пристроями, які відправляють дані на сервер через мобільний зв'язок або Wi-Fi), LoRaWAN (це бездротова технологія, яка дозволяє передавати дані на великі відстані, особливо в сільській місцевості; датчики можуть використовувати LoRaWAN для передачі даних на вузли збору даних, які потім відправляють їх на сервер через Інтернет), NB-IoT (Narrowband IoT) (це стандарт мобільного зв'язку, спеціально призначений для IoT-пристроїв; він дозволяє датчикам відправляти дані на сервер через мобільний оператор, що забезпечує широкий охоплення та надійність). Ці технології надають широкі можливості для збору даних від датчиків та їх передачі на сервер для подальшої обробки та аналізу.

Збір та обробка великого обсягу даних, які надходять від сенсорів, може стати складною задачею для серверної частини. Необхідно розробити ефективні алгоритми зберігання, обробки та аналізу цих даних, щоб забезпечити їхню точність та достовірність. Для збору та обробки великого обсягу даних для серверної частини системи моніторингу компонентів довкілля можна використовувати різні технології та підходи, спеціально адаптовані для роботи з великими обсягами даних. Ось деякі з найбільш ефективних з них: Hadoop та Apache Spark - це фреймворки для обробки великих обсягів даних паралельно та розподілено. Hadoop забезпечує можливості зберігання та обробки даних на кластері серверів, а Apache Spark – надає потужні інструменти для аналізу та обробки даних у реальному часі. Також використовується розподілена система потокової обробки даних Apache Kafka, яка призначена для обміну великими потоками даних між різними компонентами системи моніторингу. Використання

таких бази даних, як MongoDB, Cassandra, або HBase, є ефективними для зберігання та обробки великих обсягів неструктурованих даних, що часто зустрічаються в системах моніторингу довкілля. Трапляється також використання алгоритмів машинного навчання та аналізу даних, яке дозволяє виявляти закономірності та патерни у великих масивах даних, що може допомогти у прогнозуванні та прийнятті рішень щодо якості довкілля. Може бути використане кешування даних та оптимізації запитів для підвищення продуктивності та ефективності обробки даних у реальному часі. Ці технології можуть бути використані окремо або в поєднанні одна з одною для забезпечення ефективної збору та обробки великих обсягів даних в системі моніторингу компонентів довкілля.

Оскільки дані про довкілля можуть бути чутливими та критично важливими для захисту навколишнього середовища, серверна частина повинна забезпечувати високий рівень безпеки даних. Це означає застосування шифрування, автентифікації та інших заходів захисту даних від несанкціонованого доступу. Забезпечення безпеки даних для серверної частини системи моніторингу компонентів довкілля є дуже важливим, оскільки ці дані можуть містити конфіденційну інформацію про якість повітря, води, ґрунту та інші параметри довкілля. Для забезпечення захищеності може використовуватись шифрування даних. Можуть використовуватись методи шифрування для захисту даних під час їх передачі через мережі та зберігання на сервері. Важливо застосовувати шифрування як для даних в спокійному стані, так і для даних під час передачі через мережу. Також може бути реалізований механізм аутентифікації користувачів та мікроконтролерів, які надсилають дані на сервер. Кожен користувач або пристрій має мати унікальний ідентифікатор та авторизаційні дані, щоб отримати доступ до системи. Досить популярними у таких системах є використання заходів захисту від різних видів атак, таких як атаки типу "перехоплення даних", "використання коду" та "зміна даних". Це може включати моніторинг та реагування на підозрілу активність, використання систем виявлення вторгнень (intrusion detection systems). Також можлива реалізація регулярного створення резервних копій даних для

запобігання втраті інформації в разі випадкового видалення, виходу з ладу обладнання або кібератак. Одним із заходів для забезпечення безпеки системи є встановлення суворих правил доступу до даних, що визначають, хто має право переглядати, редагувати або видаляти конкретні дані. Ці технології та методи допомагають забезпечити безпеку даних для серверної частини системи моніторингу компонентів довкілля, зберігаючи конфіденційність, цілісність та доступність інформації про якість довкілля.

Система моніторингу довкілля може розглядатися як критично важлива, тому необхідно забезпечити її масштабованість та надійність. Це означає розробку архітектури серверної частини, яка здатна працювати при великому обсязі даних та високих навантаженнях без втрати продуктивності. Використання хмарних платформ, таких як Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform тощо, дозволяє легко масштабувати серверні ресурси відповідно до потреб системи моніторингу довкілля. Хмарні сервіси також надають можливості автоматичного масштабування та високої доступності. Також можливе використання технологій контейнеризації, таких як Docker та Kubernetes, дозволяє розгортати, масштабувати та керувати додатками серверної частини ефективно та надійно. Для того, щоб серверна частина була здатна працювати при високих навантаженнях без втрати продуктивності можна розробити серверну частину як розподілену систему, що складається з кількох незалежних компонентів, які можуть працювати разом для обробки даних та виконання завдань. Це дозволяє розподіляти навантаження та забезпечувати резервне копіювання. Використання технологій балансування навантаження, таких як Nginx або HAProxy, дозволяє розподілити трафік між різними серверами, забезпечуючи оптимальне використання ресурсів. Для вчасного виявлення та запобігання проблем у серверній частині системи моніторингу компонентів довкілля використовують системи моніторингу та автоматизації, такі як Prometheus, Grafana, Ansible тощо. Саме вони дозволяють вчасно виявляти проблеми та автоматично втручатися для їх вирішення. Ці технології та методи дозволяють забезпечити масштабованість та

надійність серверної частини системи моніторингу компонентів довкілля, що є критично важливим для забезпечення стабільної та ефективної роботи системи в умовах зростаючого обсягу даних та навантаження.

Важливо забезпечити зручний доступ до даних про довкілля для користувачів через веб-інтерфейси або мобільні додатки. Це допоможе користувачам зрозуміти стан довкілля та приймати обгрунтовані рішення щодо його збереження та охорони. Для забезпечення зручного доступу до даних про довкілля для користувачів через веб-інтерфейси або мобільні додатки можна використовувати різноманітні технології та інструменти. Для створення інтерактивних та зручних веб-інтерфейсів для користувачів дуже часто використовують такі технології як: HTML, CSS, JavaScript. Використання фреймворків, таких як React, Angular або Vue.js, застосовують для розробки більш складних та функціональних веб-додатків. Розробка мобільних додатків для платформ Android та iOS здійснюється за допомогою мов програмування, таких як Java/Kotlin для Android і Swift/Objective-C для iOS. Також використовуються такі фреймворки, як React Native, Flutter або Xamarin, для розробки мобільних додатків, які працюють як на Android, так і на iOS. Доволі популярним також є використання WebSocket або Server-Sent Events для реалізації оновлень даних в режимі реального часу без необхідності постійного оновлення сторінки або додатка. Ці технології допомагають забезпечити зручний та ефективний доступ до даних про довкілля для користувачів через веб-інтерфейси або мобільні додатки, забезпечуючи при цьому зручність, безпеку та продуктивність.

Ці аспекти предметної області визначають основні вимоги та завдання для розробки серверної частини системи моніторингу компонентів довкілля. Розуміння цих аспектів допомагає ефективно розробляти та впроваджувати системи моніторингу довкілля з метою збереження природних ресурсів та забезпечення здоров'я та безпеки населення.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

1. Aclima Sensing Network – це система моніторингу довкілля, яка використовує сенсори для збору даних про якість повітря, рівень забруднення та інші параметри довкілля. Нижче наведені деякі переваги та недоліки цієї системи:

Переваги:

1) Aclima Sensing Network має високу точність даних, так як використовує високоякісні сенсори та технології для збору даних, що дозволяє отримувати точні та надійні вимірювання параметрів довкілля;

2) система має широкий спектр параметрів, тому може вимірювати різні параметри довкілля, такі як рівень забруднення повітря, температура, вологість, рівень шуму тощо;

3) серверна частина забезпечує можливість отримання даних у реальному часі, що дозволяє оперативно реагувати на зміни у середовищі;

4) система має потенціал для масштабування, що дозволяє обробляти великі обсяги даних та підтримувати зростаючу кількість сенсорів;

5) серверна частина цієї системи може забезпечити аналітичні звіти та графіки для візуалізації даних та аналізу трендів у довкіллі.

Недоліки:

1) побудова та підтримка інфраструктури для системи може вимагати значних матеріальних витрат;

2) система вимагає регулярної підтримки та оновлень для забезпечення надійності та актуальності даних;

3) в зоні поганого зв'язку можуть виникати проблеми з передачею даних з сенсорів до серверної частини;

4) збір та зберігання даних про довкілля можуть спричинити проблеми з приватністю даних та їх захистом від несанкціонованого доступу;

5) сенсори потребують енергопостачання, що може бути проблематичним у віддалених або важкодоступних місцях.

Серверна частина Aclima Sensing Network реалізована на основі сучасних технологій інтернету речей (IoT), хмарних обчислень та аналітики даних.

Для збору даних від датчиків якості повітря в цій системі використовуються пристрої IoT, що дозволяють передавати дані через бездротові мережі, такі як Wi-Fi, LoRaWAN або NB-IoT. Ці пристрої можуть бути підключені до центрального сервера через інтернет для зберігання та обробки даних.

Для обробки та аналізу великого обсягу даних система використовує хмарні обчислення. Система використовує хмарні сервіси для зберігання даних, проведення аналітичних операцій та прогнозування якості повітря.

Для зберігання та керування даними використовуються сучасні бази даних, такі як MongoDB, PostgreSQL або Amazon DynamoDB. Ці системи дозволяють ефективно зберігати та управляти великим обсягом структурованих та неструктурованих даних.

Aclima Sensing Network володіє великим потенціалом для вдосконалення моніторингу довкілля та сприяння збереженню навколишнього середовища. На рисунку 1.1 зображена система моніторингу компонентів довкілля Aclima Sensing Network.

Рисунок 1.1 – Система моніторингу компонентів довкілля Aclima Sensing Network

2. IBM Green Horizons - це великий, амбіційний проект, який використовує передові технології для прогнозування та управління якістю повітря. Датчики Internet of Things (IoT) збирають дані про якість повітря в різних місцях міста. Після цього дані передаються на сервери IBM для обробки за допомогою алгоритмів машинного навчання.

Можливість прогнозувати майбутню якість повітря, використовуючи зібрані дані та поточні тенденції, є відмінною особливістю IBM Green Horizons. Цей аспект робить IBM Green Horizons цінним інструментом для державних установ і організацій, які працюють над контролем забруднення повітря та сталим розвитком. Нижче наведені деякі переваги та недоліки цієї системи:

Переваги:

1) IBM Green Horizons використовує передові технології штучного інтелекту, машинного навчання та аналізу даних для передбачення екологічних змін, що дозволяє точно прогнозувати забруднення повітря та інші екологічні проблеми;

2) платформа забезпечує можливість отримання даних у реальному часі, що дозволяє оперативно реагувати на екологічні загрози та ризики;

3) IBM Green Horizons надає різноманітні інструменти та функції для моніторингу якості повітря, вимірювання рівня забруднення та прогнозування екологічних змін;

4) платформа має потенціал для масштабування та підтримки великого обсягу даних та користувачів;

5) IBM Green Horizons надає розширені можливості для аналізу даних та генерації звітів, що дозволяє приймати обґрунтовані рішення щодо екологічного стану.

Недоліки:

1) впровадження, підтримка та вчасне обслуговування платформи може вимагати значних фінансових витрат;

2) впровадження може бути складним завданням через потребу в інтеграції з існуючими системами та джерелами даних;

3) збір та зберігання даних може породжувати проблеми з приватністю та захистом персональної інформації;

4) платформа може бути залежною від точності та доступності даних, що надходять з різних джерел;

5) потенційні проблеми з сумісністю з існуючими системами можуть вплинути на ефективність використання платформи.

Серверна частина платформи IBM Green Horizons побудована з використанням ряду передових технологій, що дозволяють забезпечити надійний та ефективний збір, аналіз та управління даними екологічного моніторингу. Використані такі технології як: хмарні обчислення, бази даних, мікросервісна архітектура, датчики та пристрої IoT, аналітика та штучний інтелект.

Для збору даних про екологічні параметри, такі як якість повітря, рівень забруднення атмосфери та інші, IBM Green Horizons використовує датчики та пристрої IoT. Ці дані передаються на серверну частину для подальшої обробки та аналізу.

IBM Green Horizons використовує такі хмарні платформи, як IBM Cloud та Amazon Web Services (AWS), для забезпечення масштабованості, надійності та доступності інфраструктури. Це дозволяє легко масштабувати ресурси в залежності від потреб та обсягу даних.

Для зберігання та обробки великого обсягу даних IBM Green Horizons використовує різні типи баз даних, такі як реляційні (наприклад, IBM Db2 або PostgreSQL) або нереляційні (наприклад, Apache Cassandra або MongoDB), в залежності від потреб і обсягів даних.

Серверна частина побудована з використанням мікросервісної архітектури, де окремі компоненти виконують конкретні функції і взаємодіють між собою через API. Це дозволяє забезпечити модульність та гнучкість системи. Також, серверна частина використовує аналітичні інструменти та методи штучного інтелекту для

аналізу даних, виявлення патернів та прогнозування екологічних змін. Використання машинного навчання дозволяє покращити точність та надійність прогнозів. Також використовується шифрування даних, що дає змогу зберегти серверну частину в безпеці від кібератак.

Ці технології спільно створюють потужну та ефективну серверну частину IBM Green Horizons, яка забезпечує високу продуктивність та точність аналізу даних про екологічне середовище. Приклад роботи системи IBM Green Horizons зображений на рисунку 1.2.

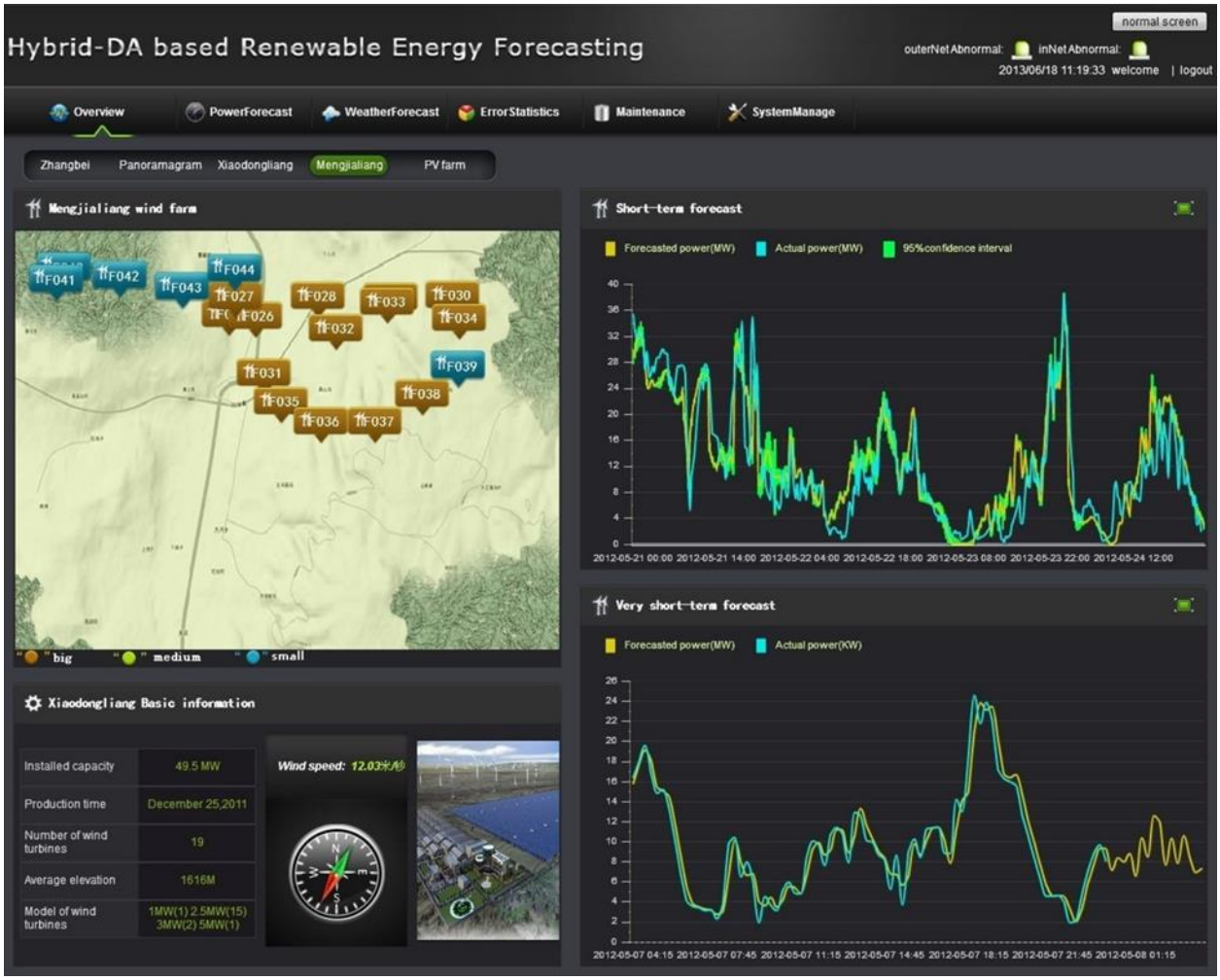


Рисунок 1.2 – Робота системи моніторингу компонентів довкілля IBM Green Horizons

3. Система Smart Greenhouse - це сучасне рішення для автоматизованого контролю та управління умовами вирощування рослин у теплицях. Основна ідея полягає в тому, щоб створити оптимальне середовище для росту рослин шляхом контролю різних параметрів, таких як температура, вологість, освітленість та інші. Серверна частина системи Smart Greenhouse грає важливу роль у забезпеченні ефективного функціонування та управління умовами вирощування рослин у теплицях. Розглянемо деякі переваги та недоліки цієї частини:

Переваги:

1) серверна частина дозволяє автоматизувати багато аспектів управління теплицею, включаючи контроль параметрів клімату, системи поливу та освітлення. Це сприяє оптимізації виробничих процесів та забезпечує кращі умови для росту рослин;

2) завдяки серверній частині оператори можуть отримувати дані про умови в теплиці в реальному часі, що дозволяє вчасно реагувати на зміни та вирішувати проблеми;

3) система дозволяє операторам віддалено керувати параметрами теплиці через веб-інтерфейс або мобільний додаток, що забезпечує зручність та доступність управління навіть на відстані;

4) серверна частина дозволяє оптимізувати використання енергії в теплиці, наприклад, включати системи опалення або освітлення лише за необхідності, що допомагає зменшити витрати на енергію.

Недоліки:

1) система потребує регулярного обслуговування та оновлення програмного забезпечення, що може бути пов'язане з додатковими витратами;

2) якщо серверна частина зазнає неполадок або перебоїв у роботі, це може призвести до втрати контролю над умовами в теплиці та негативно позначитися на рості рослин та врожайності;

3) як у будь-якої комп'ютерної системи, є можливість виникнення системних помилок або програмних збоїв, що також є недоліком, оскільки це може призвести до непередбачених проблем та неможливістю керування теплицею.

При розробці серверної частини системи Smart Greenhouse використовуються різноманітні технології для забезпечення надійності, ефективності та функціональності системи.

Багато серверних частин систем Smart Greenhouse базуються на хмарних платформах, таких як Amazon Web Services (AWS), Microsoft Azure або Google Cloud Platform. Ці платформи забезпечують гнучкість, масштабованість та високу доступність, що дозволяє ефективно обробляти дані та керувати системою з будь-якої точки світу.

Для зберігання та управління даними про умови вирощування рослин використовуються різні типи баз даних, такі як реляційні (наприклад, MySQL, PostgreSQL) або NoSQL (наприклад, MongoDB, Cassandra).

Можна зберігати та аналізувати величезну кількість даних про умови всередині та поза межами розумної теплиці на основі Інтернету речей. Використовуючи ці дані, оператори приймають обґрунтовані рішення щодо управління врожаєм і за потреби коригують процеси для отримання оптимального врожаю.

Деякі серверні частини Smart Greenhouse можуть бути побудовані на основі мікросервісної архітектури, де різні функціональні блоки системи представлені як окремі сервіси, що співпрацюють між собою.

Для аналізу даних про умови вирощування рослин та прийняття рішень використовуються різноманітні аналітичні інструменти та фреймворки, такі як Apache Spark, TensorFlow або SciPy.

Ці технології допомагають створити потужну та ефективну серверну частину системи Smart Greenhouse, яка забезпечує надійне та ефективне керування тепличними умовами для вирощування рослин. На рисунку 1.3 є схематичне зображення системи Smart Greenhouse.

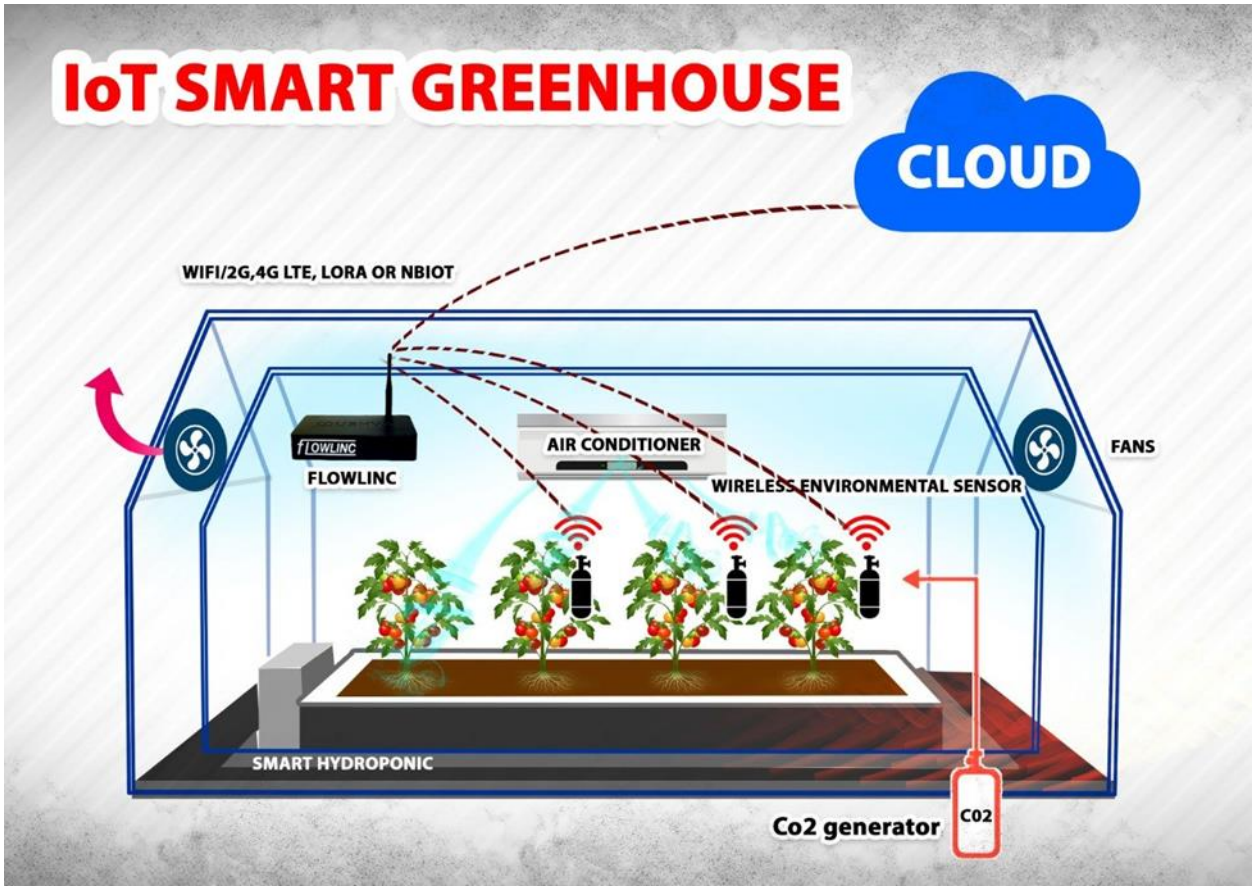


Рисунок 1.3 – Схематичне зображення системи Smart Greenhouse

У результаті порівняльного аналізу переваг та недоліків існуючих рішень щодо серверної частини систем моніторингу компонентів довкілля видно, що кожна з них має свої унікальні переваги та обмеження. Після здійснення аналізу, можна виділити такі переваги та недоліки у кожної з розглянутих систем моніторингу компонентів довкілля:

Переваги:

1) Aclima Sensing Network: висока точність та надійність збору даних, інтеграція з сучасними IoT технологіями;

2) IBM Green Horizons: використання штучного інтелекту для аналізу даних та прогнозування показників довкілля;

3) Smart Greenhouse: отримування даних про умови в теплиці в реальному часі, що дозволяє вчасно реагувати на зміни та вирішувати проблеми.

Недоліки:

1) Aclima Sensing Network: високі витрати на впровадження та обслуговування;

2) IBM Green Horizons: обмежена масштабованість системи та високі вимоги до обробки даних;

3) Smart Greenhouse: потреба у регулярному обслуговуванні, що може стати додатковими витратами.

1.3 Підходи до вирішення задачі за темою дослідження

При розробці серверної частини мікроконтролерної системи моніторингу компонентів довкілля можна використовувати різні підходи, які базуються на вимогах проекту, доступних технологіях та інших факторах. Існує декілька можливих підходів:

1) можна використати хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure або Google Cloud Platform для зберігання та обробки даних. Головною перевагою хмарних платформ є масштабованість та гнучкість, яка дозволить легко розширювати систему залежно від потреб;

2) також є можливим варіант розгортання серверної частини на власних фізичних або віртуальних серверах. Завдяки цьому ми отримаємо більший контроль над інфраструктурою та даними;

3) можуть бути використані мікроконтролери або малі комп'ютери (наприклад, Raspberry Pi) для обробки та аналізу даних на місці збору. Це надасть можливість реагувати на події в реальному часі без необхідності передачі даних на центральний сервер;

4) можливі гібридні рішення, тобто, комбінація хмарних та локальних серверних рішень для оптимального використання ресурсів та забезпечення високої доступності та надійності;

5) технології блокчейн можуть бути застосовані для забезпечення безпеки та недоторканності даних, особливо в сферах, де важлива відстежуваність та недоступність даних для маніпуляцій;

6) використання аналітичних інструментів, методів машинного навчання, а також штучного інтелекту може бути корисним для аналізу та прогнозування даних про довкілля, а також для реагування на виявлені тенденції та аномалії.

1.4 Постановка задачі

Метою даної дипломної роботи є розробка та дослідження серверної частини мікроконтролерної системи моніторингу компонентів довкілля. Основні завдання включають наступне:

– ретельний аналіз потреб користувачів та вимог до системи моніторингу довкілля;

– вивчення та вибір оптимальних технологій для реалізації серверної частини;

– розробка архітектури серверної частини системи моніторингу компонентів довкілля, зокрема збір, зберігання, аналіз та візуалізацію даних;

– реалізація та тестування серверної частини з використанням вибраних технологій;

– оцінка ефективності та продуктивності розробленої серверної частини на основі реальних даних.

В результаті виконання дипломної роботи очікується отримання наступних результатів:

– розроблена та реалізована серверна частина мікроконтролерної системи моніторингу компонентів довкілля;

– описана архітектура розробленої серверної частини та вибрані технології;

– проведений аналіз продуктивності та ефективності роботи розробленої системи на основі отриманих експериментальних даних;

– представлені рекомендації щодо вдосконалення та подальшого розвитку системи моніторингу компонентів довкілля.

Для досягнення поставленої мети та вирішення завдань буде використано наступні методи дослідження:

– аналіз літературних джерел та вивчення існуючих підходів до розробки систем моніторингу компонентів довкілля;

– аналіз переваг та недоліків існуючих аналогів систем моніторингу компонентів довкілля;

– експериментальне порівняння різних технологій та платформ для реалізації серверної частини;

– програмування та реалізація алгоритмів обробки даних та аналізу в обраному середовищі програмування;

– тестування та оцінка продуктивності розробленої системи з використанням реальних даних.

2 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ МІКРОКОНТРОЛЕРНОЇ СИСТЕМИ МОНІТОРИНГУ КОМПОНЕНТІВ ДОВКІЛЛЯ

2.1 Значення вибору правильних засобів розробки для забезпечення ефективності, надійності та масштабованості системи

Вибір правильного інструменту розробки для серверної частини системи моніторингу навколишнього середовища має вирішальне значення для забезпечення її ефективності, надійності та масштабованості.

Система повинна бути швидкою та якомога продуктивнішою: інструменти розробки повинні мати можливість оптимально виконувати операції для обробки та зберігання великих обсягів даних.

Необхідно використовувати стабільний і надійний інструмент розробки, який допоможе уникнути виникнення серйозних помилок, які можуть привести до збоїв у роботі системи.

Також, для покращення серверної частини системи можна використовувати масштабованість. Хорошим рішенням є забезпечити масштабування системи під збільшення обсягу або корисного навантаження даних за допомогою хмарних рішень або контейнеризації.

Не менш важливим є те, щоб вибрані засоби розробки програмного забезпечення мали високоякісну документацію та активну спільноту, здатну надавати підтримку та вирішувати проблеми.

Не потрібно забувати й про зручність у розробці та розширенні системи: бібліотеки та фреймворки повинні сприяти швидкому розгортанню нових функцій та простій інтеграції з іншими системами.

Необхідними у використанні також є інструменти для моніторингу та аналізу безпеки, які допоможуть вчасно виявляти та усувати потенційні загрози.

Вибираючи правильні інструменти розробки, можна створити стабільну, ефективну та масштабовану систему моніторингу навколишнього середовища для надійної роботи та задоволення потреб користувачів.

2.2 Функціональні вимоги мікроконтролерної системи моніторингу компонентів довкілля

Функціональні вимоги – це вимоги до програмного забезпечення, які визначають внутрішню роботу, функціонування системи. Обчислення даних, обробка даних та інші спеціальні функції, які повинна виконувати система. На відміну від нефункціональних вимог, які визначають, якою має бути система, функціональні вимоги визначають, що повинна робити система.

Для забезпечення коректної роботи мікроконтролерної системи моніторингу компонентів довкілля, система повинна мати можливість отримувати дані з мікроконтролера у різних точках мережі.

Для отримання даних необхідно реалізувати підтримку різних протоколів зв’язку (наприклад, HTTP, MQTT).

Після отримання даних потрібно здійснювати перевірку на коректність та достовірність отриманих значень. Наприклад, значення вологості повітря не може бути менше, ніж 0% та не може бути більше за 100%, значення концентрації пилу не може дорівнювати нулю або бути від’ємним. Крім того, необхідно перевіряти чи отримані дані відповідають встановленим стандартам або форматам. Наприклад, перевірка формати дати або часу, відповідність до одиниць вимірювання, тощо. Також бажано переконатися, що дані, які надсилаються з мікроконтролера, відповідають формату та структурі. Наприклад, необхідно перевірити правильність об'єкта JSON та формат запиту HTTP.

Ще однією з функціональних вимог є збереження даних у базу даних або у файл, а також можливість збереження історичних даних для подальшого аналізу та використання.

Необхідним є можливість виконання запитів для отримання даних за різними параметрами (наприклад, такими як, дата, час). Для зручності користувача потрібно забезпечити підтримку параметрів запитів у зрозумілому форматі, наприклад, через параметри URL або тіло запиту JSON.

Корисним буде фільтрація результатів запиту. Можливість використання фільтрів для певних критеріїв таких як конкретна дата або діапазон дат.

Також, важливим аспектом є адміністрування та керування системою. Необхідно забезпечити можливість спостереження та керування ресурсами серверної частини.

2.3 Нефункціональні вимоги мікроконтролерної системи моніторингу компонентів довкілля

Нефункціональні вимоги – це вимоги до програмного забезпечення, які задають критерії для оцінки якості його роботи. На відміну від функціональних вимог, які визначають що система повинна робити, нефункціональні вимоги визначають якою система повинна бути.

Система повинна бути продуктивною та забезпечувати достатню швидкодію. Час відповіді серверної частини на запити користувачів повинен бути якомога меншим, щоб забезпечити хорошу швидкодію навіть при великому навантажені.

Для безпекової організації системи необхідно використовувати механізми аутентифікації та авторизації для контролю доступу користувачів до функціоналу системи.

Ще однією важливою складовою є масштабованість. Масштабованість – це здатність системи обробляти більший обсяг роботи або бути легко розширеною.

Система повинна підтримувати різні протоколи зв’язку для взаємодії з мікроконтролерами, які використовують різні способи передачі даних.

Необхідно забезпечити доступність: користувачі повинні мати доступ до системи у будь-який час та у будь-яких умовах. Важливим є забезпечення зручності

у використанні. Інтуїтивно зрозумілий інтерфейс сприяє зручному використанню системи.

2.4 Опис стеку технологій

2.4.1 Node.js – середовище виконання JavaScript

Node.js – це середовище виконання JavaScript, побудоване на рушії V8 Chrome, яке забезпечує виконання коду на сервері. Воно дозволяє виконувати JavaScript на стороні сервера і використовується для розробки серверних додатків та мережевих додатків. Нижче наведена таблиця у якій описано переваги та недоліки використання технології Node.js (табл. 2.1).

Таблиця 2.1 – Переваги та недоліки використання Node.js

Переваги використання	Недоліки використання
Node.js працює на основі високопродуктивного JavaScript-двигуна V8, який забезпечує швидке виконання коду.	Оскільки Node.js працює в середовищі, де тільки один потік, це може бути проблемою для інтенсивних обчислювальних завдань. Але можна використовувати робочі потоки (worker threads), хоча це не так просто, як у мовах, які підтримують багатопоточність.
Асинхронна модель безперешкодного вводу-виводу дозволяє Node.js обробляти велику кількість одночасних запитів, не блокуючи виконання.	Коли використовується велика кількість зворотних викликів, код може стати важким для читання та розуміння.

Кінець таблиці 2.1 – Переваги та недоліки використання Node.js

Переваги використання	Недоліки використання
Можливість використання однієї і тієї ж мови програмування (JavaScript) на стороні клієнта та сервера значно спрощує процес розробки.	Деякі модулі та бібліотеки в екосистемі Node.js можуть не працювати належним чином.
Активна спільнота розробників безперервно вдосконалює Node.js та створює нові інструменти	Динамічна типізація JavaScript дозволяє виявити помилки тільки під час виконання, а не під час компіляції. Цю проблему можна частково вирішити, використовуючи TypeScript.
Завдяки асинхронній природі Node.js можна легко масштабувати додатки шляхом обробки подій та використання зворотних викликів.	Як відомо, Node.js швидко розвивається, що може призвести до змін API. Оскільки нові модифікації можуть бути несумісними зі старими, це може ускладнити підтримку довгострокових проєктів.

Node.js є потужним інструментом для розробки серверної частини, особливо хорошим для побудови систем у реальному часі та обробки великої кількості запитів одночасно. Але потрібно враховувати його обмеження, особливо щодо обчислювальних інтенсивних завдань і потенційних проблем із асинхронним програмуванням.

2.4.2 NPM – менеджер пакетів для JavaScript

NPM (Node Package Manager) – менеджер пакетів JavaScript, який входить у склад Node.js. Розробники можуть легко встановлювати, оновлювати та видаляти інструменти та бібліотеки, не витрачаючи часу на самостійний пошук та скачування файлів.

Менеджер пакетів NPM працює за принципом репозиторію, де розробники можуть встановлювати пакети. Кожен пакет має свій унікальний ідентифікатор, який називається ім’ям пакета.

Менеджер може отримати доступ до репозиторію за допомогою протоколу HTTP. Розробник використовує команду `npm install` для встановлення пакета. Ця команда надсилає запит до сховища, яке повертає його в архіві. NPM розпаковує архів у проєкт і включає пакет у список встановлених пакетів.

Пакет – це одиниця програмного забезпечення, яку можна встановити та використовувати в проєкті. У більшості випадків вона складається з коду, інструкцій і метаданих, які містять інформацію про сам пакет, таку як ім’я, версію, опис і список залежностей. Є можливість керувати пакетами в проєкті за допомогою файлу package.json. У цьому файлі наведено перелік встановлених пакетів і їхніх версій. Можна використовувати команди npm для встановлення, оновлення та видалення пакетів, коли файл package.json готовий. Це дозволяє підтримувати проєкт у поточному стані та впроваджувати нові функції, не турбуючись про труднощі інтеграції. Нижче наведена таблиця у якій описано переваги та недоліки використання менеджеру пакетів NPM (табл. 2.2).

Таблиця 2.2 – Переваги та недоліки використання NPM

Переваги використання	Недоліки використання
NPM пропонує більше мільйона пакетів для задоволення різних потреб розробників.	Через велику кількість пакетів і залежностей NPM, іноді, має проблеми з безпекою.

Кінець таблиці 2.2 – Переваги та недоліки використання NPM

Переваги використання	Недоліки використання
Встановлення пакетів та робота з ними здійснюється за допомогою простих команд, що робить процес швидким і зручним.	У зв'язку з великою кількістю пакетів деякі з них можуть бути застарілими. У підсумку можуть виникнути проблеми з сумісністю та безпекою проєкту.
NPM особливо корисний для великих проєктів з багатьма модулями, оскільки він полегшує керування залежностями.	Директорія `node_modules` швидко розширюються, що створює проблеми з дисковим простором і продуктивністю файлової системи.
Екосистема NPM постійно розширюється, доповнюється та підтримується великою спільнотою розробників, що дозволяє швидко вирішити проблеми, які виникли та забезпечує регулярні оновлення.	Процес налаштування та оновлення проєкту може бути ускладнений, якщо пакети конфліктують. Інструменти, такі як `npm dedupe` і `npm shrinkwrap`, можуть бути корисними, але вони не завжди можуть вирішити всі проблеми.
NPM спрощує процес розробки, створюючи скрипти, які автоматизують функції, такі як запуск тестів, збірка проекту та розгортання.	Хоча з випуском NPM 5 і вище швидкість встановлення пакетів значно покращилася, NPM раніше мав проблеми зі швидкістю.

NPM – простий у використанні та потужний інструмент для керування пакетами у JavaScript-проєктах. Незважаючи на це, потрібно звернути увагу на проблеми безпеки, сумісності та продуктивності, використовуючи додаткові інструменти та практики, щоб гарантувати стабільність і безпеку своїх проєктів.

2.4.3 Docker – платформа контейнеризації

Docker – це платформа контейнеризації, яка дозволяє створювати, запускати та масштабувати додатки в ізольованих середовищах, які називаються контейнерами. Контейнери – це легкий спосіб пакування та запуску застосунків, які можуть працювати на будь-якій машині з встановленою платформою.

Docker має багато переваг перед іншими методами розгортання застосунків, такими як віртуальні машини. Хоча обидві технології розв'язують проблему розгортання (deployment) застосунків у різних середовищах, вони використовують для цього різні підходи. Розглянемо основні відмінності цих технологій:

1) кожна віртуальна машина (VM) містить гостьову операційну систему. Вона може бути будь-якою, і вона може відрізнятися від основної операційної системи, встановленої на комп'ютері користувача. Це робить VM досить складним. На відміну від цього, контейнери Docker розташовуються на одному сервері з операційною системою хоста та поділяються між собою. Таким чином, контейнери Docker значно легші, ніж VM;

2) незважаючи на те, що Docker і віртуальна машина (VM) створюють окреме середовище для запуску застосунків, їхня основна відмінність полягає в тому, як ці технології ізолюють простір. Віртуальна машина (VM) має власну гостьову операційну систему, тому вона віртуалізує як ядро операційної системи, так і прикладний рівень програмного забезпечення. У свою чергу Docker віртуалізує лише прикладний рівень і працює поверх основної операційної системи хоста. Нижче на рисунку порівнюється принцип віртуалізації VM та Docker (рис. 2.1);

3) Docker і віртуальна машина (VM) використовують ресурси системи, розподілені між інстансами. Протягом усього часу роботи віртуальних машин (VM) вони завчасно запитують певну кількість ресурсів і блокують їх від інших програм. На відміну від віртуальних машин (VM), контейнери Docker просто запитують

апаратні ресурси в одному ядрі операційної системи, а не блокують велику кількість;

4) віртуальні машини (VM) не використовують основну операційну систему, а використовують окремі ядра операційної системи. Через спільне ядро хоста контейнери знаходяться під більшим ризиком безпеки. Таким чином, віртуальні машини (VM) є більш безпечними, ніж Docker. Рекомендується не надавати root-доступ (права суперкористувача) до застосунків або використовувати адміністративні права для запуску контейнерів Docker, якщо це необхідно для підвищення безпеки процесу;

5) контейнери Docker легші та потребують менше ресурсів, ніж віртуальні машини (VM). Через це Docker-контейнер запускається в рази швидше. Docker-контейнери не потребують встановлення операційної системи, тому їх легше масштабувати та дублювати. Запуск віртуальної машини (VM), який займає більше часу, можна порівняти з запуском окремої автономної системи. Віртуальні машини (VM) також потребують постійного розподілу ресурсів між контейнерами, якщо навантаження чи трафік змінюються.

Нижче на рисунку порівнюється принцип роботи Docker-контейнера та віртуальної машини (рис. 2.2).

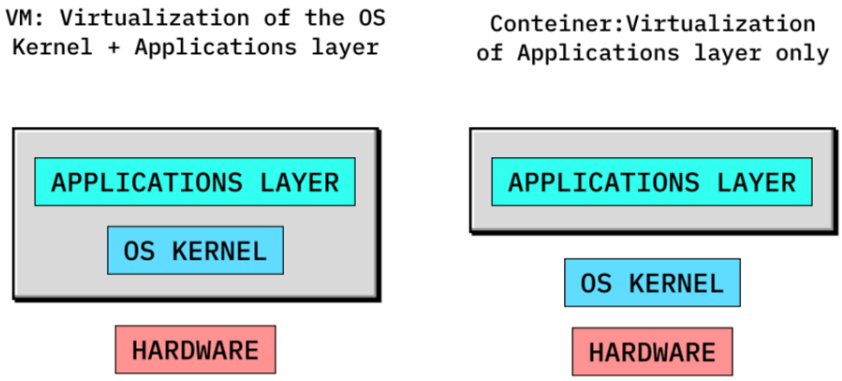


Рисунок 2.1 – Порівняння принципу віртуалізації Docker-контейнера та віртуальної машини (VM)

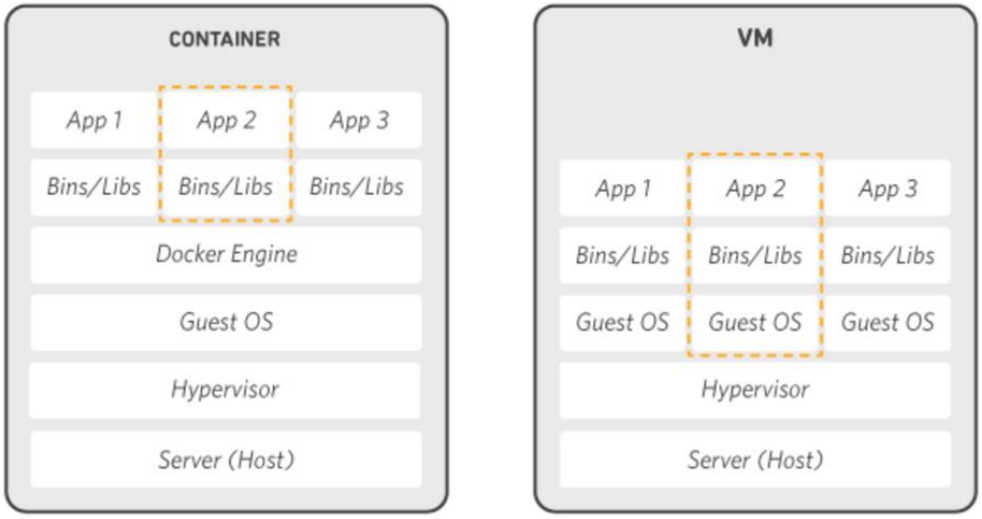


Рисунок 2.2 – Порівняння принципу роботи Docker-контейнера та віртуальної машини (VM)

Docker створює уявне середовище для кожного контейнера. У цьому середовищі містяться ядро операційної системи, бібліотеки, файли та налаштування, необхідні для роботи застосунку. Контейнери не можуть взаємодіяти безпосередньо, оскільки вони розташовані окремо один від одного. Це підвищує стабільність і запобігає конфліктам. Крім того, платформа забезпечує просте керування контейнерами. Запуск, зупинка, перезапуск і видалення контейнерів доступні за допомогою Docker CLI. Для створення та керування групами контейнерів також можна використовувати Docker Compose. Нижче наведена таблиця у якій описано переваги та недоліки використання платформи Docker (табл. 2.3).

Таблиця 2.3 – Переваги та недоліки використання платформи Docker

Переваги використання	Недоліки використання
План контейнера передбачає повну стандартизацію. Контейнерізована програма не залежить від архітектури або ресурсів хоста;	При великій масштабованості та навантаженні необхідно дуже чітке та якісне налаштування систем.

Кінець таблиці 2.3 – Переваги та недоліки використання платформи Docker

Переваги використання	Недоліки використання
Ще однією перевагою є абстрагування програми від хоста, можливість простого та лінійного масштабування. Таким чином, кілька контейнерів можна запускати одночасно на одній машині та на тестовому сервері. Їх також можна масштабувати в робочому середовищі.	Процес видалення докер контейнерів займає велику кількість часу і вимагає чимало операцій введення/виведення.
Завдяки використанню контейнерів, можна прив'язати всі компоненти та залежність до додатку, що дозволяє працювати як з цілісним об'єктом. На хості не потрібні встановлення додаткових компонентів або залежність для запуску програми, яка знаходиться всередині контейнера.	Додаткові надбудови у системі в будь-якому випадку призводять до збільшення навантаження та витрат ресурсів.
Незважаючи на те, що ізоляція в контейнерах не досягає такого ж рівня, як у віртуалізації, вона має легке середовище виконання та є ізоляційною на рівні процесів. Коли контейнер працює на тому самому ядрі, він може запускатися дуже швидко. Запуск сотень контейнерів на робочій машині не буде проблемою.	Контейнеризація є надбудовою над ОС, тим самим ускладнюючи реалізацію завдання.

2.4.4 Postman – інструмент для тестування та налагодження API

Postman є потужним інструментом для тестування API, який значно полегшує розробку та налагодження мікроконтролерних систем.

Postman – це інтегроване середовище розробки (IDE) для API, яке дозволяє тестувати запити HTTP. Воно має підтримку багатьох типів запитів, таких як GET, POST, PUT і DELETE, і пропонує зручний інтерфейс для роботи з API. Нижче наведена таблиця у якій описано переваги та недоліки використання Postman (табл. 2.4).

Таблиця 2.4 – Переваги та недоліки використання Postman

Переваги використання	Недоліки використання
За допомогою Postman легко створювати та надсилати HTTP-запити до API. Це корисно для тестування кінцевих точок	Хоча Postman добре підходить для базового тестування API, йому може бракувати можливостей для складних сценаріїв тестування.
Щоб автоматизувати процес налагодження, Postman дозволяє додавати скрипти тестів для перевірки відповідей API.	Postman може стати повільним і вимагати великих обсягів оперативної пам’яті, що може вплинути на продуктивність системи, коли працює з великими наборами даних або колекціями запитів.
Створення та тестування запитів полегшується простим інтерфейсом Postman, що економить час розробникам і зменшує кількість помилок.	Інтерфейс Postman може стати складним для навігації, особливо для новачків. Велика кількість налаштувань та функцій може вимагати багато часу для освоєння.

Хоча Postman є потужним і корисним інструментом для тестування API, його недоліки варто враховувати під час вибору інструменту для проєкту. Для складніших або більших проєктів може знадобитися розширення можливостей Postman або додаткові інструменти. Розуміння цих обмежень допоможе ефективно використовувати Postman і успішно розробляти та тестувати API.

2.4.5 Протокол передачі даних HTTP

Всесвітня павутина базується на протоколі передачі даних HTTP (HyperText Transfer Protocol). Цей протокол дозволяє нам використовувати наш браузер для доступу до веб-сайтів і взаємодії з ними, наприклад, перемикатися між сторінками, завантажувати файли та переглядати зображення, обмінюватися повідомленнями та оплачувати покупки.

Протокол передачі гіпертексту, або текстових документів із посиланнями на інші документи, відомий як HTTP. Наразі його можна використовувати для передачі будь-яких типів даних.

Протокол HTTP працює за допомогою технології клієнт-сервер. Користувач відправляє запит на сервер, а програма спеціального типу обробляє запит, створює відповідь і повертає його клієнту.

Хоча браузер зазвичай виконує функцію клієнта, інші програми також можуть виконувати ці функції. Наприклад, пошуковий робот, який сканує веб-сайти для індексації пошуковими системами. Веб-сервер – це програма на фізичному сервері, на якому зберігається сайт.

Наприклад, коли ваш браузер намагається відобразити сторінку, він надсилає запит серверу на отримання HTML-документа з його вмістом. Далі браузер переглядає документ і запитує файли, такі як стилі елементів, зображення та скрипти, необхідні для відображення сторінки. Після цього браузер збирає всі ці елементи відповідно до вказівок у HTML-документі та показує результат на екран комп’ютера.

Коли браузер передає певні дані серверу, це ще один приклад. Згадайте пароль і логін вашого облікового запису. Щоб сформувати HTML-документ з головною сторінкою облікового запису, сервер перевіряє деталі входу в спеціальній таблиці всередині бази даних, де зберігаються дані всіх користувачів. Якщо у таблиці є користувач із зазначеними даними, сервер формує HTML-документ і відправляє браузеру. Передбачений спеціальний сценарій для ситуацій, коли такого користувача немає або дані входу неправильні: надіслати повідомлення з помилкою, яке відображатиме браузер на екрані. Схема HTTP запиту/відповіді зображена нижче (рис. 2.3).

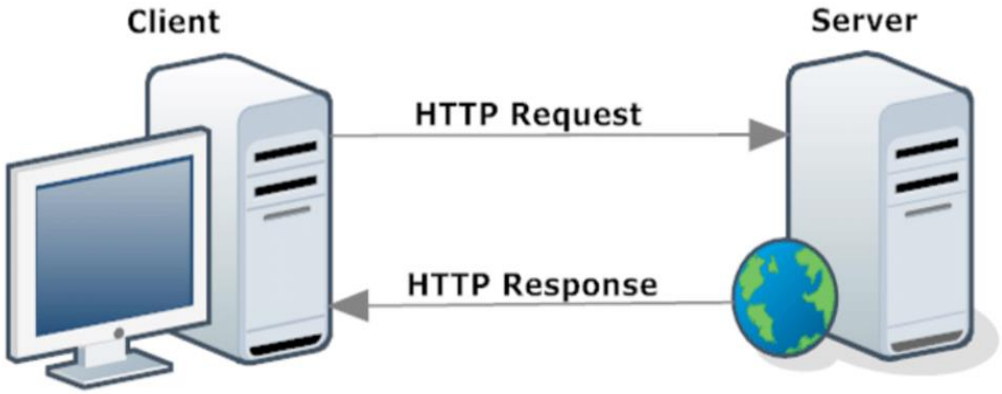


Рисунок 2.3 – Схема HTTP запиту/відповіді

HTTP-запити складаються з наступних частин:

– запитова лінія: містить метод запиту (GET, POST, PUT, DELETE тощо), URI ресурсу та версію протоколу HTTP;

– заголовки: надають додаткову інформацію про запит або про самі дані, які передаються;

– тіло запиту (не обов'язкове): містить дані, що передаються на сервер (наприклад, дані з датчиків мікроконтролера).

HTTP-відповіді мають аналогічну структуру:

– статусна лінія: містить версію протоколу, код статусу (200, 404, 500 тощо) та статусне повідомлення;

– заголовки: надають метадані про відповідь;

– тіло відповіді (не обов'язкове): містить дані, що повертаються клієнту.

Кожен HTTP-метод виконує певні функції. Наприклад, GET використовується для запиту даних з серверу (наприклад, отримання поточних налаштувань системи), POST використовується для відправки даних на сервер (наприклад, надсилання даних з датчиків), PUT використовується для оновлення існуючих ресурсів на сервері, DELETE використовується для видалення ресурсів з сервера.

Протокол HTTP має як переваги, так і недоліки. Нижче наведено таблицю у якій описано переваги та недоліки протоколу HTTP (табл. 2.5).

Таблиця 2.5 – Переваги та недоліки протоколу HTTP

Переваги	Недоліки
Навіть мікроконтролери з обмеженими ресурсами можуть використовувати HTTP, оскільки протокол добре документований і широко підтримуваний.	Для мікроконтролерів з обмеженими обчислювальними можливостями та пам’яттю HTTP-запити можуть бути надто ресурсномісткими.
Усі сучасні пристрої та платформи підтримують HTTP, що полегшує інтеграцію мікроконтролерів з іншими системами.	Затримки передачі даних через мережу, пов’язані з HTTP, можуть бути критичними для деяких додатків, які потребують швидкої реакції.
Серверна частина легко масштабується для обробки великої кількості запитів від різних мікроконтролерів. Це дозволяє створювати великі та розподілені системи моніторингу.	У зв’язку з тим, що HTTP є синхронним протоколом, мікроконтролери повинні чекати відповіді сервера перед тим, як можуть працювати далі.

В системах моніторингу навколишнього середовища HTTP є зручним і необхідним інструментом для взаємодії між мікроконтролерами та сервером. Його головними перевагами є широке використання, просту реалізацію та підтримку безпеки. Однак, важливо враховувати і недоліки протоколу, особливо в контексті обмежень ресурсів мікроконтролерів та вимог до реального часу.

2.4.6 Спрощений мережевий протокол MQTT

MQTT (Message Queue Telemetry Transport) – спрощений мережевий протокол, що працює на TCP/IP (набір протоколів мережі Інтернет). Використовується для обміну повідомленнями між пристроями за принципом видавець-підписник.

У контексті мікроконтролерної системи моніторингу компонентів довкілля, MQTT використовується для ефективного збору, передачі та обробки даних в реальному часі.

MQTT використовує модель публікації-підписки, тобто пристрої можуть надсилати або публікувати повідомлення на певні теми, а інші пристрої можуть підписуватися на ці теми, щоб отримати повідомлення. Це дозволяє створювати розподілені системи сполучення, у яких пристрої можуть безпосередньо взаємодіяти один з одним.

MQTT використовує проміжне програмне забезпечення, відоме як брокери повідомлень, для полегшення передачі повідомлень між різними пристроями. Після отримання повідомлень від пристроїв, які публікують, брокер передає їх підписаним пристроям. Нижче на рисунку наведена схема роботи MQTT протоколу (рис. 2.4).

MQTT підтримує три рівні QoS (Quality of service), які визначають рівень надійності доставки повідомлень: QoS 0 (найменший рівень надійності), QoS 1 (гарантована доставка принаймні одного разу) та QoS 2 (повна гарантія доставки).

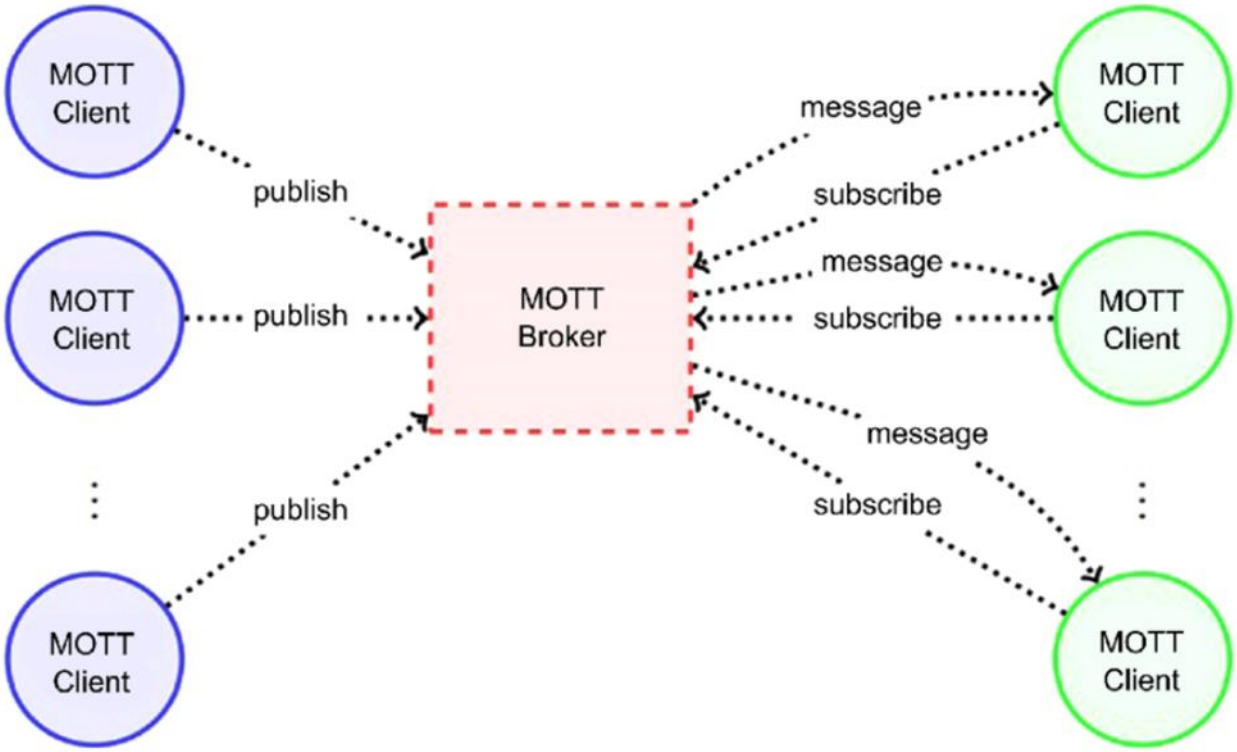


Рисунок 2.4 – Схема роботи MQTT протоколу

MQTT визначає методи, щоб вказати бажану дію, яка повинна виконуватися на ідентифікованому ресурсі. Чим є цей ресурс, будь то вже наявні дані або дані, що генеруються динамічно, залежить від реалізації сервера. Часто ресурс відповідає файлу або результату виконання якогось файлу, розміщеного на сервері. Існують такі методи:

– connect (з’єднати). Чекає встановлення з'єднання з сервером;

– disconnect (роз’єднати). Чекає доки клієнт MQTT закінчить будь-яку роботу, що має зробити, і доки роз'єднається TCP/IP сесія;

– subscribe (підписатися). Чекає на завершення методу Subscribe чи UnSubscribe;

– unsubscribe (відписатися). Відписатися: просить сервер відписати клієнта від одної або кількох тем;

– publish (публікувати). Одразу повертається в потік виконання додатку після того, як передасть запит клієнту MQTT;

Нижче наведена схема потоку транзакцій публікації та підписки MQTT (рис. 2.5).

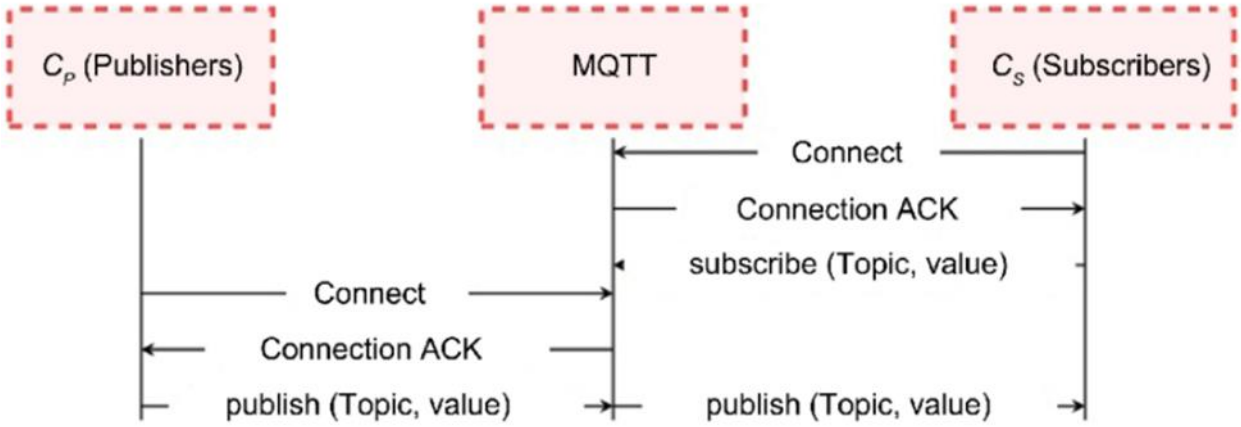


Рисунок 2.5 – Схема потоку транзакцій публікації та підписки MQTT

Мережевий протокол MQTT має свої переваги та недоліки. Нижче наведена таблиця у якій описані плюси та мінуси цього протоколу (табл. 2.6).

Таблиця 2.6 – Переваги та недоліки мережевого протоколу MQTT

Переваги	Недоліки
MQTT розроблений для використання в умовах обмежених ресурсів, що робить його ідеальним для вбудованих систем та мікроконтролерів.	Використання MQTT через використання брокера повідомлень може призвести до більших витрат на мережевий трафік і може призвести до збільшення складності конфігурації та підтримки системи.
Що важливо для систем моніторингу довкілля, це можливість швидкої та надійної передачі даних у режимі реального часу, яку пропонує MQTT.	Неправильна конфігурація MQTT може призвести до проблем з безпекою.

Кінець таблиці 2.6 – Переваги та недоліки мережевого протоколу MQTT

Переваги	Недоліки
Що важливо для систем моніторингу довкілля, це можливість швидкої та надійної передачі даних у режимі реального часу, яку пропонує MQTT.	Неправильна конфігурація MQTT може призвести до проблем з безпекою.
Можливість масштабування брокера MQTT дозволяє обробляти велику кількість підключених пристроїв і обмінюватися даними між ними.	Налаштування та підтримка брокера MQTT може виявитися складним завданням, особливо для нових користувачів. Для забезпечення максимальної продуктивності та безпеки необхідне ретельне планування та конфігурація для оптимальної працездатності та безпеки.

Підбиваючи підсумок, MQTT є потужним і ефективним протоколом для мікроконтролерних систем моніторингу довкілля. Однак для досягнення оптимальної продуктивності та безпеки необхідні ретельне планування та правильна конфігурація. Він дуже добре підходить для широкого спектру застосувань Інтернету речей і моніторингу довкілля через свою простоту, ефективність і підтримку реального часу.

2.4.7 Базові засоби безпеки

Обмеження доступу та авторизація є важливими компонентами систем безпеки серверної частини мікроконтролерної системи моніторингу компонентів довкілля. Метою авторизації є перевірка прав користувачів на доступ до ресурсів

системи, наприклад, визначення, чи має користувач право виконувати певні дії або отримати певну інформацію.

Користувачі повинні пройти процес аутентифікації, щоб система могла ідентифікувати їх. Введення логіну та пароля, використання токенів або інших методів можуть бути частиною цього.

Після успішної аутентифікації система повинна перевірити, які дії або ресурси має право виконувати користувач. Це може залежати від групи, ролей користувачів або інших критеріїв.

Логування подій є не менш важливою частиною систем моніторингу та безпеки. Логи (англ. – Log file) – це спеціальний текстовий файл з журналом подій про роботу системи у хронологічному порядку. Логування – це процес запису подій, які відбуваються в програмній системі, щоб відстежувати та аналізувати їх для вирішення проблем, відновлення даних або реагування на інциденти.

Для ефективного логування подій необхідно визначити, які події або дії в системі необхідні для логування. Це можуть бути помилки, недозволений доступ, видалення або зміна важливих даних та інші події, які вимагають ретельної перевірки.

Також необхідно встановити рівень деталізації для логів. Важливо отримувати достатню кількість інформації для аналізу, але при цьому не перевантажувати систему зайвою інформацією.

Потрібно вирішити у якому форматі зберігати логи. Це може бути в текстовому форматі, як-от JSON або CSV, або за допомогою спеціалізованих систем логування.

Логування може значно спростити життя, якщо вміло користуватись та працювати з ними. Але і не є запорукою вирішення будь-яких проблем, пов’язаних з історією користувацьких чи системних дій.

Захист від помилок у даних є критично важливим для будь-якої програмної системи, особливо для мікроконтролерної системи моніторингу компонентів навколишнього середовища. Некоректні або неправильно оброблені дані можуть

призвести до багатьох проблем, таких як уразливості безпеки, збої системи або некоректні результати аналізу.

Необхідно переконатись, що всі надіслані дані відповідають очікуваним форматам і обмеженням. Для цього можна використовувати бібліотеки валідації, вбудовані функції перевірки та регулярні вирази для цього.

Також потрібно враховувати потенційні помилки обробників даних і створювати механізми для усунення цих помилок. Це може включати виведення повідомлень про помилки, логування їх для аналізу або повернення відповідних статусів помилок клієнту.

Усі ці аспекти є важливими для забезпечення коректної роботи системи.

2.5 Висновки

Завдяки своїй асинхронності, високій продуктивності та здатності ефективно обробляти великі обсяги вхідних даних у режимі реального часу – платформа Node.js була визнана основною платформою для розробки серверної частини.

Express.js забезпечує зручність у розробці та підтримці структури проєкту.

NPM значно спрощує процес розробки, керуючи залежностями та пакетами. NPM дозволяє отримати доступ до великої кількості модулів і бібліотек, що дозволяє швидко інтегрувати необхідні функціональні можливості, зменшуючи час розробки.

Застосунок Docker було обрано для контейнеризації, оскільки він полегшує розгортання, керування та масштабування серверної частини. Docker підвищує безпеку та знижує ризик конфліктів залежностей, створюючи відокремлене середовище для виконання додатків.

Postman використовується для тестування API, що дозволяє швидко та ефективно перевіряти коректність роботи серверної частини, забезпечуючи високу якість коду та зручність у процесі розробки.

MQTT передає дані від мікроконтролерів до серверної частини, тоді як HTTP забезпечує стаціонарну комунікацію між клієнтами та сервером. MQTT був обраний, оскільки він простий, надійний і ідеально підходить для пристроїв з обмеженими ресурсами.

Обмеження доступу та авторизацію встановлено для захисту даних і контролю доступу до різних компонентів системи. Авторизація публікацій MQTT і підписки на топіки забезпечують додатковий захист.

Валідація даних використовується для перевірки правильності вхідних даних, що зменшує ймовірність обробки неправильних або шкідливих даних.

Логування подій використовується для відстеження важливих подій у системі, що допомагає у виявленні та аналізі проблем і дає можливість реагувати на інциденти безпеки.

Таким чином, засоби розробки програмного забезпечення, які були вибрані, гарантують ефективну, масштабовану та надійну роботу серверної частини мікроконтролерної системи моніторингу компонентів довкілля. Крім того, вони гарантують високу якість і безпеку системи; це дуже важливо для успішного виконання завдань моніторингу та аналізу даних довкілля.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ МІКРОКОНТРОЛЕРНОЇ СИСТЕМИ МОНІТОРИНГУ КОМПОНЕНТІВ ДОВКІЛЛЯ

3.1 Вступ

У сучасному світі все більше людей звертають увагу на екологічні проблеми та збереження довкілля. Мікроконтролерні системи моніторингу довкілля стають все більш поширеними з метою виявлення та контролю різних параметрів середовища, таких як температура, вологість, якість повітря та концентрація шкідливих речовин. Тим не менш, одним із основних викликів є створення ефективної та надійної серверної частини. Ця частина забезпечує збір, обробку та аналіз даних із датчиків, встановлених на місцях спостереження.

Несправності на сервері мікроконтролерної системи моніторингу навколишнього середовища можуть призвести до отримання неточних даних, втрати інформації та зниження ефективності системи. Такі проблеми можуть включати неправильну обробку або втрату даних, недостатню масштабованість для обробки великої кількості даних і проблеми з безпекою, такі як атаки та несанкціонований доступ до даних. Крім того, неправильна робота серверної частини може призвести до збоїв у роботі системи та затримок у наданні інформації про стан навколишнього середовища.

Результати моніторингу довкілля можуть бути неправильними через неправильну обробку даних або втрати. Наприклад, неправильна інтерпретація даних про якість повітря може призвести до помилкових висновків щодо забруднення атмосфери.

Було б важко зібрати та обробити велику кількість даних в реальному часі. Обробка та аналіз даних можуть зайняти багато часу через відсутність ефективних механізмів масштабування.

Необхідно мати надійну серверну інфраструктуру, що забезпечує безперебійну роботу системи моніторингу. Відсутність такої надійності може призвести до втрати зв'язку з мікроконтролерами або втрати даних.

Узагальнюючи, ефективна серверна інфраструктура є важливим елементом у забезпеченні надійності та ефективності мікроконтролерних систем моніторингу довкілля.

3.2 Архітектура програмного забезпечення для серверної частини

3.2.1 Брокер MQTT

Брокер MQTT – це посередник, який дає змогу клієнтам MQTT спілкуватися. Функціонуючи як центральний хаб, брокер MQTT ефективно обробляє потік повідомлень між пристроями та програмами. Зокрема, брокер MQTT отримує повідомлення, опубліковані клієнтами, фільтрує повідомлення за темами та розповсюджує їх передплатникам.

Використання брокерів MQTT для ввімкнення моделі зв’язку публікації/підписки (pub/sub) допомагає зробити MQTT високоефективним і масштабованим протоколом.

Цей зв’язок за допомогою брокера забезпечує легкий, масштабований і надійний механізм для пристроїв для обміну інформацією в мережевому середовищі, відіграючи вирішальну роль у створенні ефективних і швидко реагуючих екосистем Інтернету речей та інших розподілених програм.

Оскільки він відповідає за полегшення зв’язків між клієнтами MQTT – видавцями (publisher) та підписниками (subscriber) – брокер MQTT відіграє важливу роль в архітектурі MQTT.

Ось деякі з основних причин, чому брокери MQTT є важливими:

1) маршрутизація повідомлень: на основі тем, які вони підписали, брокер MQTT отримує повідомлення від видавців і направляє їх до відповідних

підписників. Це гарантує швидку та точну доставку повідомлень без необхідності, щоб клієнти мали пряме зв’язок один з одним;

2) масштабованість: брокери MQTT здатні обробляти велику кількість підключень одночасно, що важливо для сценаріїв спілкування IoT (Internet of Things) і M2M (Machine-to-Machine), у яких можуть бути тисячі або навіть мільйони пристроїв, які взаємодіють. Здатність брокера керувати цими з’єднаннями та повідомленнями гарантує, що протокол MQTT масштабується ефективно;

3) безпека: брокери MQTT можуть виконувати такі заходи безпеки, як автентифікація та шифрування, щоб переконатися, що дані, що передаються між пристроями та програмами IoT, залишаються безпечними;

4) інтеграція: брокери MQTT можуть інтегруватися з іншими протоколами зв’язку та хмарними платформами, щоб забезпечити повне рішення IoT;

5) управляння сеансами: брокер MQTT відповідає за керування сеансами клієнтів. Це включає зберігання інформації про підписку клієнта та обробку повідомлень, які зберігаються, щоб доставити клієнтам, коли вони виходять в Інтернет. Ця функція керування сеансом гарантує, що повідомлення не будуть втрачені клієнтами, коли вони відключатимуться від брокера та знову підключатимуться.

Алгоритм роботи брокера MQTT можна розділити на кілька кроків:

1) створення брокера: створення TCP (Transmission Control Protocol) сервер, ініціалізація брокера MQTT на цьому сервері;

2) обробка підключених клієнтів: очікування підключень нових клієнтів до брокера, ідентифікація клієнта та надання йому ідентифікатора;

3) авторизація клієнтів: визначення правил авторизації для клієнтів, які намагаються публікувати або підписуватися на певні топіки, перевірка даних авторизації клієнта перед дозволом на публікацію або підписку.

4) маршрутизація повідомлень: прийом та обробка вхідних MQTT повідомлень, визначення клієнтів, яким необхідно відправити ці повідомлення, і доставка їх до відповідних клієнтів;

5) взаємодія з клієнтами: відправлення підтверджень про те, що повідомлення клієнтів були надіслані, відправлення повідомлень в разі зміни стану підписки на топік;

6) обробка відключень клієнтів: контроль відключень клієнтів від брокера MQTT, при необхідності видалення підписок та інших клієнтських даних;

7) логування подій: ведення журналу подій для відслідковування дій брокера та клієнтів.

Нижче наведена схема взаємодії клієнта та брокера MQTT (рис. 3.1).

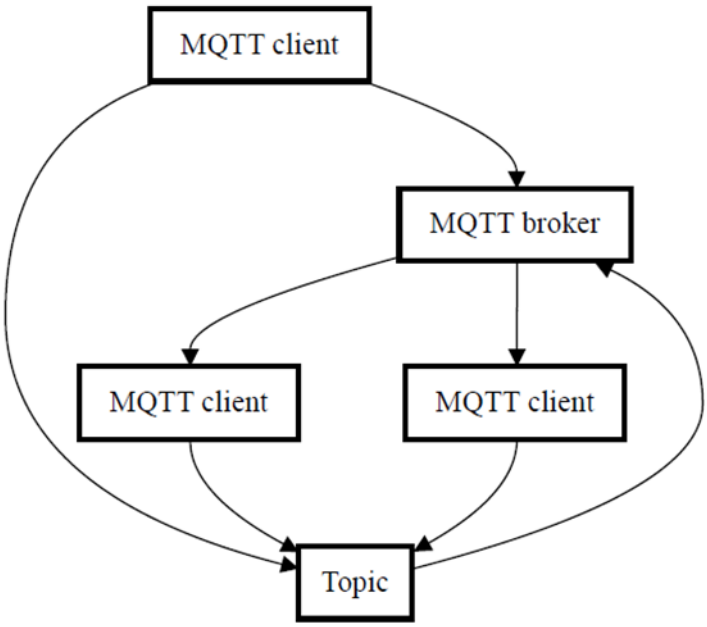


Рисунок 3.1 – Схема взаємодії клієнта та брокера MQTT

MQTT-брокер, який використовується у системі моніторингу компонентів довкілля, гарантує надійний, захищений і ефективний обмін даними між сенсорами та сервером. Він гарантує високу продуктивність і гнучкість системи, організовуючи моніторинг і аналіз даних у режимі реального часу. Забезпечити всі ці функції було б значно складніше без MQTT брокера.

3.2.2 Express.js сервер

Express – це мінімалістичний та гнучкий фреймворк для веб-застосунків, побудованих на Node.js, що надає широкий набір функціональності.

Express.js дозволяє створювати маршрути для обробки HTTP-запитів.

Методи Express.js можуть читати параметри запиту, обробляти дані запиту та створювати відповіді з різними типами контенту, такими як текст, HTML (HyperText Markup Language) і JSON (JavaScript Object Notation).

Проміжне програмне забезпечення може виконувати спільні завдання, такі як аутентифікація, авторизація, ведення журналів, тощо, перед тим, як запит досягне обробника маршруту, завдяки Express.js.

Express.js має велику екосистему сторонніх пакетів, які можна використовувати для розширення його можливостей.

Система моніторингу компонентів навколишнього середовища залежить від Express.js сервера, який виконує багато важливих завдань для обробки та надання даних клієнтам. Ось декілька важливих функцій, які виконує Express.js сервер:

1) Express.js забезпечує створення RESTful API (інтерфейс, що використовуються двома комп'ютерними системами для безпечного обміну інформацією через Інтернет), яке дозволяє клієнтам отримувати доступ до даних, що зберігаються на сервері;

2) користувачі можуть легко надсилати запити та отримувати відповідні відповіді з Express.js. Наприклад, користувачі можуть отримувати останні вимірювання або запитувати дані за певний період часу;

3) Express.js полегшує обробку та форматування різноманітних даних;

4) викорситовуючи Express.js можна легко додавати нові маршрути та функції. Якщо потрібно додати новий API-ендпоінт, просто додайте новий маршрут;

5) ініціалізація та запуск сервера з Express.js дуже прості. Достатньо кількох рядків коду для створення сервера та запуску його на вказаному порту.

Нижче наведена блок-схема, яка описує процес обробки HTTP-запитів Express.js сервером (рис. 3.2).

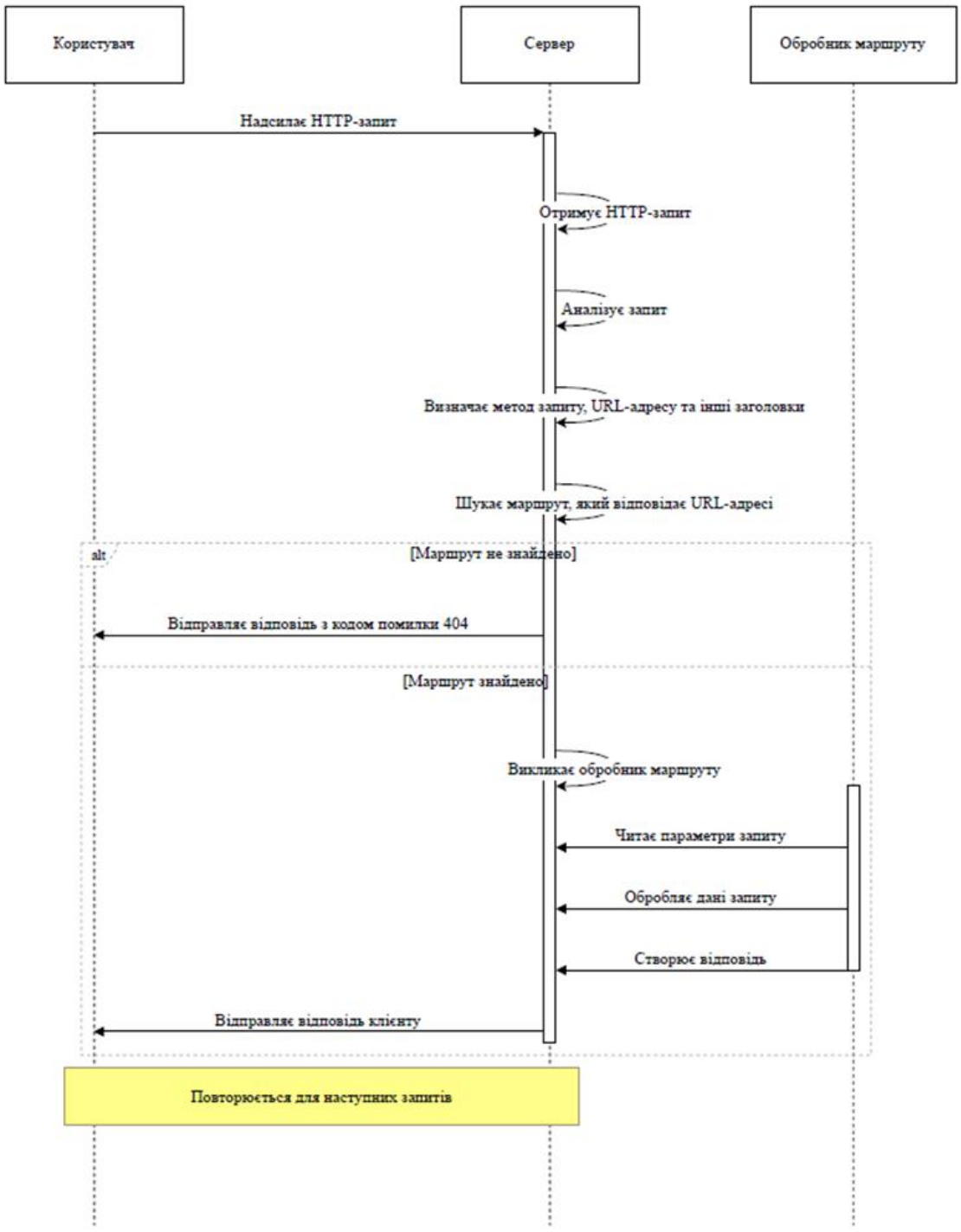


Рисунок 3.2 – Блок-схема, яка описує процес обробки HTTP-запитів Express.js сервером

Алгоритм роботи Express.js можна розділити на такі кроки:

1) користувач використовує Node.js для запуску Express.js сервера. Сервер створює екземпляр програми Express і налаштовує його параметри, сервер визначає порти, через які він буде обробляти HTTP-запити;

2) Express.js приймає HTTP-запит від користувача. Далі, завдяки аналізу запиту сервер визначає його метод (GET, POST, PUT, DELETE тощо), а також URL-адресу та інші заголовки. Потім, сервер шукає маршрут, який відповідає URL-адресі запиту;

3) Express.js викликає обробник маршруту, якщо знайдено правильний маршрут. Функція, яка виконується для обробки запиту, називається обробником маршруту. Обробник маршруту має здатність переглядати параметри запиту, обробляти дані запиту та генерувати відповідь;

4) обробник маршруту створює об’єкт відповіді, який містить тіло відповіді, заголовки та код стану HTTP. Потім, Express.js надсилає відповідь клієнту;

5) сервер продовжує обробляти HTTP-запити та прослуховувати їх за допомогою кроків, описаних вище.

Отже, Express.js є потужним і універсальним інструментом, який ідеально підходить для розробки серверної частини системи моніторингу компонентів довкілля на мікроконтролері. Його варто використовувати через його масштабованість, гнучкість, спільноту та інтеграцію з іншими відомими бібліотеками.

3.2.3 Файлова система

Файлова система – це структура, за допомогою якої операційна система здійснює організацію та управління файлами в сховищі, наприклад, жорсткому диску, твердотільному накопичувачі (SSD) або USB-накопичувачі. Саме вона визначає особливості зберігання та організації даних, а також доступу до них. Різні

файлові системи мають свої особливості та інколи навіть прив’язані до певних операційних систем або пристроїв.

Файлова система використовується для зберігання та доступу до даних, що зчитуються з мікроконтролерів. У нашому випадку це дані про показники довкілля, які зберігаються в текстовому файлі.

Файлова система може використовуватися в серверній частині мікроконтролерної системи моніторингу компонентів довкілля для зберігання різних типів даних:

1) дані вимірювань і моніторингу: файлова система може зберігати результати вимірювань, датчики та інші дані, зібрані мікроконтролерами. Ці дані можуть бути використані для аналізу, статистики та створення звітів;

2) конфігураційні файли: файлова система містить деякі налаштування та параметри системи. Налаштування мережі, параметри зв’язку з сервером та інші конфігураційні дані можуть бути частиною цього;

3) логи та журнали: файлова система може збирати журнали та логи діяльності мікроконтролерів та інших пристроїв. Це може допомогти в аналізі роботи системи та пошуку проблем.

Загалом, у серверній частині мікроконтролерної системи моніторингу компонентів довкілля файлова система відіграє важливу роль у збереженні різноманітних даних і ресурсів. Логи, конфігураційні файли, дані для спостереження та вимірювання зберігаються в файлових системах. Для аналізу, спостереження та управління системою файлова система надає простий спосіб доступу до цих даних.

3.2.4 MQTT клієнт

MQTT клієнт – це скрипт або програмне забезпечення, яке використовується для підключення до брокера MQTT та взаємодії з ним. Цей клієнт можна включити у програмний код або запустити як самостійний процес.

Основною функцією MQTT клієнтів є надсилання та отримання повідомлень за допомогою протоколу MQTT. Клієнт може вибирати між підпискою на топіки для отримання повідомлень (підписка) і публікацією повідомлень на певних топіках (публікація).

Далі показана схема роботи MQTT клієнта (рис. 3.3).

Рисунок 3.3 – Схема роботи MQTT клієнта

MQTT клієнт виконує ще декілька важливих функцій:

1) клієнт використовує параметри підключення, які включають URL брокера, ідентифікаційні дані (якщо необхідно) та порт для з’єднання з MQTT брокером;

2) за допомогою певних топіків клієнт може надсилати повідомлення на брокер. Повідомлення можуть бути представлені в будь-якому форматі, але найчастіше вони представлені у вигляді JSON-об’єкту або у текстовому форматі;

3) клієнт може підписатися на один або кілька топіків брокера, щоб отримувати повідомлення, надіслані на ці топіки іншими клієнтами або пристроями;

4) клієнт може встановити обробники подій, які будуть обробляти повідомлення. Реакція на нові дані, логування даних, виконання дій на основі отриманих даних, тощо можуть бути частиною цього;

5) клієнти можуть контролювати своє підключення до брокера, виконувати повторне підключення, якщо втрачається з'єднання, або реагувати на інші події, пов'язані з брокером MQTT або мережею.

Отже, у серверній частині системи моніторингу довкілля клієнт MQTT відіграє важливу роль у зберіганні, обробці та аналізі цих даних, а також у забезпеченні надійного та ефективного обміну даними між мікроконтролерами та сервером.

3.3 Опис реалізації

3.3.1 Створення та налаштування MQTT брокера

Створення та налаштування MQTT брокера включає кілька кроків. Для початку необхідно імпортувати необхідні для роботи модулі (рис 3.4).

```
5   const aedes = require('aedes')();
6   const net = require('net');
```

Рисунок 3.4 – Імпорт необхідних модулів

Кожен з модулів має свої конкретні функції:

– "aedes": бібліотека для створення MQTT брокера.

– "net": модуль для створення TCP сервера.

Далі налаштовуємо та запускаємо MQTT брокер (рис. 3.5).

```
const mqttPort = 1883;

// Налаштування MQTT брокера
const server = net.createServer(aedes.handle);
server.listen(mqttPort, function () {
    console.log(`MQTT Broker running on port: ${mqttPort}`);
});
```

Рисунок 3.5 – Налаштування MQTT брокера

Створюємо TCP (Transmission Control Protocol) сервера, який буде обробляти MQTT з'єднання за допомогою бібліотеки “aedes”. Запускаємо сервер на порті 1883.

Також необхідно налаштувати авторизацію та аутентифікацію. Налаштовуємо аутентифікацію користувачів (рис 3.6).

“authorizedUsers” – це об’єкт з іменами користувачів та паролями. Функція “authenticate” перевіряє, чи є користувач у списку авторизованих користувачів і чи правильний пароль.

```
const authorizedUsers = {
    'user1': 'password1',
    'user2': 'password2'
};

aedes.authenticate = (client, username, password, callback) => {
    const isAuthorized = authorizedUsers[username] && authorizedUsers[username] === password.toString();
    if (isAuthorized) {
        client.user = username;
        callback(null, true);
    } else {
        const error = new Error('Authentication failed');
        callback(error, false);
    }
};
```

Рисунок 3.6 – Налаштування аутентифікації користувачів

Реалізовуємо, також, авторизацію публікацій (рис 3.7).

```
aedes.authorizePublish = (client, packet, callback) => {
    if (client.user && packet.topic.startsWith(`sensor/data/${client.user}`)) {
        callback(null);
    } else {
        console.log(`Unauthorized publish attempt by ${client.user}`);
        callback(new Error('Unauthorized publish attempt'));
    }
};
```

Рисунок 3.7 – Налаштування авторизації публікацій

У цій частині коду проводиться перевірка, чи має користувач право публікувати вказану тему.

Не забуваємо і про авторизацію підписок (рис. 3.8).

```
43  aedes.authorizeSubscribe = (client, sub, callback) => {
44      if (client.user && sub.topic.startsWith(`sensor/data/${client.user}`)) {
45          callback(null, sub);
46      } else {
47          console.log(`Unauthorized subscribe attempt by ${client.user}`);
48          callback(new Error('Unauthorized subscribe attempt'));
49      }
50  };
```

Рисунок 3.8 – Налаштування авторизації підписок

У цій частині коду проводиться перевірка, чи має користувач право підписуватись на вказану тему.

Далі, реалізовуємо обробник подій брокера (рис 3.9).

На рисунку 3.9 показана обробка подій, які пов'язані з клієнтами та повідомленнями в мережі MQTT за допомогою бібліотеки Aedes.

Функція з параметром “client” викликається, коли клієнт підключається до брокера. Вона виводить повідомлення про підключення клієнта до консолі, вказуючи ідентифікатор клієнта та ідентифікатор брокера.

Функція з параметром “clientDisconnect”, коли клієнт відключається від брокера. Вона виводить повідомлення про відключення клієнта з консолі, вказуючи ідентифікатор клієнта та ідентифікатор брокера.

Функція з параметром “subscribe” викликається, коли клієнт підписується на одну або декілька тем. Вона виводить повідомлення про підписку клієнта на теми до консолі, вказуючи ідентифікатор клієнта, список підписаних тем та ідентифікатор брокера.

Функція з параметром “unsubscribe” викликається, коли клієнт відписується від однієї або декількох тем. Вона виводить повідомлення про відписку клієнта від

тем до консолі, вказуючи ідентифікатор клієнта, список відписаних тем та ідентифікатор брокера.

Функція з параметром “publish” викликається, коли клієнт публікує повідомлення на тему. Вона виводить повідомлення про публікацію повідомлення клієнтом до консолі, вказуючи ідентифікатор клієнта, тему повідомлення та ідентифікатор брокера.

```
aedes.on('client', function (client) {
    console.log(`[CLIENT_CONNECTED] Client ${(client ? client.id : client)} connected to broker
    ${aedes.id}`);
});

aedes.on('clientDisconnect', function (client) {
    console.log(`[CLIENT_DISCONNECTED] Client ${(client ? client.id : client)} disconnected from the broker
    ${aedes.id}`);
});

aedes.on('subscribe', function (subscriptions, client) {
    console.log(`[TOPIC_SUBSCRIBED] Client ${(client ? client.id : client)} subscribed to topics:
    ${subscriptions.map(s => s.topic).join(',')} on broker ${aedes.id}`);
});

aedes.on('unsubscribe', function (subscriptions, client) {
    console.log(`[TOPIC_UNSUBSCRIBED] Client ${(client ? client.id : client)} unsubscribed to topics:
    ${subscriptions.join(',')} from broker ${aedes.id}`);
});

aedes.on('publish', async function (packet, client) {
    if (client) {
        console.log(`[MESSAGE_PUBLISHED] Client ${(client ? client.id : 'BROKER_' + aedes.id)}
        has published message on ${packet.topic} to broker ${aedes.id}`);
    }
});
```

Рисунок 3.9 – Реалізація обробника подій брокера

3.3.2 Взаємодія з MQTT клієнтом

Реалізація взаємодії з MQTT клієнтом показана нижче (рис. 3.10).

```
const mqtt = require('mqtt');
const brokerUrl = `mqtt://localhost:${mqttPort}`;
const topic = 'sensor/data';

//З'єднання з MQTT брокером
const client = mqtt.connect(brokerUrl)

client.on('connect', () => {
    console.log('Connected to MQTT broker');
    client.subscribe(`${topic}`, (err) => {
        if (!err) {
            console.log(`Subscribed to topic: ${topic}`);
            sendData();
        } else {
            console.error(`Failed to subscribe to topic: ${topic}`, err);
        }
    });
});
```

Рисунок 3.10 – Взаємодія з MQTT клієнтом

Взаємодія з MQTT клієнтом в даному випадку передбачає використання бібліотеки “mqtt”, що дозволяє здійснювати комунікацію з MQTT брокером.

Спочатку, встановлюємо з'єднання з брокером MQTT за допомогою методу “mqtt.connect()”.

Коли клієнт успішно підключається до брокера MQTT, викликається функція, передбачена обробником подій “connect” та виводиться повідомлення про підключення до MQTT брокера.

Після успішного підключення до брокера MQTT відбувається підписка на тему. У цьому випадку підписка відбувається на тему, що вказує на "sensor/data".

Після підписки виконується функція зворотного виклику, де перевіряється наявність помилок.

Якщо підписка відбулася успішно, виводиться відповідне повідомлення у консоль, а також викликається функція “sendData()”, яка, відправляє дані.

Якщо ні, виводиться відповідне повідомлення, яке сповіщає про помилку.

3.3.3 Веб-сервер для доступу до даних

Для надання доступу до даних через HTTP-запити реалізовано веб-сервер з використанням фреймворку Express.js. Для реалізації необхідно виконати декілька дій.

Спочатку, імпортуємо необхідні модулі, а також створюємо додаток Express та налаштовуємо його на прослуховування порту 3000 (рис. 3.11).

```
7   const fs = require('fs');
8   const express = require('express');
9   const app = express();
10  const port = 3000;
```

Рисунок 3.11 – Імпорт необхідних модулів та налаштування серверу

Для комунікації між клієнтом (у нашому випадку мобільним додатком) та сервером в Інтернеті необхідні HTTP-запити. HTTP-запит дозволяє клієнту

(мобільному додатку) отримати останнє зчитування з сенсорів (температура, вологість, AQI, концентрація пилу, наявність газу).

Одним з реалізованих HTTP-запитів є запит для отримання даних про останнє зчитування сенсорів (рис. 3.12).

```
// Запит для отримання останніх вимірів
app.get('/last-reading', (req, res) => {
    fs.readFile('data.txt', 'utf8', (err, data) => {
        if (err) {
            return res.status(500).send('Error reading file');
        }

        const lines = data.trim().split('\n');
        const lastReading = lines[lines.length - 1].split(' ');

        const reading = {
            datetime: `${lastReading[0]} ${lastReading[1]}`,
            temp: parseFloat(lastReading[2]),
            humidity: parseFloat(lastReading[3]),
            aqi: parseFloat(lastReading[4]),
            dustConcentration: parseFloat(lastReading[5]),
            gasLeak: lastReading[6] === '1'
        };

        res.json(reading);
    });
});
```

Рисунок 3.12 – HTTP-запит для отримання даних про останнє зчитування сенсорів

Використовуємо метод “GET” для маршруту “/last-reading”, який повертає останній запис у файлі “data.txt” (розбиває його на компоненти та повертає як JSON).

Також був створений запит для отримання погодинних зчитувань за конкретну дату. Цей HTTP-запит дозволяє отримати дані про зчитування сенсорів за кожну годину вказаної дати (рис. 3.13).

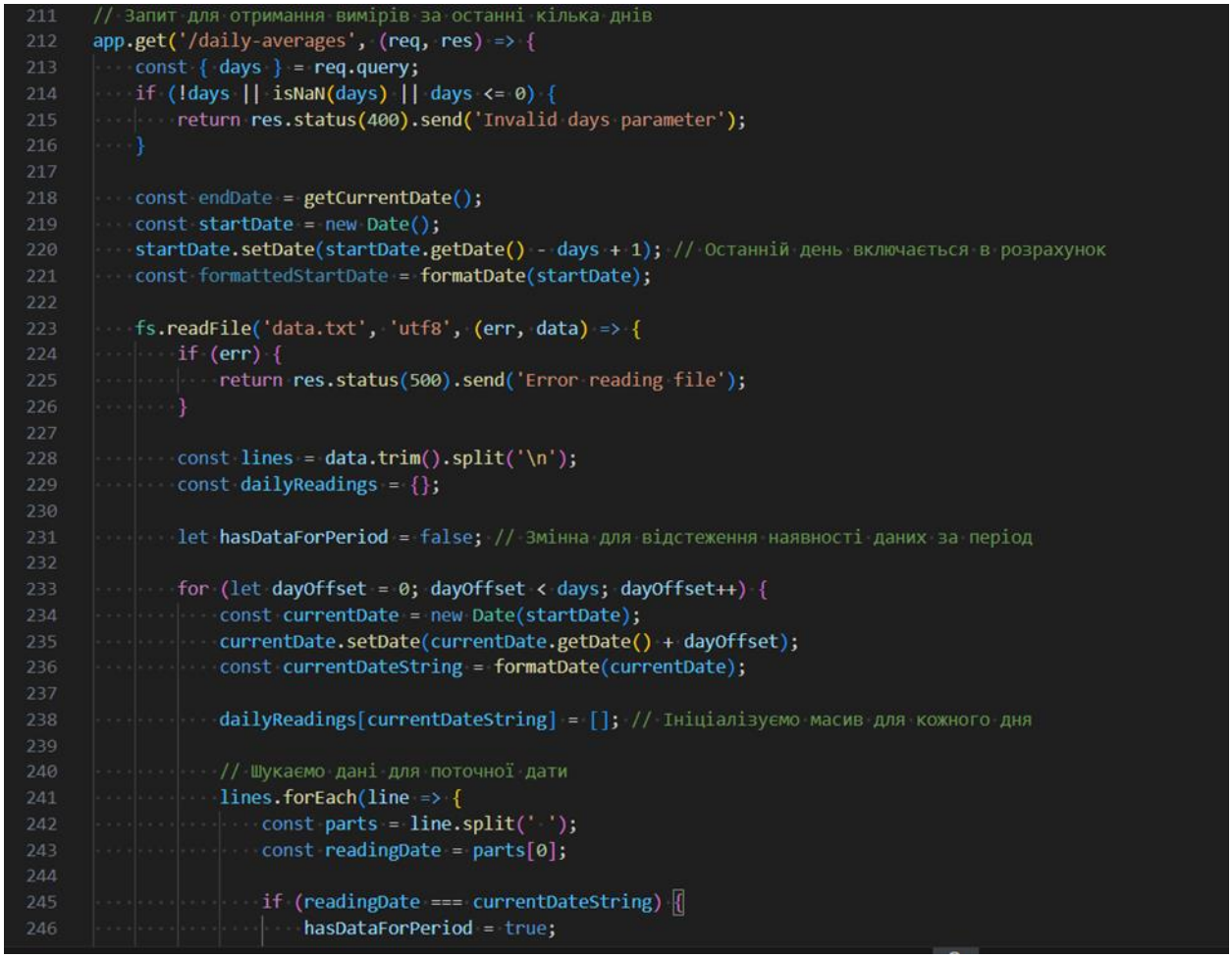
Для реалізації цього запиту був використаний метод “GET” для маршруту “/hourly-readings”, який обробляє параметр дати та повертає відповідні дані.

```
app.get('/hourly-readings', (req, res) => {
    const { date } = req.query;
    if (!date || !isValidDate(date)) {
        return res.status(400).send('Invalid date format');
    }

    fs.readFile('data.txt', 'utf8', (err, data) => {
        if (err) {
            return res.status(500).send('Error reading file');
        }

        const lines = data.trim().split('\n');
        const hourlyReadings = {};

        for (let hour = 0; hour < 24; hour++) {
            hourlyReadings[hour] = {
                temp: [],
                humidity: [],
                aqi: [],
                dustConcentration: []
            };
        }

        lines.forEach(line => {
            const parts = line.split(' ');
            const readingDate = parts[0];
            const readingTime = parts[1];

            if (readingDate === date) {
                const hour = parseInt(readingTime.split(':')[0]);

                const temp = parseFloat(parts[2]);
                let humidity = parseFloat(parts[3]);
                let aqi = parseFloat(parts[4]);
                let dustConcentration = parseFloat(parts[5]);
```

Рисунок 3.13 – HTTP-запит для отримання погодинних зчитувань за конкретну дату

Третім реалізованим запитом є HTTP-запит, який дозволяє отримати середньодобові значення за вказану кількість днів.

Для його реалізації використовуємо метод “GET” для маршруту “/daily-averages”, який обробляє параметр кількості днів з запиту та повертає відповідні середні значення (рис. 3.14).



```
// Запит для отримання вимірів за останні кілька днів
app.get('/daily-averages', (req, res) => {
    const { days } = req.query;
    if (!days || isNaN(days) || days <= 0) {
        return res.status(400).send('Invalid days parameter');
    }

    const endDate = getCurrentDate();
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - days + 1); // Останній день включається в розрахунок
    const formattedStartDate = formatDate(startDate);

    fs.readFile('data.txt', 'utf8', (err, data) => {
        if (err) {
            return res.status(500).send('Error reading file');
        }

        const lines = data.trim().split('\n');
        const dailyReadings = {};

        let hasDataForPeriod = false; // Змінна для відстеження наявності даних за період

        for (let dayOffset = 0; dayOffset < days; dayOffset++) {
            const currentDate = new Date(startDate);
            currentDate.setDate(currentDate.getDate() + dayOffset);
            const currentDateString = formatDate(currentDate);

            dailyReadings[currentDateString] = []; // Ініціалізуємо масив для кожного дня

            // Шукаємо дані для поточної дати
            lines.forEach(line => {
                const parts = line.split(' ');
                const readingDate = parts[0];

                if (readingDate === currentDateString) {
                    hasDataForPeriod = true;
```

Рисунок 3.14 – HTTP-запит для отримання середньодобових значень за вказану кількість днів

Також для тестування роботи запитів використано Postman. За допомогою Postman можна перевірити API додатка шляхом відправлення запитів і отримання відповідей без написання власного коду (рис. 3.15).

Є можливість спробувати різні HTTP методи (GET, POST, PUT, DELETE) в залежності від того, які функції надає ваш API. Можна використовувати параметризацію для тестування різних значень параметрів запитів.

Також, за допомогою Postman можна перевірити валідність даних та переконатися, що API справляється з неправильними даними, такими як недійсні параметри запиту, пусте тіло тощо.

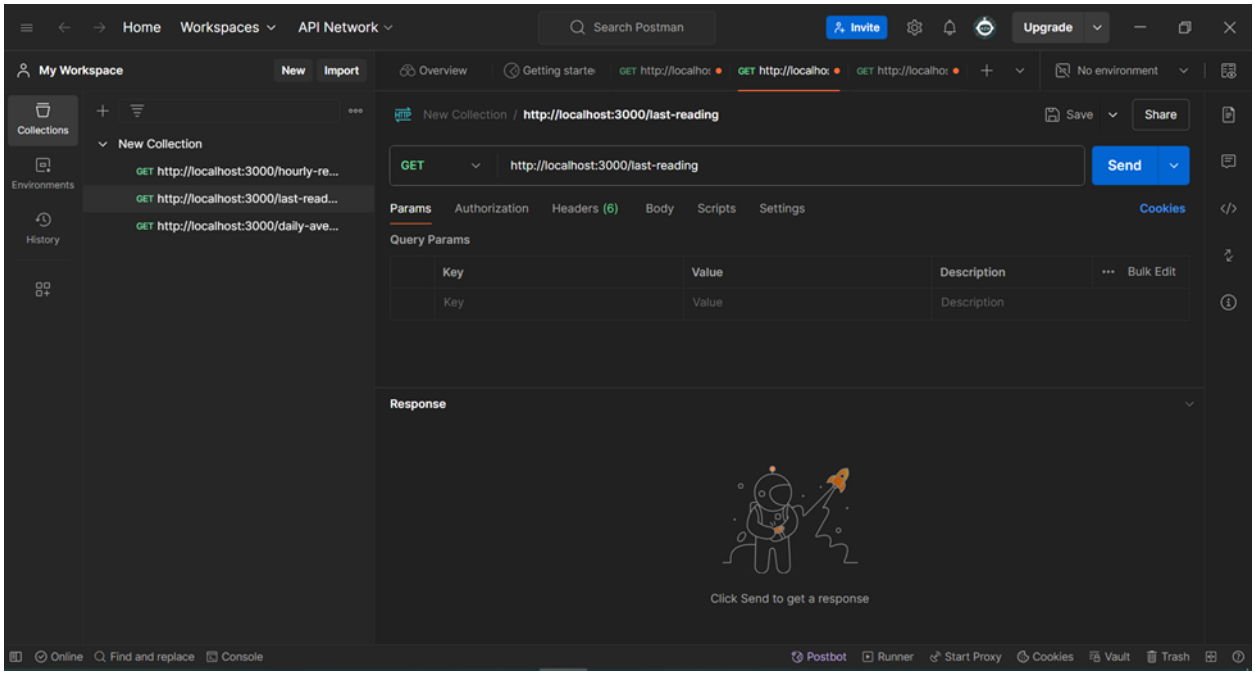


Рисунок 3.15 – Тестування HTTP-запитів за допомогою Postman

Таким чином, HTTP-запити забезпечують зручний та ефективний спосіб для взаємодії клієнтів з сервером, дозволяючи отримувати та відправляти дані у стандартизованому та безпечному форматі.

3.3.4 Обробка та збереження даних

Серверна частина мікроконтролерної системи моніторингу компонентів довкілля включає в себе обробку та збереження даних з різних джерел, таких як MQTT брокер і текстовий файл, за допомогою якого вони зберігаються.

Код створює клієнта MQTT, який підключається до брокера за адресою “mqtt://localhost:${mqttPort}”, де mqttPort - це порт брокера MQTT. Після успішного підключення відбувається підписка на тему “sensor/data”, яка передбачає отримання даних від пристроїв датчиків. Після отримання даних вони обробляються за допомогою обробників подій, які реагують на події підключення клієнта, його відключення, підписку та відписку від теми та публікацію повідомлень. Зазвичай ці дані потім можна обробляти або відображати в інтерфейсі користувача.

Для збереження даних створений веб-сервер на Express, який відповідає на HTTP-запити. Запити “/last-reading”, “/hourly-readings” та “/daily-averages” повертають вимірювані дані, які зберігаються в текстовому файлі з назвою “data.txt”. Дані зчитуються за допомогою функції “fs.readFile()” (рис. 3.16) ми отримуємо дані про виміри. Після цього дані обробляються і повертаються у відповідь на HTTP-запити. Це забезпечує доступ до даних через API сервера Express.

```
116  fs.readFile('data.txt', 'utf8', (err, data) => {
117      if (err) {
118          return res.status(500).send('Error reading file');
119      }
120
121      const lines = data.trim().split('\n');
122      const lastReading = lines[lines.length - 1].split(' ');
123
124      const reading = {
125          datetime: `${lastReading[0]} ${lastReading[1]}`,
126          temp: parseFloat(lastReading[2]),
127          humidity: parseFloat(lastReading[3]),
128          aqi: parseFloat(lastReading[4]),
129          dustConcentration: parseFloat(lastReading[5]),
130          gasLeak: lastReading[6] === '1'
131      };
```

Рисунок 3.16 – Функція для зчитування даних з файлу

Також, було реалізовано допоміжні функції для обробки дати, перевірки її валідності та обчислення середніх значень. Наприклад, функція “isValidDate()” (рис. 3.17) перевіряє правильний формат дати, а функція “calculateHourlyAverage()” (рис. 3.18) розраховує середні значення для годинних вимірювань.

```
101  function isValidDate(dateString) {
102      const regex = /^\d{4}-\d{2}-\d{2}$/;
103      return regex.test(dateString);
104  }
```

Рисунок 3.17 – Функція для перевірки формату дати

```
function calculateHourlyAverage(readings) {
    const total = {
        temp: 0,
        humidity: 0,
        aqi: 0,
        dustConcentration: 0
    };

    readings.temp.forEach(temp => total.temp += temp);
    readings.humidity.forEach(humidity => total.humidity += humidity);
    readings.aqi.forEach(aqi => total.aqi += aqi);
    readings.dustConcentration.forEach(dustConcentration => total.dustConcentration += dustConcentration);

    const count = readings.temp.length;
    return {
        temp: (total.temp / count).toFixed(2),
        humidity: (total.humidity / count).toFixed(2),
        aqi: (total.aqi / count).toFixed(2),
        dustConcentration: (total.dustConcentration / count).toFixed(2),
        gasLeak: readings.gasLeak
    };
}
```

Рисунок 3.18 – Функція для розрахунку середніх значень для годинних вимірювань

Таким чином, інтегровано різні елементи обробки та збереження даних, що надає можливість спостерігати та аналізувати дані, що надходять від датчиків, а також надавати корисну інформацію через веб-сервер за допомогою запитів HTTP.

3.3.5 Робота з Docker

Docker використовується для створення контейнера, в якому буде запущено Node.js додаток.

Є кілька моментів при роботі з Docker, які потрібно зазначити. По-перше, я використовую базовий образ “node:14”, який містить встановлену версію Node.js 14.x. Всі команди виконуються в контексті робочого каталогу “/my-node-app”.

Встановлюємо робочий каталог для контейнера за допомогою “WORKDIR /my-node-app”. Копіюємо файли “package.json”, “package-lock.json” та всі інші файли у поточний каталог контейнера. Це забезпечує доступність файлів, необхідних для встановлення залежностей та запуску Node.js додатку.

Використовуємо команду “RUN npm install” для встановлення залежностей, вказаних у файлі “package.json”. Окремо виконуємо команду “RUN npm install mqtt”, щоб встановити бібліотеку MQTT, яку ми використовуємо.

Після встановлення залежностей потрібно скопіювати всі інші файли з локального контексту у контейнер. Це включає в себе всі необхідні файли, такі як “server.js”, “data.txt” та будь-які інші файли, необхідні для роботи сервера.

Використовуємо “EXPOSE 3000”, щоб відкрити порт 3000 у контейнері. Це необхідно для того, щоб зовнішній світ міг звертатися до додатку через цей порт.

А також, використовуємо CMD ["node", "server.js"] для вказівки команди, яка буде виконуватися при запуску контейнера. Це запускає ваш сервер Node.js з файлу “server.js”.

Нижче наведено Dockerfile (рис. 3.19), який використовується при контейнеризації Node.js додатку.

Dockerfile

```
1   FROM node:14
2
3   WORKDIR /my-node-app
4
5   COPY package*.json ./
6   COPY package-lock*.json ./
7
8   RUN npm install
9
10  RUN npm install mqtt
11
12  COPY . .
13
14  EXPOSE 3000
15
16  CMD ["node", "server.js"]
```

Рисунок 3.19 – Вміст файлу для налаштування роботи з Docker

Далі, для того, щоб запустити Docker контейнер на віддаленому сервері, потрібно переконатись, що у нас є доступ до віддаленого сервера через SSH (Secure Shell) або через інші механізми авторизації. Також, необхідно, щоб на віддаленому

сервері був встановлений Docker. Наступним кроком є використання команди “docker save” для збереження образу у файлі та “scp” або інші методи передачі файлів для пересилання цього образу на віддалений сервер. Якщо Docker образ вже доступний на віддаленому сервері, все готово для запуску контейнера. Використовуємо команду “docker run”. Далі, для керування запущеними контейнерами можна використовувати команди “docker start”, “docker stop”, “docker restart” та інші команди. Нижче наведено рисунок, де зображено працюючий Docker контейнер (рис. 3.20).

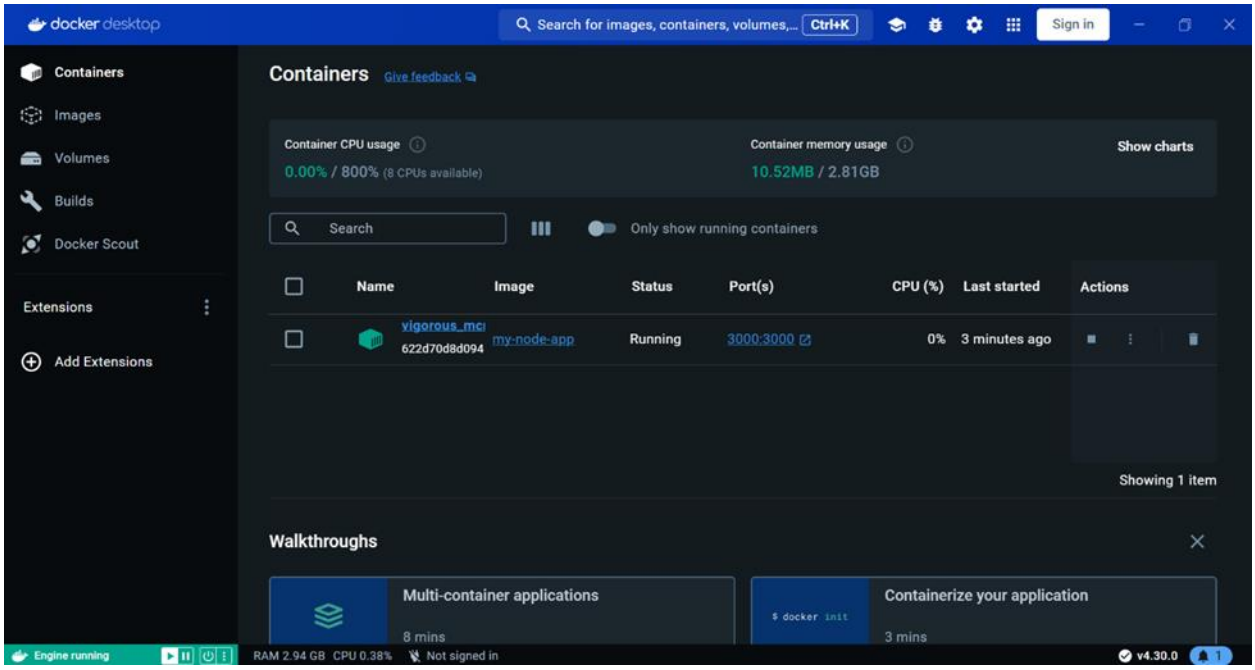


Рисунок 3.20 – Працюючий Docker контейнер

3.4 Напрямки подальшого вдосконалення

Реалізована серверна частина мікроконтролерної системи моніторингу компонентів довкілля вже має досить непоганий функціонал, але є декілька напрямків для подальшого вдосконалення.

По-перше, варто звернути увагу на безпеку системи. Покращення безпеки мікроконтролерної системи моніторингу є важливим кроком, щоб гарантувати захист даних, надійність і стійкість системи. Необхідно використовувати

параметризовані запити для обробки бази даних та обмежувати введення користувачів для уникнення вразливостей типу Cross-Site Scripting (XSS) та SQL ін'єкцій. Також, можна використовувати HTTPS (HyperText Transfer Protocol Secure). Цим ми захистимо передачу даних між клієнтом і сервером від прослуховування.

Не менш важливим аспектом є масштабування. Можна підвищити масштабованість сервера за допомогою кластерів Node.js або інструментів управління процесами, таких як PM2. Для забезпечення коректного балансування навантаження можна реалізувати балансувальник навантаження перед серверами для розподілу навантаження між екземплярами.

Модульне тестування (Unit testing) – це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Створення тестових сценаріїв є корисним для перевірки різних аспектів коду, включаючи обробку запитів API, роботу з файлами та інтеграцію з MQTT брокером.

Для моніторингу працездатності сервера та додатків можна використовувати такі програми як Prometheus та Grafana. Вони допоможуть відстежити продуктивність сервера та виявити проблеми. Хорошим рішенням буде використання ELK Stack (Elasticsearch, Logstash, Kibana) для аналізу та візуалізації логів.

Також, необхідно пам’ятати про резервне копіювання даних. Бажано організувати регулярне резервне копіювання даних та їх зберігання в безпечному місці.

Ці аспекти є важливими, оскільки вони забезпечують всебічні покращення системи, від підвищення продуктивності та безпеки до зручності використання та надійності. За допомогою цих покращень можна побудувати надійну, безпечну та ефективну систему моніторингу компонентів довкілля, яка відповідає вимогам сучасних стандартів і пропонує найкращий досвід користувача.

3.5 Висновки

У розділі «Опис програмної реалізації серверної частини мікроконтролерної системи моніторингу компонентів довкілля» детально описано різні аспекти розробки та роботи серверної частини системи.

Серверна частина, реалізована на базі Node.js та MQTT, забезпечує взаємодію між датчиками, брокером і кінцевими користувачами через REST API, що дозволяє ефективно збирати, зберігати та аналізувати дані про стан довкілля.

MQTT брокер реалізовано на основі бібліотеки Aedes, яка забезпечує надійну і масштабовану обробку повідомлень. Брокер забезпечує прийом та обробку даних від датчиків у реальному часі.

Завдання MQTT клієнта зібрати дані з датчиків і передати їх брокеру. Також, реалізовано періодичну публікацію даних, що забезпечує безперервний моніторинг параметрів довкілля.

За допомогою REST API та Express.js реалізовано різні кінцеві точки для доступу до даних, зокрема останні виміри, погодинні та щоденні середні значення. Також, забезпечено перевірку вхідних даних для підвищення надійності і коректності відповідей.

Реалізовано зберігання даних у текстовому файлі, що дозволяє легко переглядати історію вимірювань.

Також, розглянуто напрямки подальшого вдосконалення системи: покращення безпеки, підвищення масштабованості сервера, запропоновані методи тестування, моніторингу працездатності сервера та організації резервного копіювання даних.

Загалом, якщо використовувати серверну частину мікроконтролерної системи моніторингу компонентів довкілля, можна ефективно та надійно збирати, зберігати та аналізувати дані. Удосконалення, запропоновані для системи, підвищать її продуктивність, безпеку та зручність використання, що сприятиме її розвитку та успішному застосуванню.

ВИСНОВОК

За результатами теоретичних і практичних досліджень було розроблено та впроваджено серверну частину мікроконтролерної системи моніторингу компонентів навколишнього середовища.

У першому розділі було досліджено поточні технології та методи створення систем моніторингу довкілля. Було розглянуто основні частини цих систем, а також їхні переваги та недоліки. Переглянуто сучасні методи обробки, зберігання та передачі даних, а також технології, які гарантують, що системи моніторингу працюють ефективно.

У другому розділі було розроблено архітектуру серверної частини мікроконтролерної системи моніторингу, а також визначено основні модулі системи, а також те, як вони взаємодіють і що потрібно для їх роботи. Схеми показали структуру та процеси обробки даних, а також те, як вони взаємодіють з іншими частинами системи.

У третьому розділі реалізовано програмне забезпечення серверної частини. Розроблено та протестовано основні компоненти системи, такі як механізми зберігання даних, брокер MQTT і веб-сервер на базі Node.js. Тестування системи проводилося з використанням Postman для перевірки коректності API та взаємодії з клієнтськими додатками.

Розроблена система гарантує надійний збір, обробку та зберігання даних від датчиків, що дозволяє швидко отримувати інформацію про стан навколишнього середовища.

Використання технологій MQTT та Node.js дозволяє легко розширювати систему, додаючи нові датчики та збільшуючи функціональність, здебільшого не змінюючи програмний код.

Забезпечено основні механізми безпеки для доступу до серверу та передачі даних. Для підвищення рівня безпеки запропоновано використання протоколів шифрування та аутентифікації.

Система легко адаптується до різних сценаріїв використання, що робить її чудовою для моніторингу різних параметрів навколишнього середовища.

Покращення безпеки, оптимізація обробки даних, інтеграція з іншими сервісами та покращення інтерфейсу користувача — це теми подальшого розвитку системи.

Загалом, реалізована серверна частина мікроконтролерної системи моніторингу компонентів довкілля є ефективним інструментом для моніторингу екологічних параметрів, що забезпечує надійне та своєчасне отримання даних для прийняття обґрунтованих рішень щодо стану навколишнього середовища.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Переваги node JS - artjoker. IT компанія Artjoker - створення сайтів та інтернет маркетинг. URL: https://artjoker.ua/blog/v-chem-preimushchestva-nodejs/ (дата звернення: 23.04.2024).

2. Що таке бази даних, їх призначення та види?. FutureNow. URL: https://futurenow.com.ua/shho-take-bazy-danyh-yih-pryznachennya-ta-vydy/ (дата звернення: 19.02.2024).

3. Що таке радіотехнологія LoRa і як її використовують для передачі данихLoRa й LoRaWAN®: як навчити пристрої спілкуватися з людиною - Jooby. Jooby. URL: https://jooby.eu/uk/blog/radiotekhnolohiya-lora/ (дата звернення: 08.02.2024).

4. Що таке NodeJS - Підручник NodeJS. Підручник NodeJS. URL: https://nodejs.tutorial.in.ua/what-is-nodejs/ (дата звернення: 19.04.2024).

5. 5 types of environmental monitoring | atlas scientific. Atlas Scientific. URL: https://atlas-scientific.com/blog/types-of-environmental-monitoring/ (дата звернення: 23.02.2024).

6. Aclima pro. Aclima. URL: https://www.aclima.io/aclima-pro (дата звернення: 13.02.2024).

7. An introduction to sigfox technology – basics, architecture and security features. Circuit Digest - Electronics Engineering News, Latest Products, Articles and Projects. URL: https://circuitdigest.com/article/what-is-sigfox-basics-architecture-and-security-features (дата звернення: 22.02.2024).

8. Arable Mark – інструмент для обміну даними між приладами. Aggeek. URL: https://aggeek.net/ru-blog/arable-mark--instrument-dlya-obminu-danimi-mizh-priladami (дата звернення: 14.02.2024).

9. Automated greenhouse management with iot: monitoring and controlling | webbylab. webbylab. URL: https://webbylab.com/blog/smart-greenhouse-solutions-iot-based-environmental-monitoring-and-control/ (дата звернення: 18.02.2024).

10. Clinton D. What is Node.js? Server-Side JavaScript Development Basics. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/node-js-basics/ (дата звернення: 18.04.2024).

11. Contributors to Wikimedia projects. Apache flink - wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Apache_Flink (дата звернення: 15.02.2024).

12. IBM expands green horizons effort to fight pollution globally. eWEEK. URL: https://www.eweek.com/database/ibm-expands-green-horizons-effort-to-fight-pollution-globally/ (дата звернення: 14.02.2024).

13. IBM research launches project "green horizon" to help china deliver on ambitious energy and environmental goals. PR Newswire: press release distribution, targeting, monitoring and marketing. URL: https://www.prnewswire.com/news-releases/ibm-research-launches-project-green-horizon-to-help-china-deliver-on-ambitious-energy-and-environmental-goals-265983691.html (дата звернення: 15.02.2024).

14. Internet of things (iot) applications: smart water. Manx Technology Group. URL: https://www.manxtechgroup.com/internet-of-things-iot-applications-smart-water/ (дата звернення: 17.02.2024).

15. Microsoft azure explained: what it is and why it matters. CCB Technology. URL: https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/ (дата звернення: 20.02.2024).

16. Node js розробник: чим займається і що повинен знати. FoxmindEd. URL: https://foxminded.ua/node-js-rozrobnyk/ (дата звернення: 11.04.2024).

17. Redazione Enel X Global. Environmental monitoring systems: types and purposes | Enel X. Enel X. URL: https://corporate.enelx.com/en/stories/2021/02/environmental-monitoring-systems (дата звернення: 28.02.2024).

18. Smart water management iot solution | water quality sensors certified. Libelium. URL: https://www.libelium.com/iot-solutions/smart-water/ (дата звернення: 11.02.2024).

19. Stone Z. Aclima rolls out sensor-equipped cars to track air quality on A block by block basis. Forbes. URL: https://www.forbes.com/sites/zarastone/2020/01/28/aclima-rolls-out-sensor-equipped-cars-to-track-air-quality-on-a-block-by-block-basis/?sh=7c4e70e055a1 (дата звернення: 13.02.2024).

20. What is narrowband iot: a comprehensive guide. Narrowband. URL: https://www.narrowband.com/what-is-narrowband-iot (дата звернення: 21.02.2024).

21. Особливості файлових систем. Kingston Technology Company. URL: https://www.kingston.com/ua/blog/personal-storage/understanding-file-systems (дата звернення: 03.05.2024).

22. Учасники проектів Вікімедіа. Express.js – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Express.js (дата звернення: 23.04.2024).

23. Учасники проектів Вікімедіа. MQTT – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/MQTT (дата звернення: 19.04.2024).

24. Що таке логи та як з ними працювати. Онлайн-курси від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/logging-in-concepts-requirements-levels/ (дата звернення: 25.04.2024).

26. Що таке Docker: простими словами про контейнеризацію: Стаття з блогу IT-школи Hillel. Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel. URL: https://blog.ithillel.ua/articles/shcho-take-docker-prostimi-slovami-pro-konteynerizatsiyu (дата звернення: 02.05.2024).

27. Comparison with HTTP and MQTT In Internet of Things (IoT). IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/8597401 (дата звернення: 22.04.2024).

28. Express.js | Guide books. Guide books. URL: https://dl.acm.org/doi/abs/10.5555/3035728 (дата звернення: 25.04.2024).

29. FREEhost.UA. URL: https://freehost.com.ua/ukr/faq/wiki/docker-novij-podhod-k-razrabotke-i-vnedreniju-programmnogo-obespechenija/ (дата звернення: 06.05.2024).

30. HTTP і HTTPs: що це таке і в чому різниця | HOSTiQ Wiki. HOSTiQ Wiki. URL: https://hostiq.ua/wiki/ukr/http-https/#what-is-http (дата звернення: 03.05.2024).

31. MQTT-S – A publish/subscribe protocol for Wireless Sensor Networks. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/4554519 (дата звернення: 20.04.2024).

32. Що таке протокол HTTPS та принципи його роботи. UKEY WAF - Надійний захист веб-сайту. URL: https://ukeywaf.com/shho-take-protokol-https-ta-prynczypy-jogo-roboty/ (дата звернення: 28.04.2024).

33. Що таке Docker і навіщо він?. QualityAssuranceGroup. URL: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення: 30.04.2024).

34. Як тестувати HTTP протокол?. Онлайн-курси від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/http-protocol-what-and-where-to-test/ (дата звернення: 28.04.2024).

35. About npm | npm docs. npm Docs. URL: https://docs.npmjs.com/about-npm (дата звернення: 28.03.2024).

36. Codex A. C. What is npm and how does it work? | Reintech media. Hire Developers for SaaS teams - Software Developers as a Service | Reintech. URL: https://reintech.io/blog/what-is-npm-and-how-does-it-work (дата звернення: 28.03.2024).

37. Contributors to Wikimedia projects. Raspberry pi - wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Raspberry_Pi (дата звернення: 10.02.2024).

38. Hashemi-Pour C., Bigelow S. J., Courtemanche M. What is docker and how does it work?. IT Operations. URL:

https://www.techtarget.com/searchitoperations/definition/Docker (дата звернення: 04.05.2024).

39. HTTP-запит (HTTP request) - QALight. QALight. URL: https://qalight.ua/baza-znaniy/http-zapyt-http-request/ (дата звернення: 29.04.2024).

40. Introduction of message queue telemetry transport protocol (MQTT) - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/introduction-of-message-queue-telemetry-transport-protocol-mqtt/ (дата звернення: 22.03.2024).

41. MQTT: відкритий мережевий протокол та його значення в IoT. Hardware libre. URL: https://www.hwlibre.com/uk/MQTT/ (дата звернення: 05.05.2024).

42. MQTT protocol | CQR. CQR. URL: https://cqr.company/ua/wiki/protocols/mqtt-protocol/ (дата звернення: 03.05.2024).

43. Mqtt. Real Time Automation, Inc. URL: https://www.rtautomation.com/technologies/mqtt/ (дата звернення: 14.03.2024).

44. Node.js – Node.js file stats. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/learn/manipulating-files/nodejs-file-stats (дата звернення: 14.04.2024).

45. Node.js – The V8 JavaScript Engine. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/learn/getting-started/the-v8-javascript-engine (дата звернення: 11.04.2024).

46. Running containers. Docker Documentation. URL: https://docs.docker.com/engine/reference/run/ (дата звернення: 03.05.2024).

47. Simulate user traffic to test your API performance | Postman Learning Center. Postman Learning Center. URL: https://learning.postman.com/docs/collections/performance-testing/testing-api-performance/ (дата звернення: 01.05.2024).

48. Start containers automatically. Docker Documentation. URL: https://docs.docker.com/config/containers/start-containers-automatically/ (дата звернення: 05.05.2024).

49. Ukrainian W. Уроки від W3Schools українською онлайн. W3Schools українською. Безплатні уроки онлайн для початківців, школярів та студентів. URL: https://w3schoolsua.github.io/tags/ref_httpmethods.html#gsc.tab=0 (дата звернення: 27.04.2024).

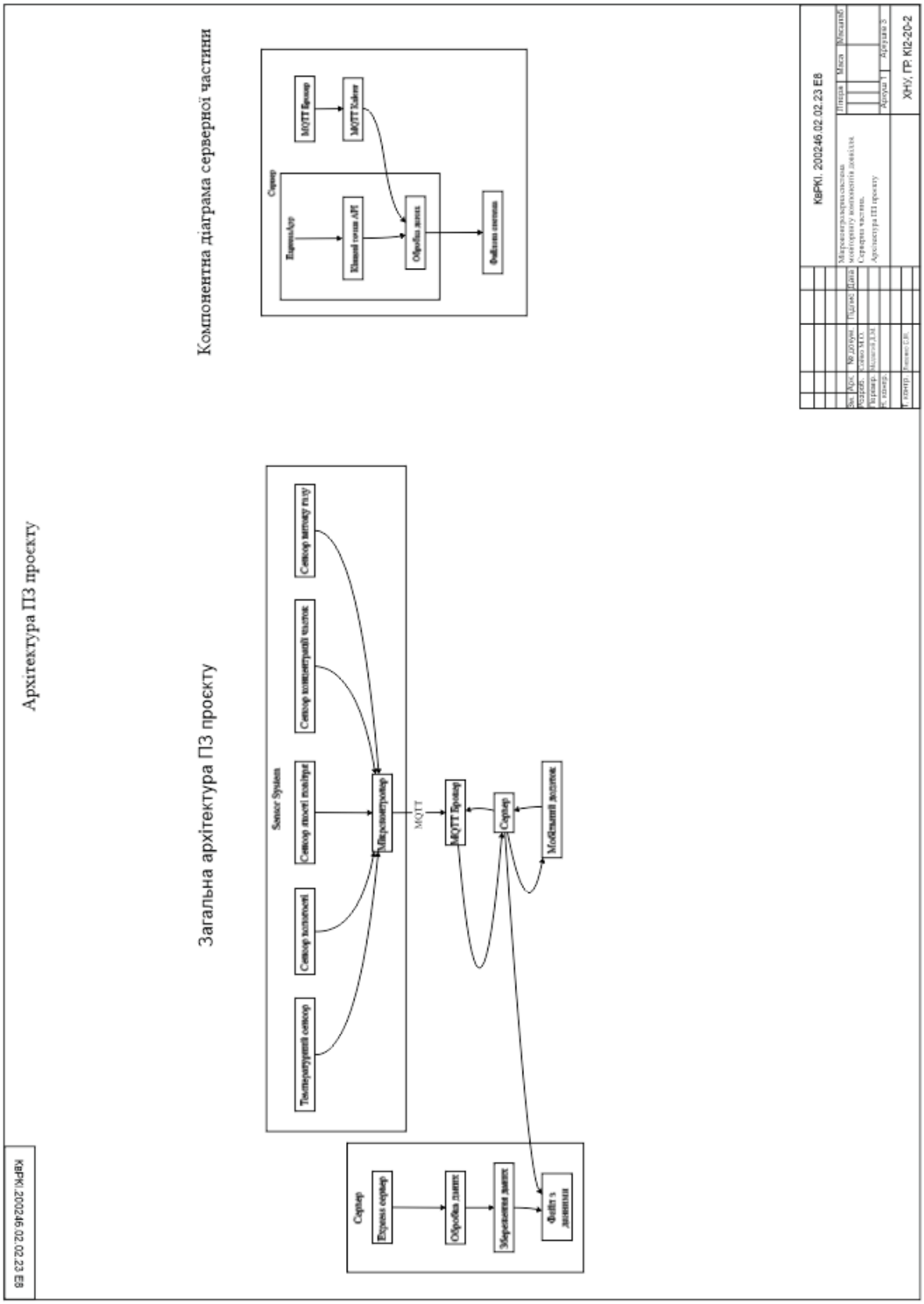
50. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/nodejs/ (дата звернення: 14.04.2024).

51. What is node.js and why you should use it. Kinsta®. URL: https://kinsta.com/knowledgebase/what-is-node-js/ (дата звернення: 03.04.2024).

Додаток А

(обов’язковий)

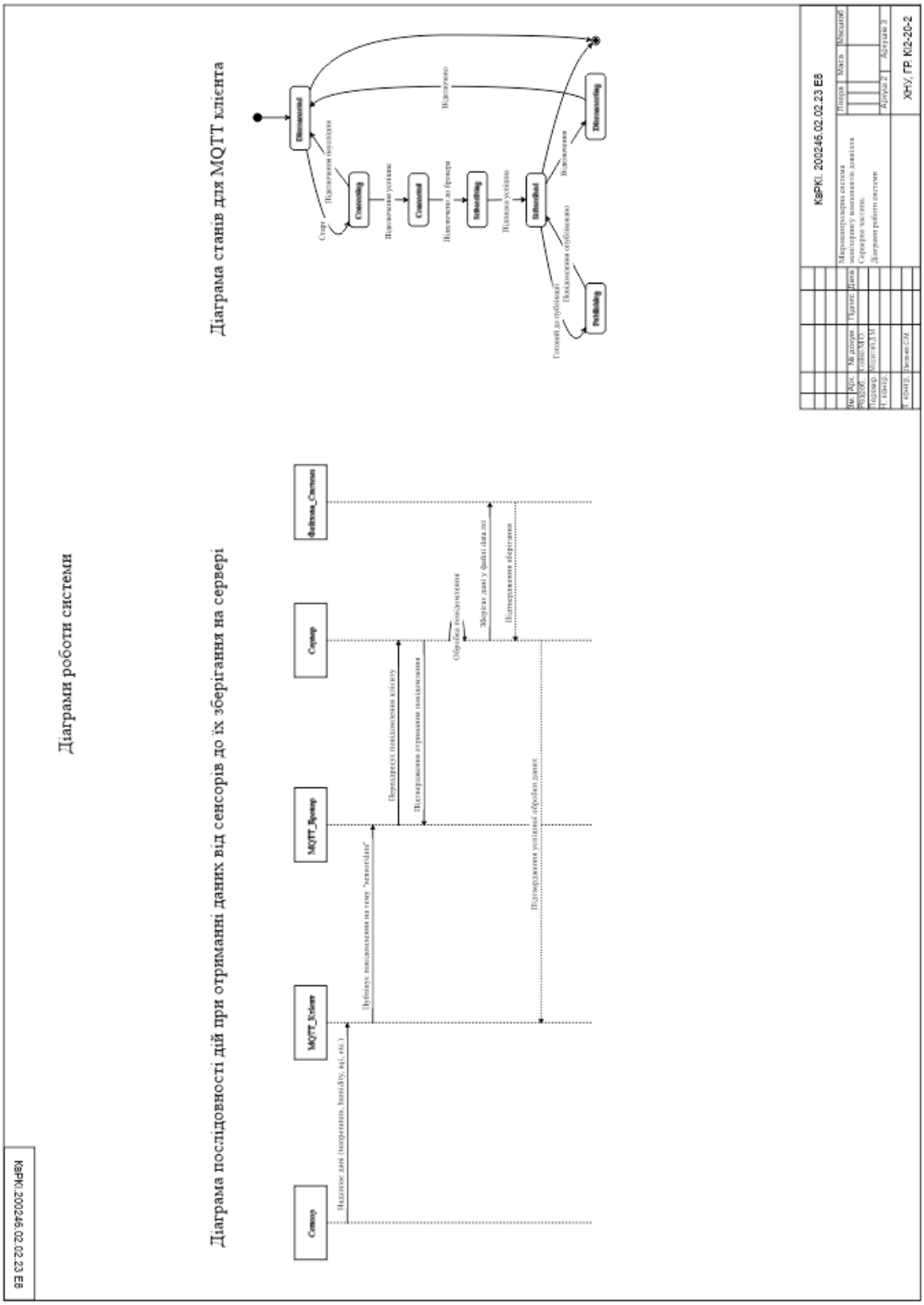
Копія креслення «Архітектура ПЗ проєкту»



Додаток Б

(обов’язковий)

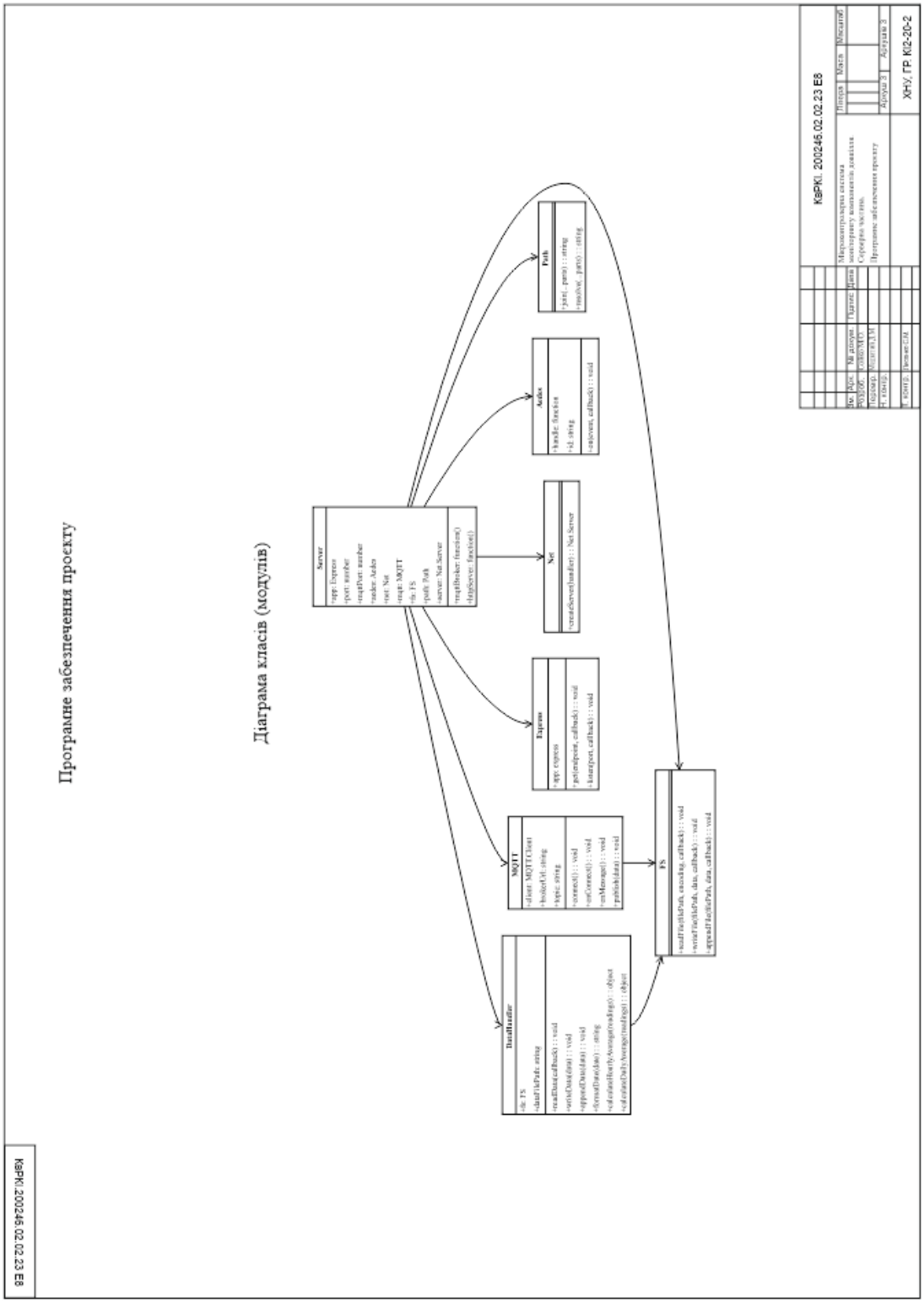
Копія креслення «Діаграми роботи системи»



Додаток В

(обов’язковий)

Копія креслення «Програмне забезпечення проєкту»



UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016369630

Дата перевірки:
17.06.2024 21:17:12 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
18.06.2024 08:12:40 EEST

ID користувача:
100005591

Назва документа: **Сойко_Мікроконтролерна система моніторингу компонентів довкілля. Серверна частина**

Кількість сторінок: **80** Кількість слів: **13440** Кількість символів: **107970** Розмір файлу: **6.30 MB** ID файлу: **1016176534**

5.56%
Схожість

Найбільша схожість: **0.88%** з Інтернет-джерелом (https://www.wikidata.uk-ua.nina.az/MQTT.html)

5.07% Джерела з Інтернету	187	Сторінка 82
2.02% Джерела з Бібліотеки	166	Сторінка 84

0.18% Цитат

Цитати	2	Сторінка 85
Посилання	1	Сторінка 85

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	15

Mon Jun 17 19:34:55 EEST 2024, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

ID: 131162 Назва: БКР Мікроконтролерна система моніторингу компонентів довкілля. Серверна частина Додано в БД: 2024-06-17 Автора: М. О. Сойко Керівники: Д. М. Медзатий Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	96086	823	2104 (2%)	21 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Сойко Максим Олександрович

Тема: Мікроконтролерна система моніторингу компонентів довкілля. Серверна частина

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3___ Кількість сторінок записки ___67___

1. Короткий зміст роботи та прийнятих рішень: Метою дипломної роботи є розробка та дослідження серверної частини мікроконтролерної системи моніторингу компонентів довкілля. У роботі проаналізовано теорію та сучасні технології, необхідні для реалізації серверної частини системи моніторингу. Проведено вибір технологій, розроблено архітектуру програмного забезпечення, створено та протестовано програмні модулі для збору, обробки та аналізу екологічних даних.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню та містить всі необхідні елементи для реалізації серверної частини мікроконтролерної системи моніторингу компонентів довкілля.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі кваліфікаційної роботи проведено дослідження предметної області та визначено вимоги до системи. У другому розділі обрано технології для реалізації серверної частини, розроблено архітектуру програмного забезпечення. У третьому розділі виконано програмну реалізацію серверної частини, проведено тестування та аналіз отриманих результатів.

4. Позитивні сторони роботи: висока практична цінність роботи, використання сучасних технологій та методів розробки програмного забезпечення, чітка структуризація матеріалу та логічна послідовність викладення.

5. Негативні сторони роботи: недостатня увага приділена оптимізації продуктивності та безпеки серверної частини.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно згідно діючих стандартів оформлення документації. Графічні матеріали мають високу якість та добре ілюструють результати дослідження.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні, результати дослідження мають практичну значущість та можуть бути використані для подальшої розробки систем моніторингу довкілля. Незважаючи на наявні недоліки, робота в цілому відповідає вимогам і показує хороші результати.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) В. М. Н., доцент кафедри АКТТеР Ларченко Л. О.

"19" 06 2024 р. _____ (підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Сойко Максима Олександровичв
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-20-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

22 квітня 2024 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Мікроконтролерна система моніторингу компонентів довкілля. Серверна частина

Автор: Сойко Максим Олександрович

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Медзатий Дмитро Миколайович, к.т.н., доц.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення використовуються частково, або мають відповідні посилання на оригінальні джерела, що підтверджує правомірне використання матеріалів;
3) виявлені збіги є загальновживаними виразами або фразами, що підтверджується наявністю посилань системи на численні джерела для кожного фрагменту;
4) деякі запозичення стосуються стандартних технічних описів та специфікацій, які є загальноприйнятими в галузі і не підлягають авторському праву;
5) усі виявлені системою модифікації тексту стосуються технічних термінів та назв технологій, що не вважається зміною тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 5.56% і адресується до 353 першоджерел, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи Д. М. Медзатий

Гарант ОП С.М. Лисенко

Завідувач кафедри КІІС Т. О. Говорущенко