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

# КВАЛІФІКАЦІЙНА РОБОТА

Система моніторингу стану ПК/Серверів у локальній мережі

Назва теми

Рівень вищої освіти *перший (бакалаврський)*

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

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

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

Шифр КвРКІ 301125.23.01.22 ПЗ

Виконав здобувач III курсу, група КІ2с-23-1 _____ Владислав ЯРИШ
Підпис Ініціали, прізвище

Керівник _____ Василь ЯЦКІВ
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«19» червня 2026 р. підпис Ініціали, прізвище
дата

Хмельницький 2026

# ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ПЕРШИЙ (БАКАЛАВРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

"10" 01 2026 р.

## ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Яришу Владиславу Романовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Система моніторингу стану ПК/Серверів у локальній мережі
Керівник проекту (роботи) Яцків Василь Васильович, д.т.н., проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

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

Система моніторингу стану ПК/Серверів у локальній мережі та постановка задачі щодо її удосконалення

Проєктування системи моніторингу стану ПК/Серверів у локальній мережі

Програмно-апаратна реалізація системи моніторингу стану ПК/Серверів у локальній мережі

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

Архітектура ПЗ проєкту

Архітектура ПЗ для кіберфізичної системи

Апаратне забезпечення проєкту

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |

7. Дата видачі завдання «_10_»_____01_____2026 р.

## КАЛЕНДАРНИЙ ПЛАН

| №з/п | Назва етапів (розділів) дипломного проекту (роботи) | Термін виконання етапів проекту (роботи) | Примітка |
|---|---|---|---|
| 1 | Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником | 10.01.2026 | виконано |
| 2 | Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження | 01.02.2026 | виконано |
| 3 | Робота над розділом 1 – дослідження предметної області та постановка задачі | 01.03.2026 | виконано |
| 4 | Робота над розділом 2 – вибір компонентів для проєктування системи моніторингу стану ПК/Серверів у локальній мережі | 01.04.2026 | виконано |
| 5 | Робота над розділом 3 – проєктування системи моніторингу стану ПК/Серверів у локальній мережі | 29.04.2026 | виконано |
| 6 | Оформлення пояснювальної записки згідно вимог | 25.05.2026 | виконано |
| 7 | Попередній захист ВКР | 26.05.2026 | виконано |
| 8 | Захист ВКР на засіданні ЕК | Червень 2026 року | |

Здобувач_____ Владислав ЯРИШ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ Василь ЯЦКІВ
Підпис Ім'я, ПРІЗВИЩЕ

| № рядка | формат | Позначення | Найменування | Кіл. листів | № екз | Примітка |
|---|---|---|---|---|---|---|
| | | | Текстові документи | | | |
| 1 | | КвРКІ 301125.23.01.22 ПЗ | Пояснювальна записка | 74 | | |
| | | | | | | |
| | | | Графічні матеріали | | | |
| 2 | | КвРКІ 301125.23.01.22 ПЗ | Архітектура ПЗ проєкту | 1 | | |
| | | | | | | |
| 3 | | КвРКІ 301125.23.01.22 ПЗ | Архітектура ПЗ для | 1 | | |
| | | | кіберфізичної системи | | | |
| | | | | | | |
| 4 | | КвРКІ 301125.23.01.22 ПЗ | Апаратне забезпечення | 1 | | |
| | | | проєкту | | | |

| Зм | Арк | № докум | Підпис | Дата | КвРКІ 301125.23.01.22 ВП | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Яриш | | | Відомість проекту | Літера | Аркуш 1 / Аркушів 1 |
| Перевір. | | Яцків | | | | У | |
| Н. контр. | | Лисенко | | | | ХНУ, КІ2с-23-1 | |
| Затв. | | Павлова | | | | | |

# АНОТАЦІЯ

Тема кваліфікаційної роботи: «Система моніторингу стану ПК/Серверів у локальній мережі».

Автор роботи: Владислав ЯРИШ.

Керівник роботи: Василь ЯЦКІВ.

Пояснювальна записка: 74 с., 11 рис., 3 дод., 52 джерел.

Графічна частина: 3 креслення.

АРХІТЕКТУРА, БАЗА ДАНИХ, ВЕБ-ПАНЕЛЬ, ЛОКАЛЬНА МЕРЕЖА, МОНІТОРИНГ, МЕТРИКИ, СЕРВЕР.

Кваліфікаційну роботу бакалавра присвячено розробці та реалізації системи моніторингу стану ПК і серверів у локальній мережі. Актуальність теми зумовлена потребою у своєчасному виявленні перевантажень, збоїв і нестабільних режимів роботи обчислювальної інфраструктури, що використовується в умовах навчальних лабораторій та інших локальних інформаційних середовищ.

Метою роботи є проєктування, реалізація та тестування програмної системи, призначеної для автоматизованого збору, передавання, обробки, збереження та візуалізації системних показників ПК і серверів у локальній мережі. Для досягнення поставленої мети проаналізовано предметну область, обґрунтовано архітектуру системи, вибрано технічні та програмні засоби реалізації, розроблено модель зберігання метрик, реалізовано агент збору системних показників, серверний модуль приймання й обробки даних та веб-інтерфейс для перегляду результатів моніторингу. У результаті створено програмне рішення, яке забезпечує централізований контроль стану обчислювальних вузлів, підтримує накопичення історії метрик і дає змогу своєчасно виявляти відхилення в роботі інфраструктури.

Підпис здобувача

30.05.2026

Дата

## ЗМІСТ

Вступ ..... 4
1 Аналіз предметної області та постановка задачі моніторингу стану пк/серверів у локальній мережі ..... 6
1.1 Особливості експлуатації ПК та серверів у локальних мережах навчальних лабораторій ..... 6
1.2 Системи моніторингу обчислювальної інфраструктури та їх призначення ..... 9
1.3 Аналіз існуючих підходів і рішень до моніторингу ПК та серверів ..... 12
1.4 Архітектурні принципи побудови систем збору, передачі та зберігання метрик ..... 16
1.5 Постановка задачі створення сервісу моніторингу стану ПК/серверів у локальній мережі ..... 19
1.6 Висновки до першого розділу ..... 22
2 Проєктування та реалізація системи моніторингу стану пк/серверів у локальній мережі ..... 24
2.1 Загальна архітектура системи моніторингу ..... 24
2.2 Обґрунтування вибору технічних та програмних засобів реалізації ..... 32
2.3 Розроблення структури даних та моделі зберігання метрик ..... 38
2.4 Організація збору, обробки та передачі даних ..... 44
2.5 Організація взаємодії з користувачем та візуалізації результатів ..... 50
2.6 Висновки до другого розділу ..... 56
3 Практична реалізація та експериментальна перевірка системи моніторингу ..... 59
3.1 Реалізація програмного агента для збору системних показників ..... 59
3.2 Реалізація серверного модуля приймання, обробки та зберігання метрик ..... 63
3.3 Реалізація веб-інтерфейсу системи моніторингу та засобів відображення результатів ..... 67

3.4 Тестування системи моніторингу в умовах експериментальної експлуатації ..... 71
3.5 Висновки до третього розділу ..... 75
Висновки ..... 77
Перелік джерел посилань ..... 79
Додаток А Копія креслення «Архітектура ПЗ проєкту» ..... 85
Додаток Б Копія креслення «Архітектура ПЗ для кіберфізичної системи» ..... 86
Додаток В Копія креслення «Апаратне забезпечення проєкту» ..... 87

# ВСТУП

У сучасних умовах функціонування інформаційного середовища стабільна робота ПК і серверів у локальній мережі набуває особливого значення, оскільки саме ці вузли забезпечують виконання навчальних, організаційних та прикладних завдань, пов’язаних із доступом до програмних сервісів, зберіганням даних, мережею взаємодії та обробкою інформації. У межах навчальних лабораторій така інфраструктура зазвичай використовується інтенсивно, причому режими її роботи постійно змінюються залежно від кількості користувачів, характеру занять, типу програмного забезпечення та обчислювального навантаження.

Практика експлуатації локальних мереж показує, що традиційний підхід до контролю стану комп’ютерної техніки часто зводиться до періодичних ручних перевірок або реагування вже після появи очевидних проблем. Такий спосіб не забезпечує повноцінного уявлення про реальний стан вузлів, оскільки зміни навантаження на процесор, оперативну пам’ять, дискову підсистему, мережу чи температурні режими мають динамічний характер і не завжди проявляються миттєво. У результаті проблема нерідко виявляється вже тоді, коли вона починає впливати на роботу користувачів, а пошук її причини потребує додаткового часу та ускладнюється відсутністю історії змін показників.

Особливої актуальності ця проблема набуває в навчальних лабораторіях, де комп’ютерне обладнання експлуатується в умовах нерівномірного навантаження, а одна й та сама техніка може використовуватися для різних типів завдань - від базової роботи з прикладними програмами до використання середовищ розробки, віртуалізації, інструментів аналізу даних і мережевих сервісів. У таких умовах стабільність функціонування окремого ПК або сервера вже не розглядається ізольовано, оскільки стан кожного вузла впливає на загальну керованість, передбачуваність і надійність усієї локальної мережі. Саме тому зростає потреба у створенні системи, яка забезпечує централізоване,

безперервне та наочне спостереження за ключовими параметрами роботи інфраструктури.

Актуальність теми кваліфікаційної роботи зумовлена потребою у створенні програмного рішення, яке дозволяє організувати централізований контроль стану ПК і серверів у локальній мережі навчальної лабораторії без надмірного ускладнення інфраструктури розгортання. Особлива увага приділяється саме таким показникам, як завантаження центрального процесора, використання оперативної пам’яті, стан дискової підсистеми, мережева активність і температурні параметри, оскільки ці характеристики найбільш повно відображають поточний стан обчислювального вузла та дають змогу виявляти причини зниження продуктивності або нестабільної роботи.

Метою кваліфікаційної роботи є розроблення системи моніторингу стану ПК і серверів у локальній мережі, яка забезпечує автоматизований збір системних показників, їх передавання, централізоване збереження, візуалізацію та своєчасне виявлення відхилень у роботі контрольованих вузлів.

Для досягнення поставленої мети в роботі визначено такі завдання: проаналізувати особливості експлуатації ПК і серверів у локальних мережах та існуючі підходи до моніторингу; обґрунтувати архітектуру майбутньої системи; вибрати технічні та програмні засоби реалізації; розробити модель даних і механізми зберігання метрик; реалізувати агент збору системних показників, серверний модуль та веб-інтерфейс; виконати експериментальну перевірку працездатності системи та оцінити результати її функціонування.

Об’єктом кваліфікаційної роботи є процес моніторингу стану ПК і серверів у локальній мережі.

Предметом кваліфікаційної роботи є методи, моделі та програмні засоби автоматизованого збору, передавання, зберігання та візуалізації системних показників обчислювальних вузлів.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ МОНІТОРИНГУ СТАНУ ПК/СЕРВЕРІВ У ЛОКАЛЬНІЙ МЕРЕЖІ

## 1.1 Особливості експлуатації ПК та серверів у локальних мережах навчальних лабораторій

Кваліфікаційна робота розглядає експлуатацію ПК та серверів у локальних мережах навчальних лабораторій як складний і багатофакторний процес, у межах якого поєднуються змінні режими навантаження, різнорідність апаратної бази, різні сценарії використання програмного забезпечення та обмежені можливості оперативного технічного супроводу [16]. У подібному середовищі обчислювальні ресурси не працюють у стабільному, передбачуваному режимі, характерному для вузькоспеціалізованих серверних систем, а перебувають у стані постійних коливань, що визначаються розкладом занять, складом груп, типом виконуваних робіт і навіть індивідуальними особливостями роботи користувачів [11]. Протягом одного навчального дня навантаження на процесор, оперативну пам’ять, дискову підсистему та мережеві інтерфейси може змінюватися від мінімального фону до пікових значень, коли одночасно запускаються середовища розробки, віртуальні машини, системи моделювання, інструменти аналізу даних або мережеві сервіси [12]. За відсутності систематичного контролю подібні коливання сприймаються лише фрагментарно, що ускладнює формування об’єктивного уявлення про реальний стан інфраструктури [9].

Особливістю навчальних лабораторій є також те, що одне й те саме обладнання використовується різними групами користувачів, кожна з яких працює зі своїм набором програмних засобів і власними сценаріями навантаження [18]. У межах одного тижня конфігурація робочого середовища може змінюватися декілька разів, що призводить до ситуацій, коли певні вузли мережі періодично переходять у режими підвищеного споживання ресурсів, а інші залишаються недовантаженими [17]. Подібна нерівномірність ускладнює

ручну оцінку ефективності використання апаратних засобів, оскільки суб'єктивні спостереження не дають змоги простежити повну картину у часовому розрізі [14]. Водночас саме часовий аспект є критично важливим для розуміння того, чи має місце короткочасний пік активності, чи формується стійка тенденція до перевантаження певного компонента системи [5].

Окрему роль у такій інфраструктурі відіграють сервери, які забезпечують спільні сервіси лабораторії, зокрема файлові ресурси, системи керування навчальними матеріалами, середовища віртуалізації або мережеві служби [21]. Ці вузли працюють у режимі, близькому до безперервного, і обслуговують запити з багатьох робочих станцій одночасно [13]. Навіть короткочасне зниження продуктивності сервера або нестабільність його роботи відразу відбивається на великій кількості користувачів, що сприймається як загальна «повільна робота мережі» або «зависання системи» [10]. Насправді ж першопричина може полягати у перевантаженні процесора, дефіциті оперативної пам'яті, уповільненні доступу до дискової підсистеми або у перегріві апаратних компонентів [3, 20]. Без централізованого спостереження за цими параметрами встановлення джерела проблеми часто перетворюється на тривалий і не завжди ефективний процес, що базується на припущеннях, а не на об'єктивних даних [1].

Додаткову складність створює різнорідність апаратної бази, оскільки в навчальних лабораторіях зазвичай експлуатуються ПК різних поколінь і конфігурацій, а також сервери з іншими характеристиками та призначенням [15]. У таких умовах одна й та сама програмна задача може по-різному впливати на різні вузли мережі: для одних вона не становить суттєвого навантаження, а для інших стає причиною відчутного зниження продуктивності або навіть нестабільної роботи [6]. Це ускладнює уніфікований підхід до оцінювання стану систем, оскільки просте порівняння «швидко» або «повільно» не відображає реальної картини [22]. Набагато інформативнішим є аналіз конкретних метрик у динаміці, який дозволяє зіставляти поточні показники з типовими значеннями

для конкретного вузла та робити висновки про відхилення від нормального режиму роботи [4].

Важливим чинником є й температурний режим обладнання, оскільки навчальні аудиторії не завжди мають ідеальні умови охолодження, а інтенсивне використання обчислювальних ресурсів призводить до підвищеного тепловиділення [7]. Перегрів процесора, відеокарти або елементів живлення може не лише знижувати продуктивність через механізми термального тротлінгу, але й прискорювати зношення апаратних компонентів, що в довгостроковій перспективі впливає на надійність усієї інфраструктури [19]. За відсутності регулярного контролю температур подібні проблеми зазвичай виявляються вже на етапі відмови або помітного погіршення роботи, тоді як систематичне спостереження за цими показниками дозволяє фіксувати небезпечні тенденції завчасно [8].

Не менш значущою є й мережева складова, оскільки у локальних мережах навчальних лабораторій активно використовуються спільні ресурси, відбувається передача великих обсягів даних, а також працюють сервіси, чутливі до затримок і пропускної здатності [23]. Перевантаження окремих сегментів мережі або мережевих інтерфейсів серверів і робочих станцій може проявлятися у вигляді загального зниження швидкодії прикладних програм, що знову ж таки часто сприймається як абстрактна «проблема з мережею» [24]. Без фіксації конкретних показників трафіку, швидкостей і кількості помилок передавання даних встановлення реального стану речей залишається приблизним і залежить від суб’єктивних оцінок [25].

Суттєвий вплив на процес експлуатації має й обмежений час, який може бути виділений на технічне обслуговування та діагностику, оскільки навчальний процес потребує стабільної доступності обладнання [2]. Тривалі простої для перевірки стану кожного вузла вручну є небажаними, а інколи й практично неможливими [26]. У таких умовах зростає цінність інструментів, що дозволяють отримувати зведену картину стану інфраструктури без необхідності фізичного

доступу до кожного ПК або сервера [27]. Накопичення історії показників, їх візуалізація та можливість швидко перейти від загального огляду до детального аналізу конкретного вузла створюють основу для більш упорядкованого й передбачуваного процесу експлуатації [28].

У підсумку особливості експлуатації ПК та серверів у локальних мережах навчальних лабораторій зводяться до поєднання змінних навантажень, різнорідності обладнання, високої залежності навчального процесу від стабільності роботи інфраструктури та обмежених можливостей оперативного ручного контролю [29]. Це формує об’єктивну потребу в автоматизованому, постійному та централізованому спостереженні за ключовими параметрами роботи систем, що дозволяє не лише фіксувати поточний стан, а й відстежувати тенденції, виявляти потенційні проблеми на ранніх етапах і підтримувати працездатність лабораторної інфраструктури на стабільному рівні без зайвих простоїв і втрат часу на пошук причин несправностей [30].

## 1.2 Системи моніторингу обчислювальної інфраструктури та їх призначення

Розглянули системи моніторингу обчислювальної інфраструктури як невід’ємний елемент сучасного підходу до експлуатації ПК та серверів у локальних мережах, де стабільність і передбачуваність роботи обладнання мають безпосередній вплив на якість виконання навчальних і прикладних завдань [16]. У контексті комп’ютерних систем під моніторингом розуміється безперервне або періодичне спостереження за ключовими параметрами стану апаратних і програмних компонентів, їх фіксація у часі та подальший аналіз з метою своєчасного виявлення відхилень від нормального режиму роботи [11]. На відміну від разових перевірок або епізодичних вимірювань, такий підхід формує цілісну картину функціонування інфраструктури, де кожен показник

розглядається не ізольовано, а у зв’язку з попередніми та поточними значеннями [5].

Призначення систем моніторингу полягає не лише у відображенні поточного стану ресурсів, а й у створенні основи для більш усвідомленого керування інфраструктурою [14]. Коли інформація про завантаження процесора, використання оперативної пам’яті, стан дискової підсистеми, активність мережевих інтерфейсів і температурні режими збирається систематично, з’являється можливість простежувати характерні для конкретного середовища шаблони поведінки [9]. У навчальних лабораторіях це особливо важливо, оскільки режими роботи обладнання змінюються залежно від розкладу занять, типу виконуваних робіт і кількості одночасних користувачів [18]. Накопичені дані дозволяють відрізняти нормальні пікові навантаження від аномальних ситуацій, що потребують втручання, і водночас уникати надмірної реакції на короткочасні коливання, які не мають суттєвого впливу на стабільність системи [10].

Ще однією важливою функцією систем моніторингу є підтримка процесу пошуку та локалізації причин збоїв [1]. У реальних умовах експлуатації проблеми з продуктивністю або стабільністю часто проявляються опосередковано, наприклад у вигляді повільної роботи прикладних програм, затримок у доступі до мережевих ресурсів або періодичних зависань робочих станцій [23]. Без об’єктивних вимірювань такі симптоми можуть трактуватися по-різному, що ускладнює прийняття обґрунтованих рішень [25]. Наявність історії метрик дозволяє співставляти момент виникнення проблеми зі змінами у навантаженні на конкретні компоненти системи, що значно звужує коло можливих причин і прискорює процес відновлення нормальної роботи [4].

Важливе місце у призначенні систем моніторингу займає і питання попередження критичних ситуацій [19]. Коли контроль обмежується лише фіксацією факту відмови, реагування завжди має запізнілий характер і супроводжується простоєм обладнання або втратою доступності сервісів [2]. У

випадку ж постійного спостереження за параметрами стає можливим виявлення тенденцій, які свідчать про поступове погіршення стану системи, наприклад про зростання середнього завантаження процесора, зменшення обсягу вільної оперативної пам’яті, заповнення дискового простору або підвищення робочих температур [7, 8]. Такі сигнали не обов’язково означають негайну відмову, проте вони створюють підґрунтя для своєчасних дій, спрямованих на запобігання більш серйозним наслідкам [30].

Системи моніторингу також виконують важливу інформаційну функцію, пов’язану з прозорістю стану інфраструктури [27]. У середовищі, де використовується велика кількість ПК і декілька серверних вузлів, ручне опитування або перевірка кожного елемента стає практично неможливою задачею [26]. Централізований збір і візуалізація показників дозволяють отримувати зведене уявлення про стан всієї мережі, швидко виявляти вузли з нетиповою поведінкою та переходити від загального огляду до детального аналізу конкретного компонента [28]. Це спрощує повсякденну експлуатацію та зменшує залежність від суб’єктивних оцінок, які часто формуються на основі окремих скарг або епізодичних спостережень [22].

Окремої уваги заслуговує роль систем моніторингу у плануванні розвитку інфраструктури [15]. Коли дані про використання ресурсів накопичуються протягом тривалого часу, вони починають відображати не лише поточний стан, а й загальні тенденції зміни навантаження [6]. Це дозволяє більш обґрунтовано підходити до питань модернізації обладнання, розширення обсягу пам’яті, заміни дискових накопичувачів або оптимізації мережевої архітектури [21]. Замість прийняття рішень на основі припущень або окремих інцидентів з’являється можливість спиратися на реальні показники, що характеризують роботу системи у звичних для неї режимах [12].

У навчальних лабораторіях додатковим аспектом призначення систем моніторингу стає їх освітній потенціал [29]. Наявність наочних графіків і статистики дозволяє демонструвати взаємозв’язок між програмними

навантаженнями та станом апаратних ресурсів, показувати вплив різних типів задач на продуктивність системи та формувати уявлення про практичні аспекти експлуатації комп'ютерних мереж [9].

У підсумку системи моніторингу обчислювальної інфраструктури виконують комплексну роль, поєднуючи функції спостереження, аналізу, попередження проблем і підтримки прийняття рішень [16]. Їх призначення виходить далеко за межі простого відображення поточних показників і охоплює завдання забезпечення стабільності, передбачуваності та ефективності експлуатації ПК і серверів у локальних мережах, що особливо актуально для умов навчальних лабораторій із динамічними та різнорідними режимами використання обладнання [30].

## 1.3 Аналіз існуючих підходів і рішень до моніторингу ПК та серверів

У межах кваліфікаційної роботи системи моніторингу ПК та серверів доцільно розглядати не як «одну програму», а як клас рішень, що відрізняються архітектурою збору даних, способом їх доставки на сервер, форматом зберігання метрик і тим, як саме реалізовано візуалізацію та сповіщення. На практиці найчастіше зустрічаються два підходи. Перший орієнтований на класичне інфраструктурне адміністрування з централізованою системою, де вузли додаються як хости, до них прив'язуються шаблони, правила збору і тригери. Другий ближчий до «метрик як часових рядів», коли агенти/експортери віддають показники у стандартизованому вигляді, а далі ці показники скрапляться або пушаться у сховище й відображаються на дашбордах. Обидва підходи в навчальній лабораторії мають сенс, але відрізняються складністю розгортання, гнучкістю та зручністю супроводу.

Одним із найбільш поширених рішень у категорії «класична система моніторингу» виступає Zabbix. Логіка Zabbix базується на тому, що вузли описуються як хости, а типовий набір перевірок і умов спрацювання задається

через шаблони. У документації підкреслюється, що Zabbix постачає набір попередньо налаштованих шаблонів “out-of-the-box”, які прискорюють розгортання моніторингу. Пороги та умови для сповіщень реалізуються через triggers, які вмикаються при досягненні заданих виразів/умов і можуть масштабуватися через шаблони на багато хостів. Практична сильна сторона Zabbix у лабораторії - швидке масштабування «за шаблоном» та наявність централізованого інтерфейсу з історією. Водночас Zabbix часто вимагає уважного налаштування, а при великій кількості різних типів вузлів доводиться витрачати час на адаптацію шаблонів під конкретні конфігурації, щоб тривоги не перетворювалися на шум.

Другий дуже популярний напрям - стек Prometheus + Grafana, інтерфейс Grafana зображено на рисунку 1.1.


Рисунок 1.1 - Інтерфейс Grafana [51]

У такій моделі збір системних показників на Linux-вузлах часто робиться через Node Exporter, який, за офіційним описом Prometheus, “exposes a wide variety of hardware- and kernel-related metrics”. У GitHub-описі проєкту Node Exporter також зазначено, що він призначений для експорту метрик апаратного рівня та ОС, а для Windows рекомендується окремий експортер. Візуалізація зазвичай реалізується через Grafana, яка інтегрується з Prometheus та іншими

джерелами метрик і призначена для побудови інтерактивних дашбордів. Практична перевага цього підходу - високий рівень стандартизації метрик і зручні графіки, а також типова модель “scrape” (коли сервер сам періодично зчитує метрики з вузлів). Для лабораторії це добре працює, якщо всі вузли досяжні по мережі та дозволено відкривати порти для експортерів. Складність з’являється там, де частина ПК працює в обмежених мережевих умовах або не завжди доступна для скрапінгу - у таких випадках частіше обирається модель “push” або буферизація на агенті.

Окремо варто виділити зв’язку Telegraf + InfluxDB + Grafana, яка дуже близька за логікою до теми кваліфікаційної роботи. Telegraf описується InfluxData як open-source server agent, призначений для збору метрик (і не тільки) зі “стеків, сенсорів і систем”. Інтерфейс InfluxDB зображено на рисунку 1.2.


Рисунок 1.2 - Інтерфейс InfluxDB [52]

Після збору дані природно лягають у сховище часових рядів (наприклад, InfluxDB), а візуалізація і дашборди часто будуються через Grafana, яка має повноцінний data source для InfluxDB та інструменти запитів/редагування. У лабораторних умовах цей підхід цінний тим, що “агент - TSDB - графіки”

складається у просту та логічну схему, де метрики саме і сприймаються як часові ряди, а не як “перевірки” у класичному розумінні. Обмеження полягає в тому, що Telegraf-стек краще відчувається як система метрик, а не як комплексний інструмент інвентаризації чи мережевого моніторингу “все в одному”, тому частину функцій (наприклад, реєстр вузлів, ролі, правила алертів, журнали подій) часто доводиться організовувати окремо.

Серед комерційних “all-in-one” рішень у навчальних установах та невеликих організаціях часто зустрічається Paessler PRTG Network Monitor. Позиціонування PRTG базується на сенсорній моделі: продукт має freeware-режим до 100 сенсорів, а далі використовується ліцензування за кількістю сенсорів, що прямо зазначається на офіційній сторінці продукту та в прайсингу. Сильна сторона PRTG - швидкий старт, авто-виявлення (залежно від сценарію) та великий набір “сенсорів” під різні протоколи. Але для лабораторії сенсорна модель може неочікувано “роздувати” вартість при детальному моніторингу кожного ПК/сервера, оскільки один вузол легко споживає десятки сенсорів, якщо збирати CPU, RAM, диск, мережу, сервіси тощо.

Огляд готових рішень показує, що універсальні системи на кшталт Zabbix/Checkmk/PRTG сильні централізованим керуванням, авто-виявленням і великою кількістю вбудованих сценаріїв, а метрик-орієнтовані стеки на кшталт Prometheus+Grafana або Telegraf+InfluxDB+Grafana краще “лягають” на задачу збору й зберігання часових рядів із подальшою побудовою графіків та правил алертів. Саме тому для теми кваліфікаційної роботи логічно обрано власний сервіс з агентом і централізованим бекендом, де канали доставки метрик (HTTP/MQTT), сховище часових рядів та веб-панель формуються під лабораторний сценарій без зайвого функціоналу, але з можливістю розширення під реальні потреби інфраструктури.

## 1.4 Архітектурні принципи побудови систем збору, передачі та зберігання метрик

Розглянуто архітектуру систем збору, передачі та зберігання метрик як послідовність узгоджених рішень, де кожен етап ланцюга – від отримання показників на вузлі до їх відображення на графіках і спрацювання алертів – має власні вимоги до точності, стабільності, масштабованості та безпеки [16]. Для навчальної лабораторії критично важливо, щоб система не залежала від ручних дій, працювала у фоні, не заважала основним задачам користувачів і водночас давала прозору картину стану всієї інфраструктури [27].

На рівні збору метрик базовим принципом виступає локальність і мінімальна інвазивність [11]. Дані про CPU, RAM, диск, мережу та температури природно виникають на конкретному вузлі, тому найнадійнішим способом є читання цих показників максимально близько до джерела – на рівні операційної системи та її системних інтерфейсів [25]. У такій моделі збирач метрик має бути легким, щоб не створювати додаткового навантаження, і передбачуваним за ресурсами, щоб сам процес моніторингу не впливав на результати вимірювання [26]. Звідси випливає необхідність контролю частоти зняття показників: занадто рідкі інтервали не дадуть побачити піки, а занадто часті можуть створювати непотрібний трафік і навантаження [23]. У лабораторному сценарії інтервал збору часто підбирається як компроміс між «детальністю» і «вартістю» збереження, тому архітектура має підтримувати конфігурованість частоти, а також уміти агрегувати або згладжувати дані на сервері [5].

Наступним принципом є уніфікація структури метрик [4]. Навіть простий набір показників на кшталт CPU чи пам’яті може мати різні способи подання залежно від ОС або конфігурації, тому архітектура повинна перетворити «сирі» дані в єдиний формат, де чітко задані одиниці вимірювання, часові мітки та ідентифікація джерела [14]. У практичних системах це означає, що агент або серверний приймач приводить значення до узгоджених типів: навантаження у

відсотках, пам’ять і дисковий простір у байтах (або у похідних величинах, але послідовно), мережеві лічильники та швидкості у стандартизованих одиницях, температури у градусах [12]. Часова мітка є не менш важливою, ніж саме значення, бо метрики без прив’язки до часу не формують часовий ряд, а без часового ряду неможливо якісно будувати графіки, робити порівняння та знаходити причинно-наслідкові зв’язки між подіями [41].

Принцип ідентифікації джерела метрик у системах моніторингу має особливе значення для локальної мережі [15]. Вузол повинен мати стабільний ідентифікатор, який не змінюється при перезавантаженнях і не залежить від тимчасових параметрів на кшталт DHCP-адреси [30]. З архітектурної точки зору це зводиться до концепції “host_id”, яка створюється під час реєстрації агента або задається адміністративно, а далі використовується як ключ для всіх записів у сховищі метрик [22]. Додаткові теги на кшталт аудиторії, ролі вузла (ПК/сервер), групи або призначення дозволяють будувати зрозумілу навігацію у панелі та спрощують фільтрацію, що важливо саме в лабораторії, де техніка часто групується за місцями й сценаріями використання [28].

Коли дані зібрані й нормалізовані, постає питання доставки, і тут архітектурно важливим стає вибір моделі передачі: “push” або “pull” [11]. У “pull”-підході сервер періодично опитує вузол і зчитує метрики самостійно [12]. Це зручно, коли всі вузли стабільно доступні, а політика безпеки дозволяє відкривати порти для експортерів [13]. У “push”-підході, навпаки, вузол сам надсилає пакети метрик на центральний сервер або брокер [1]. Для навчальної лабораторії push часто практичніший, оскільки робочі станції можуть бути вимкнені або перезавантажені, а також можуть існувати сегментації мережі, де серверу не завжди зручно «ходити» до кожного ПК [23]. Push також природно поєднується з буферизацією: якщо канал недоступний, агент накопичує дані локально і надсилає їх після відновлення зв’язку [24].

У межах передачі метрик ключовими принципами виступають надійність та прогнозованість [2]. Навіть у локальній мережі можливі короткі збої –

перезавантаження комутатора, зникнення Wi-Fi, падіння сервера або брокера [21]. Якщо система не закладає ці ситуації в архітектуру, метрики починають «губитися», а графіки матимуть провали, які неможливо пояснити [10]. Тому практична архітектура передбачає механізми повторної відправки, контроль доставки та локальну чергу [3]. Черга може бути реалізована як файлова буферизація або як компактне локальне сховище, де пакети метрик зберігаються з часовою міткою та відміткою про доставку [20]. Важливо, щоб така буферизація мала обмеження, інакше при довготривалому збої вузол може заповнити диск [25]. Через це в архітектурі часто застосовується кільцевий буфер або політика відкидання найстаріших записів [6].

Метрики збираються не заради самого факту збереження, а щоб їх можна було інтерпретувати [28]. Тому схема тегів, назви показників і правила агрегації повинні бути підібрані так, щоб запити для графіків і перевірок формувалися просто й прозоро [9]. Якщо метрики записані без тегів або з хаотичними назвами, дашборди стають складними, а алерти – ненадійними [10]. У навчальній лабораторії бажано, щоб інтерфейс дозволяв швидко знайти потрібний вузол, вибрати період і побачити основні ресурси на окремих графіках, а також чітко відстежити момент, коли система перейшла в проблемний стан [8].

З погляду експлуатації архітектурно значущим є принцип відтворюваного розгортання [31]. Сервіс моніторингу в лабораторії має бути не “налаштований вручну на одному ПК”, а розгорнутий так, щоб при необхідності його можна було швидко відновити або перенести [31]. Це підводить до контейнеризації компонентів і чіткого опису їх взаємодії: база часових рядів, брокер повідомлень, бекенд API, фронтенд веб-панелі та супутні сервіси [31]. У такій схемі кожен компонент має власні налаштування, порти, томи для даних та правила запуску, а взаємозв’язки задаються конфігурацією [31]. Це зменшує ризик помилок і робить систему прозорішою для демонстрації та супроводу [27].

Не можна оминути й принцип безпеки, який у локальних мережах часто недооцінюється [38]. Навіть у межах лабораторії можливі ситуації, коли вузол

надсилає некоректні або підроблені дані, а також коли сторонній клієнт намагається під'єднатися до сервера моніторингу [39]. Тому архітектура передбачає автентифікацію вузлів: агент використовує ключ або облікові дані для доступу до API чи брокера, а сервер перевіряє ці дані перед прийомом і записом метрик [34]. Окрім цього, бажаною є сегментація доступу на рівні тем MQTT або ендпоінтів API, щоб вузол мав змогу публікувати лише власні метрики, а не «перезаписувати» дані інших хостів [1]. Для веб-панелі вводиться розмежування ролей доступу, бо навіть у навчальному середовищі не завжди потрібно, щоб кожен користувач мав адміністративні функції [37].

Архітектурні принципи побудови систем збору, передачі та зберігання метрик зводяться до локального й легкого збору даних на вузлах, уніфікованого формату метрик із коректними часовими мітками та ідентифікацією, надійної доставки через стандартизовані канали з буферизацією при збоях, використання спеціалізованого сховища часових рядів із політиками зберігання та агрегації, а також забезпечення прозорої інтеграції цих даних у графіки й алерти [16, 28]. Така архітектура формує фундамент для подальшого проєктування конкретної реалізації сервісу моніторингу в наступних розділах кваліфікаційної роботи, де принципи набувають вигляду реальних компонентів, модулів і сценаріїв взаємодії в межах локальної мережі навчальної лабораторії [31].

## 1.5 Постановка задачі створення сервісу моніторингу стану ПК/серверів у локальній мережі

Кваліфікаційна робота виходить із того, що експлуатація обчислювальної інфраструктури навчальної лабораторії відбувається в умовах змінного навантаження, різнорідності апаратних платформ і обмежених можливостей для постійного ручного контролю. У попередніх підрозділах показано, що відсутність централізованого спостереження за станом ресурсів призводить до запізнілого виявлення проблем, складності встановлення причин нестабільної

роботи та неефективного використання наявних потужностей. Унаслідок цього сформовано потребу у створенні спеціалізованого сервісу, який забезпечує безперервний збір, передачу, збереження та аналіз метрик з ПК і серверів у межах локальної мережі з подальшою візуалізацією та автоматизованим виявленням відхилень.

Задача створення такого сервісу формулюється як побудова цілісної програмної системи, що працює у фоновому режимі, не заважає основним процесам на робочих станціях і водночас надає повну та узгоджену картину стану інфраструктури. У центрі цієї задачі знаходиться агент збору метрик, який встановлюється на кожному ПК або сервері та виконує регулярне зчитування ключових показників. До переліку базових параметрів віднесено завантаження процесора, використання оперативної пам’яті, стан дискової підсистеми, активність мережевих інтерфейсів і температурні режими основних компонентів. Агент має працювати стабільно протягом тривалого часу, підтримувати конфігурований інтервал опитування та формувати дані в уніфікованому форматі з коректними часовими мітками та ідентифікатором вузла.

Наступною складовою задачі визначено організацію надійної передачі зібраних метрик у межах локальної мережі. Канал обміну даними має забезпечувати доставку показників у реальному або близькому до реального часі, бути достатньо простим для розгортання та не вимагати складної мережевої інфраструктури. Водночас передбачено, що можливі короткочасні збої зв’язку або недоступність серверної частини, тому сервіс повинен підтримувати буферизацію даних на стороні агента з подальшою передачею накопичених пакетів після відновлення з’єднання. Це дозволяє зберегти цілісність часових рядів і уникнути прогалин у статистиці, що особливо важливо для подальшого аналізу тенденцій.

Окремим аспектом постановки задачі є вибір і організація сховища даних. Метрики розглядаються як часові ряди, для яких характерні часті записи та

переважно агреговані запити за певні проміжки часу. Відповідно, у межах сервісу необхідно забезпечити збереження великого обсягу вимірювань із можливістю швидкого отримання середніх, максимальних або мінімальних значень за обраний період. Паралельно з цим має існувати сховище для конфігураційної та довідникової інформації, зокрема опису вузлів, групування обладнання, налаштувань правил сповіщень і параметрів доступу. Такий поділ дозволяє оптимізувати роботу з даними різної природи та спростити подальший супровід системи.

Ще одним обов'язковим елементом постановки задачі є реалізація механізму автоматизованих сповіщень про відхилення від нормальних режимів роботи. Для цього необхідно визначити набір контрольованих умов, наприклад перевищення заданих порогів завантаження процесора або пам'яті, критичні температурні значення, заповнення дискового простору чи відсутність даних від вузла протягом певного часу. Сервіс має не лише фіксувати такі події, а й реєструвати їх у журналі та передавати повідомлення визначеними каналами, щоб інформація про проблему з'являлася без затримки та не губилася серед інших подій. При цьому логіка спрацювання повинна враховувати тривалість і стійкість відхилення, щоб уникати хибних сповіщень через короткочасні піки навантаження.

У межах постановки задачі також визначено вимоги до надійності та відтворюваності розгортання сервісу. Система повинна складатися з чітко розмежованих компонентів, кожен з яких виконує власну функцію, а їх взаємодія описується конфігурацією. Це дозволяє швидко розгорнути сервіс у лабораторному середовищі, відновити його роботу у разі збоїв та продемонструвати працездатність на іншому обладнанні без тривалого ручного налаштування. Важливою умовою є й мінімальні вимоги до обслуговування після впровадження, оскільки навчальна інфраструктура зазвичай не передбачає виділення значних ресурсів на постійний супровід допоміжних сервісів.

Необхідною складовою задачі визначено забезпечення базового рівня безпеки обміну даними. Кожен агент повинен мати засіб автентифікації при передаванні метрик, а серверна частина - перевіряти ці дані перед прийомом і збереженням показників. Це унеможливлює випадкове або навмисне надсилання некоректної інформації сторонніми клієнтами та зберігає цілісність зібраної статистики. Для веб-панелі передбачається розмежування доступу, щоб функції перегляду та адміністрування не змішувалися в межах одного облікового запису.

У підсумку задача створення сервісу моніторингу стану ПК та серверів у локальній мережі формулюється як розроблення програмно-орієнтованої системи, що забезпечує автоматизований збір ключових метрик, їх надійну передачу, ефективне збереження у вигляді часових рядів, наочну візуалізацію та своєчасне сповіщення про відхилення. Така постановка задачі визначає вимоги до архітектури, функціональних можливостей і експлуатаційних характеристик сервісу та створює основу для подальшого переходу до опису конкретних технічних рішень і реалізації у наступних розділах кваліфікаційної роботи.

## 1.6 Висновки до першого розділу

У першому розділі кваліфікаційної роботи проаналізовано предметну область, пов’язану з моніторингом стану ПК і серверів у локальній мережі, та сформовано теоретичну основу для подальшого проєктування системи. Розглянуто особливості експлуатації обчислювальної техніки в умовах навчальних лабораторій, для яких характерними є нерівномірне навантаження, різнорідність апаратних конфігурацій, змінні сценарії використання програмного забезпечення та обмежені можливості постійного ручного контролю. Унаслідок цього встановлено, що традиційні підходи до спостереження за станом інфраструктури не забезпечують достатньої оперативності та не дозволяють своєчасно виявляти відхилення в роботі вузлів.

У процесі аналізу систем моніторингу обчислювальної інфраструктури показано, що їх призначення не обмежується простим відображенням поточних показників. Такі системи забезпечують накопичення історії метрик, виявлення тенденцій, локалізацію причин нестабільної роботи, підтримку прийняття технічних рішень та попередження критичних ситуацій.

Під час аналізу існуючих підходів і готових рішень розглянуто сучасні платформи та стеки, що застосовуються для моніторингу ПК і серверів. Показано, що класичні системи адміністрування на кшталт Zabbix, Checkmk, PRTG або Nagios мають широкі функціональні можливості, але часто виявляються надлишковими або менш гнучкими для локального лабораторного сценарію. Водночас метрик-орієнтовані підходи, побудовані навколо часових рядів, краще відповідають задачі спостереження за системними ресурсами, оскільки спрощують збереження історії показників, побудову графіків і реалізацію алертів. Саме це створило підґрунтя для вибору власного програмного рішення з агентним збором даних, серверною обробкою, зберіганням метрик і веб-візуалізацією.

У результаті проведеного аналізу сформульовано постановку задачі створення сервісу моніторингу стану ПК і серверів у локальній мережі. Визначено, що розроблювана система повинна забезпечувати автоматизований збір системних показників, їх передавання на сервер, зберігання у вигляді часових рядів, візуалізацію у веб-інтерфейсі та підтримку сповіщень про відхилення від нормальних режимів роботи. Окремо встановлено вимоги до надійності, простоти розгортання, зручності експлуатації та базового рівня захисту даних.

У підсумку перший розділ сформував теоретичну та методичну основу кваліфікаційної роботи. Виконаний аналіз дозволив обґрунтувати актуальність теми, показати доцільність створення спеціалізованої системи моніторингу для умов локальної мережі навчальної лабораторії та визначити основні вимоги до її побудови.

# 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ СТАНУ ПК/СЕРВЕРІВ У ЛОКАЛЬНІЙ МЕРЕЖІ

## 2.1 Загальна архітектура системи моніторингу

У кваліфікаційній роботі загальну архітектуру системи моніторингу стану ПК і серверів у локальній мережі розглянуто як багаторівневу функціональну структуру, у межах якої всі програмні та мережеві компоненти узгоджено працюють для отримання, передавання, накопичення, аналізу та відображення інформації про стан обчислювальної інфраструктури. Побудова такої архітектури зумовлена тим, що в умовах навчальної лабораторії недостатньо просто періодично переглядати окремі показники на локальних машинах. Необхідною є система, яка безперервно працює у фоновому режимі, не заважає основному використанню техніки, забезпечує централізований контроль і при цьому залишається достатньо простою для впровадження, супроводу та подальшого розширення. Саме через це архітектура сформована не як набір розрізнених програмних модулів, а як єдиний сервіс, де кожна складова виконує власну чітко визначену роль і водночас підтримує роботу всього комплексу.

Під час формування архітектури враховано, що локальна мережа навчальної лабораторії має кілька характерних особливостей. По-перше, у ній одночасно функціонують різні типи вузлів: звичайні робочі станції, які активно використовуються під час занять, і серверні машини, що працюють майже безперервно та забезпечують надання спільних сервісів. По-друге, навантаження на ці вузли не є сталим. Протягом дня воно може змінюватися дуже різко, коли в один момент система працює майже в холостому режимі, а вже через короткий час на неї одночасно впливають запуск середовищ програмування, робота з базами даних, мережевий обмін, віртуалізація або виконання обчислювально інтенсивних задач. По-третє, самі вузли не завжди перебувають в однаковому стані: частина з них може бути тимчасово вимкнена, перезавантажена або тимчасово недоступна через зміни конфігурації мережі чи режиму роботи

аудиторії. Усе це вимагає такої архітектури, яка не ґрунтується на припущенні про постійну доступність усіх вузлів і не вимагає ручного втручання під час звичайного функціонування.

Загальна логіка побудови системи сформована за принципом розподіленої моделі, де на кожному контрольованому вузлі працює локальний агент збору метрик, а в центрі розміщується серверна частина, що виконує приймання, обробку, збереження та візуалізацію даних. Такий підхід виявився найбільш доцільним саме для теми моніторингу стану ПК і серверів у локальній мережі, оскільки дозволяє одразу вирішити кілька практичних задач. З одного боку, збір показників здійснюється максимально близько до джерела їх виникнення, що підвищує точність та актуальність інформації. З іншого боку, централізація подальших етапів у серверній частині дозволяє сформувати єдину точку спостереження за всією мережею, спростити порівняння вузлів між собою, організувати історичне зберігання даних і реалізувати єдині механізми аналітики, візуалізації та сповіщень.

На нижньому рівні архітектури розташовано агентний шар. Саме він є точкою безпосереднього контакту системи моніторингу з фізичним обладнанням та операційною системою. Агент встановлюється на кожен ПК або сервер, який необхідно контролювати, і виконує періодичне зчитування базових метрик стану вузла. До цих метрик віднесено показники завантаження процесора, використання оперативної пам’яті, заповнення і активність дискових накопичувачів, параметри роботи мережевих інтерфейсів, а також температурні значення, якщо операційна система та апаратна конфігурація надають доступ до таких сенсорних даних. Агент побудовано так, щоб він працював у фоновому режимі, не потребував взаємодії з користувачем і не створював відчутного навантаження на контрольований вузол. Це важливо не лише з точки зору технічної акуратності реалізації, а й з практичної позиції: система моніторингу не повинна сама ставати фактором зниження продуктивності того обладнання, стан якого вона контролює.

Функціонування агента ґрунтується на циклічному принципі. Через заданий інтервал часу він виконує опитування системних джерел даних, отримує набір поточних показників, формує структурований пакет метрик і готує його до передавання. На цьому етапі важливо, що агент не просто «знімає цифри», а виконує первинну нормалізацію. Значення переводяться до уніфікованого формату, доповнюються часовою міткою та ідентифікатором вузла, а також можуть містити додаткові ознаки, наприклад назву аудиторії, роль машини, тип операційної системи або логічну групу, до якої належить вузол. Саме такий підхід дозволяє уникнути хаотичного потоку даних, який було б складно інтерпретувати на сервері. Уже на рівні агента формується основа для подальшого впорядкування та аналізу.

Побудова агентного шару передбачає, що один і той самий принцип роботи має застосовуватись до різних типів вузлів. Це означає, що архітектура повинна підтримувати уніфіковану схему збору метрик навіть за умови відмінностей між робочими станціями та серверними машинами. На практиці це не означає, що кожен вузол обов’язково надає повністю однаковий набір показників. Наприклад, деякі ПК можуть мати доступ до температурних сенсорів процесора, а деякі - ні. Проте архітектура побудована так, щоб ця відмінність не порушувала цілісність системи: обов’язкові метрики формують ядро інформаційного пакета, а додаткові параметри можуть підключатися залежно від доступності. Це робить рішення достатньо гнучким і водночас не руйнує загальну логіку подання даних.

Наступним рівнем архітектури виступає транспортно-комунікаційний шар, який забезпечує передавання метрик від агентів до центрального сервера. У межах кваліфікаційної роботи цей рівень розглянуто як один із найбільш важливих, оскільки саме через нього проходить увесь потік інформації від контрольованих вузлів до системи аналізу. Якщо цей рівень буде побудований невдало, навіть якісний збір метрик і потужне сховище не дадуть практичної користі. Архітектура передбачає використання двох можливих способів передавання даних - через HTTP та через MQTT. Це рішення не є випадковим.

Воно продиктоване бажанням забезпечити універсальність системи та можливість адаптації під різні режими роботи мережі. HTTP зручний як зрозумілий і прозорий спосіб взаємодії з серверною частиною через API, тоді як MQTT є ефективним механізмом для обміну невеликими повідомленнями в режимі, близькому до реального часу, із підтримкою моделі публікації та підписки.

У разі використання HTTP агент формує запит до серверного API та передає в ньому підготовлений пакет метрик. Така схема добре підходить для ситуацій, коли потрібна проста та зрозуміла модель взаємодії між вузлом і сервером. Вона легко відлагоджується, добре документується і логічно вписується в архітектуру сучасних веб-сервісів. У разі застосування MQTT архітектура набуває ще більш вираженого розподіленого характеру: агент стає публікатором, який відправляє повідомлення до відповідної теми, а серверна частина або окремий модуль обробки підписується на ці теми та приймає потік даних. Такий підхід вигідний тим, що дає змогу гнучко масштабувати систему, розділяти сервіси за ролями та зменшувати пряме навантаження на API-шар. У межах локальної мережі лабораторії обидві моделі є практично застосовними, тому архітектура не зводиться жорстко лише до одного транспортного механізму, а допускає їх використання залежно від конкретного сценарію.

Окреме місце в архітектурі посідає механізм буферизації, який виступає захистом від короткочасних збоїв зв’язку або тимчасової недоступності серверної частини. У реальному середовищі неможливо повністю виключити ситуації, коли агент не може відразу передати зібрані дані. Це може бути пов’язано із перезавантаженням сервера, короткою втратою мережевого з’єднання або обслуговуванням мережевого обладнання. Саме тому архітектура не ґрунтується на припущенні, що доставка буде ідеальною в кожен момент часу. Навпаки, у ній закладено механізм локального накопичення метрик на вузлі з подальшою відправкою після відновлення зв’язку. Це рішення суттєво підвищує надійність усієї системи, оскільки дозволяє не втрачати інформацію і зберігати

більш повну картину змін стану вузла навіть у разі тимчасових комунікаційних проблем.

Після транспортного рівня архітектура переходить до серверної частини, яка є центральним координаційним вузлом усієї системи моніторингу. Серверний рівень виконує одразу кілька взаємопов'язаних функцій. Насамперед він відповідає за приймання пакетів метрик від агентів або від брокера повідомлень. Далі він виконує перевірку цілісності та коректності отриманих даних, приводить їх до внутрішньої моделі подання, розподіляє по відповідних сховищах і готує до використання іншими підсистемами. У цьому сенсі серверна частина є не просто точкою приймання інформації, а повноцінним логічним центром системи, де метрики переходять зі стану «сирих» пакетів до стану структурованих даних, придатних для візуалізації, аналітики та формування алертів. На рисунку 2.1 подано загальну архітектуру системи моніторингу стану ПК/серверів у локальній мережі.

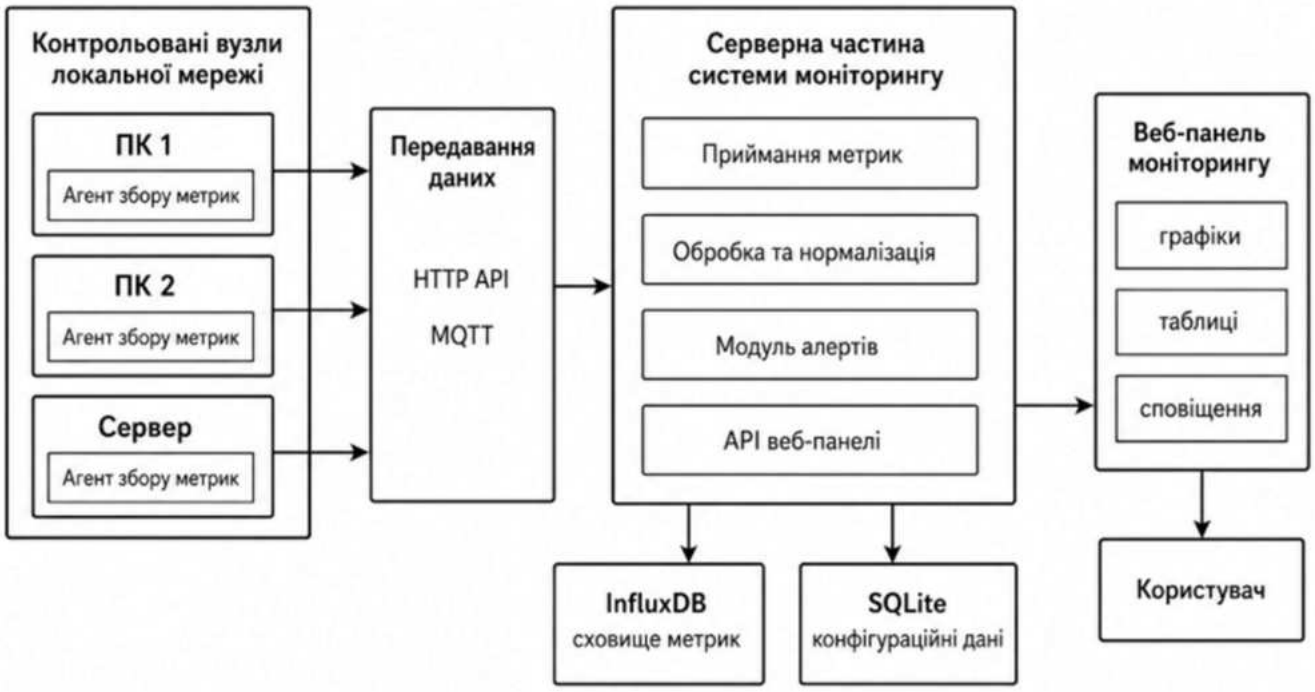



Рисунок 2.1 – Загальна архітектура системи моніторингу стану ПК/серверів у локальній мережі

Важливою архітектурною особливістю є розділення зберігання часових рядів і службових даних. Це рішення продиктоване різною природою інформації, з якою працює система. Метрики надходять часто, у великому обсязі, з чіткою прив’язкою до часу, і надалі найчастіше використовуються в агрегаційних або історичних запитах. Для такого типу даних найбільш доцільним є використання спеціалізованої бази часових рядів. Водночас система має працювати не лише з метриками, а й з інформацією про самі вузли, їхні назви, ролі, налаштування, правила алертів, користувачів і параметри доступу. Такі дані є більш статичними, структурованими та краще вписуються у звичайну реляційну модель. Саме тому архітектура передбачає використання різних підходів до зберігання: окреме сховище для часових рядів і окреме для конфігураційної та довідкової інформації.

Шар збереження часових рядів є одним із ключових, оскільки саме в ньому накопичується історія роботи всієї інфраструктури. У цій частині архітектури важливо не лише записати поточні значення, а й забезпечити можливість подальшого швидкого пошуку, фільтрації та агрегації. Наприклад, для системи моніторингу важливо не тільки знати поточне завантаження процесора на певному ПК, а й бачити, як воно змінювалося протягом останньої години, доби або тижня. Так само важливо порівняти поведінку двох вузлів, знайти періоди аномального росту навантаження або виявити момент, з якого почалося поступове погіршення температурного режиму. Саме з цієї причини в архітектурі закладено роботу з часовими рядами як із центральним типом даних, а не як із другорядним елементом.

Разом із цим у межах архітектури продумано політики зберігання, які не дозволяють системі безконтрольно нарощувати обсяг сховища. Якщо метрики збираються з невеликим інтервалом часу та одночасно надходять з багатьох вузлів, обсяг накопичених даних починає зростати дуже швидко. Через це архітектура передбачає не лише запис «сирих» значень, а й подальшу агрегацію

за певними часовими інтервалами. Це означає, що детальні дані можуть зберігатися лише обмежений час, а далі перетворюватися на середні, мінімальні чи максимальні значення за хвилину, годину або інший інтервал. Такий підхід дозволяє зберегти практичну цінність статистики і водночас не перетворити сервер моніторингу на систему, що обслуговує лише власне сховище.

Над рівнем зберігання в архітектурі розміщується аналітичний шар, який виконує роль проміжної ланки між даними та користувацьким інтерфейсом. Саме тут реалізується обчислення похідних показників, вибірка потрібних часових відрізків, формування зведень та підготовка результатів до візуалізації. На цьому ж рівні можуть виконуватися перевірки на наявність аномалій, перевищення порогів або відсутність надходження даних від окремого вузла. У контексті архітектури цей шар є дуже важливим, оскільки він переводить систему від простого «складування метрик» до їх реального практичного використання. Без нього навіть наявність великої кількості даних не дала б потрібного ефекту, оскільки користувачеві довелося б самостійно інтерпретувати сирі числа.

Ще одним важливим принципом, закладеним в архітектуру, є відтворюваність розгортання. Для сучасних програмних систем цього типу надзвичайно важливо, щоб серверна частина могла бути швидко розгорнута, відновлена або перенесена на інше обладнання без потреби в довгому ручному налаштуванні. З цієї причини архітектура передбачає контейнеризований підхід до розміщення основних компонентів: серверного API, сховища, брокера повідомлень і веб-інтерфейсу. Це спрощує супровід, знижує ризик конфігураційних помилок і дозволяє зберегти однаковість середовища під час тестування, демонстрації та експлуатації. У практичному сенсі це означає, що система моніторингу може бути розгорнута значно швидше і з меншими витратами часу на налаштування.

Питання безпеки також інтегровано в загальну архітектуру не як додатковий післядум, а як один із базових рівнів проєктування. Навіть у межах

локальної мережі не можна виходити з припущення, що будь-який вузол автоматично є довіреним. Саме тому архітектура передбачає автентифікацію агентів, перевірку їхніх даних на сервері, обмеження доступу до каналів передавання метрик і розмежування прав користувачів у веб-інтерфейсі. Це означає, що агент не просто надсилає довільні пакети, а працює в межах контрольованої логіки доступу, а серверна частина не сприймає будь-яке повідомлення як достовірне без перевірки. Такий підхід значно підвищує надійність системи не лише з технічної, а й з організаційної точки зору.

У цілісному вигляді архітектура системи моніторингу може бути представлена як послідовний ланцюг взаємодії рівнів. На першому етапі агент на вузлі зчитує метрики з операційної системи та локальних сенсорів. Далі ці метрики нормалізуються й передаються через транспортний шар до центральної серверної частини. Потім сервер приймає пакети, перевіряє їх, розподіляє по відповідних сховищах і передає в аналітичні модулі. На основі накопичених даних формуються графіки, таблиці, зведення та події сповіщень, які відображаються у веб-панелі. У результаті формується завершений цикл, у якому кожен рівень підтримує наступний, а всі разом вони забезпечують безперервний контроль стану інфраструктури.

У підсумку загальну архітектуру системи моніторингу в межах кваліфікаційної роботи сформовано як багаторівневу, модульну та розподілену структуру, що включає агентний рівень збору показників, транспортний рівень передавання даних, серверний рівень приймання й обробки, окремі сховища для часових рядів і службових даних, аналітичний модуль, підсистему алертів та веб-інтерфейс користувача. Саме така архітектура найбільш повно відповідає умовам локальної мережі навчальної лабораторії, оскільки поєднує простоту розгортання, достатню гнучкість, можливість масштабування, централізований контроль і придатність до подальшого розвитку. Унаслідок цього вона формує технічний фундамент для подальшого детального розгляду вибору засобів

реалізації, структури даних, логіки збору метрик і побудови інтерфейсу в наступних підрозділах роботи.

## 2.2 Обґрунтування вибору технічних та програмних засобів реалізації

У кваліфікаційній роботі вибір технічних та програмних засобів реалізації системи моніторингу стану ПК і серверів у локальній мережі виконано з урахуванням кількох базових вимог. Розроблюване рішення повинно забезпечувати безперервний збір системних метрик, підтримувати стабільне передавання даних у межах локальної мережі, зберігати часові ряди без надмірного ускладнення інфраструктури, надавати зручний веб-інтерфейс для перегляду графіків і при цьому залишатися достатньо простим для розгортання в умовах навчальної лабораторії. Саме тому вибір виконано не за принципом максимальної «масштабності», а за принципом збалансованості між функціональністю, простотою впровадження, ресурсною економністю та придатністю до подальшого розширення. У підсумку як основу реалізації обрано Python для агентної та серверної частини, FastAPI для серверного API, psutil для збору системних показників, HTTP і MQTT як канали передавання, InfluxDB для часових рядів, SQLite для службових і конфігураційних даних, React для веб-інтерфейсу, Chart.js для побудови графіків, а Docker Compose - для розгортання взаємопов’язаних компонентів сервісу.

Початковою точкою вибору стала агентна частина, оскільки саме вона безпосередньо працює на контрольованому вузлі й відповідає за зчитування показників стану системи. Для такого завдання потрібен інструмент, який дозволяє швидко створити фоновий сервіс, працює на різних платформах і має доступ до типових параметрів завантаження операційної системи. З цієї причини доцільним виявився Python, який у межах кваліфікаційної роботи розглянуто як мову з достатньо низьким порогом реалізації, широкою бібліотечною базою та зручністю роботи із системними й мережевими компонентами. Ключовим аргументом на користь такого вибору стала бібліотека psutil, яка в офіційній

документації прямо описується як кросплатформний засіб для отримання інформації про процеси та використання системних ресурсів, зокрема CPU, пам'яті, дисків, мережі та сенсорів. Це добре узгоджується з вимогами теми, оскільки система моніторингу повинна збирати саме ці типи показників, не вимагаючи складного доступу до низькорівневих механізмів ОС. Завдяки цьому реалізація агента на Python із використанням psutil виглядає логічною: вона дає достатню функціональність для лабораторного сценарію, дозволяє швидко розширювати набір метрик і не потребує побудови окремого складного драйверного шару.

Не менш важливим стало те, що Python зручний не лише для збору метрик, а й для реалізації серверної логіки. У межах системи моніторингу серверна частина виконує приймання метрик, перевірку їх структури, підготовку до запису в бази даних, надання API для фронтенда та обробку службових операцій. Для такої ролі було доцільно використати FastAPI. В офіційній документації FastAPI позиціонується як сучасний високопродуктивний веб-фреймворк для побудови API на Python, заснований на стандартних type hints, а також підтримує автоматичну генерацію OpenAPI-документації та інтерактивні інтерфейси документації. Для практичної реалізації це означає, що серверний компонент можна було побудувати у єдиному технологічному стеку з агентом, а модель даних - чітко описати через типи і схеми, що спрощує контроль правильності пакетів метрик. Для системи моніторингу це особливо важливо, оскільки вона працює з великою кількістю однотипних повідомлень, де будь-яка помилка у форматі даних може спричинити накопичення хибних записів або збої у візуалізації. Додатковою перевагою FastAPI стало те, що він добре підтримує асинхронну модель роботи, а також автоматично документує API, що спрощує тестування серверної частини під час розроблення та супроводу.

Для передавання метрик у межах локальної мережі в роботі обґрунтовано використання двох підходів - HTTP та MQTT. Вибір саме цієї пари не є випадковим, оскільки обидва протоколи добре доповнюють один одного. HTTP

є природним вибором для обміну з серверним API, оскільки забезпечує зрозумілу модель запит–відповідь, добре підтримується веб-фреймворками та зручний для відлагодження. Його використання доцільне тоді, коли потрібно просто і прозоро надсилати сформовані пакети метрик на сервер. Офіційні RFC-specifications для HTTP визначають сучасну семантику роботи протоколу, що робить його надійною та стандартизованою основою для побудови API-рівня. Водночас MQTT у стандарті OASIS описується як легкий клієнт-серверний publish/subscribe транспорт, простий в реалізації, відкритий і придатний для сценаріїв обміну короткими повідомленнями. Саме ця властивість робить його особливо доречним для системи моніторингу, де велика кількість вузлів регулярно надсилає невеликі пакети телеметрії. Додатково стандарт MQTT передбачає рівні якості доставки, що підвищує гнучкість вибору між мінімальними накладними витратами та більш надійним доставленням повідомлень. Через це використання MQTT є особливо доречним у тих частинах системи, де потрібен стабільний потік метрик і зручне розділення ролей між агентом, брокером та споживачами повідомлень.

У ролі брокера повідомлень доцільно використати Eclipse Mosquitto. Це рішення обрано з огляду на його орієнтацію саме на MQTT, відносну легкість розгортання та добру придатність для лабораторної інфраструктури. Офіційний сайт Mosquitto прямо позиціонує його як відкритий MQTT-брокер, а документація охоплює базові механізми конфігурації, автентифікації та взаємодії клієнтів. Для цієї роботи це важливо тому, що брокер не повинен перетворюватися на окрему складну адміністративну платформу. Його роль має залишатися чіткою: приймати повідомлення від агентів, забезпечувати їх доставлення серверним компонентам і підтримувати стабільну роботу publish/subscribe-моделі в межах локальної мережі. У таких умовах Mosquitto добре відповідає вимогам простоти, прозорості та достатньої функціональності.

Наступним ключовим питанням став вибір засобів зберігання даних. Оскільки система моніторингу працює насамперед із часовими рядами, звичайна

реляційна база даних не є оптимальною як єдине сховище для всіх типів інформації. Метрики надходять часто, мають часову природу, потребують ефективного групування за тегами й подальшої агрегації по часових інтервалах. Через це для метрик доцільно використати InfluxDB. Офіційна документація InfluxDB визначає line protocol як текстовий формат запису точок даних із measurement, tag set, field set і timestamp, що добре узгоджується з моделлю телеметрії вузлів. Окрім цього, документація окремо описує механізми retention, за якими система автоматично видаляє дані, часові мітки яких виходять за визначений період зберігання. Для лабораторного сценарію це має практичну цінність, бо дозволяє не накопичувати безмежний архів сирих метрик, а керувати обсягом сховища через політики зберігання. У підсумку InfluxDB обрано тому, що вона природно відповідає завданню роботи з телеметрією, спрощує побудову історичних графіків і не змушує адаптувати часові ряди під модель, яка для них не призначалася.

Паралельно з InfluxDB у системі виникає потреба зберігати службову інформацію іншої природи: реєстр вузлів, налаштування користувачів, ролі доступу, параметри сповіщень, конфігурацію каналів, локальні черги або допоміжні записи. Для таких сутностей доцільно використати SQLite. Офіційні матеріали SQLite визначають її як невелику, швидку, самодостатню, високонадійну, повнофункціональну SQL-систему, яка є serverless, in-process і не потребує окремого серверного процесу. Саме ці властивості роблять SQLite дуже зручною для конфігураційного рівня в дипломному проєкті: її не потрібно окремо адмініструвати, вона легко інтегрується в Python-додаток, а її модель добре підходить для зберігання відносно невеликих, але структурованих і пов’язаних між собою службових даних. Для навчальної лабораторії це особливо доречно, оскільки дозволяє не ускладнювати інфраструктуру ще одним повноцінним сервером баз даних там, де цього не вимагає функціональне навантаження. Розділення InfluxDB і SQLite виявляється практично

виправданим: перша відповідає за високочастотну телеметрію, а друга - за логіку конфігурації та керування сервісом.

Окремого обґрунтування потребує вибір засобу розгортання, оскільки система включає одразу кілька сервісів: API, фронтенд, брокер повідомлень, базу часових рядів і допоміжні компоненти. У такій ситуації ручне налаштування кожного елемента окремо створює високий ризик конфігураційних помилок і ускладнює відтворення середовища. Саме тому для роботи доцільно використати Docker Compose. Офіційна документація визначає Docker Compose як інструмент для опису та запуску multi-container applications, а також зазначає, що вся конфігурація застосунку задається в одному YAML-файлі, де визначаються сервіси, залежності, змінні оточення, мережі та томи. Для системи моніторингу це дуже важливо, оскільки дає змогу розгорнути всі компоненти як єдиний комплекс, зберегти узгодженість їх взаємодії і значно спростити перенесення рішення між різними машинами. У навчальній лабораторії така властивість має не лише технічну, а й практичну цінність: сервіс можна швидко відновити або продемонструвати без довготривалого ручного налаштування кожного окремого модуля.

Питання безпеки під час вибору засобів реалізації також враховано на рівні архітектурного рішення. Для API-рівня важливо, щоб сервер міг перевіряти структуру та зміст вхідних пакетів, а також працювати з контрольованими механізмами доступу. Для транспортного рівня MQTT суттєвим є те, що сам стандарт передбачає publish/subscribe-модель і рівні якості доставки, а безпечна реалізація в локальній мережі повинна доповнюватися автентифікацією клієнтів і сегментацією тем. Для загального підходу до довіри в інфраструктурі доречно орієнтуватися на сучасні принципи, подібні до логіки Zero Trust, де вузол не вважається автоматично надійним лише через перебування в локальній мережі. Це добре узгоджується з темою роботи, оскільки моніторингова система оперує службовими даними про стан вузлів, а тому повинна приймати метрики лише від відомих агентів і розмежовувати доступ до панелі спостереження.

Якщо розглядати обраний стек у цілісному вигляді, його перевага полягає саме у взаємній узгодженості компонентів. Python природно об’єднує агентний та серверний рівні. psutil покриває базову задачу збору метрик без зайвого ускладнення. FastAPI формує зрозумілий API-шар і забезпечує типізовану роботу з даними. HTTP і MQTT закривають потребу у двох зручних моделях передавання телеметрії. InfluxDB вирішує задачу збереження часових рядів, а SQLite - конфігураційної логіки. React і Chart.js забезпечують побудову інтерфейсу та графічного представлення результатів. Docker Compose зв’язує все це в єдиний відтворюваний спосіб розгортання. Саме така узгодженість і стала головним аргументом на користь цього набору засобів: кожен елемент не просто «підходить сам по собі», а логічно продовжує інший і працює в межах спільної архітектури. На рисунку 2.2 подано структуру програмних та технічних засобів реалізації системи моніторингу.

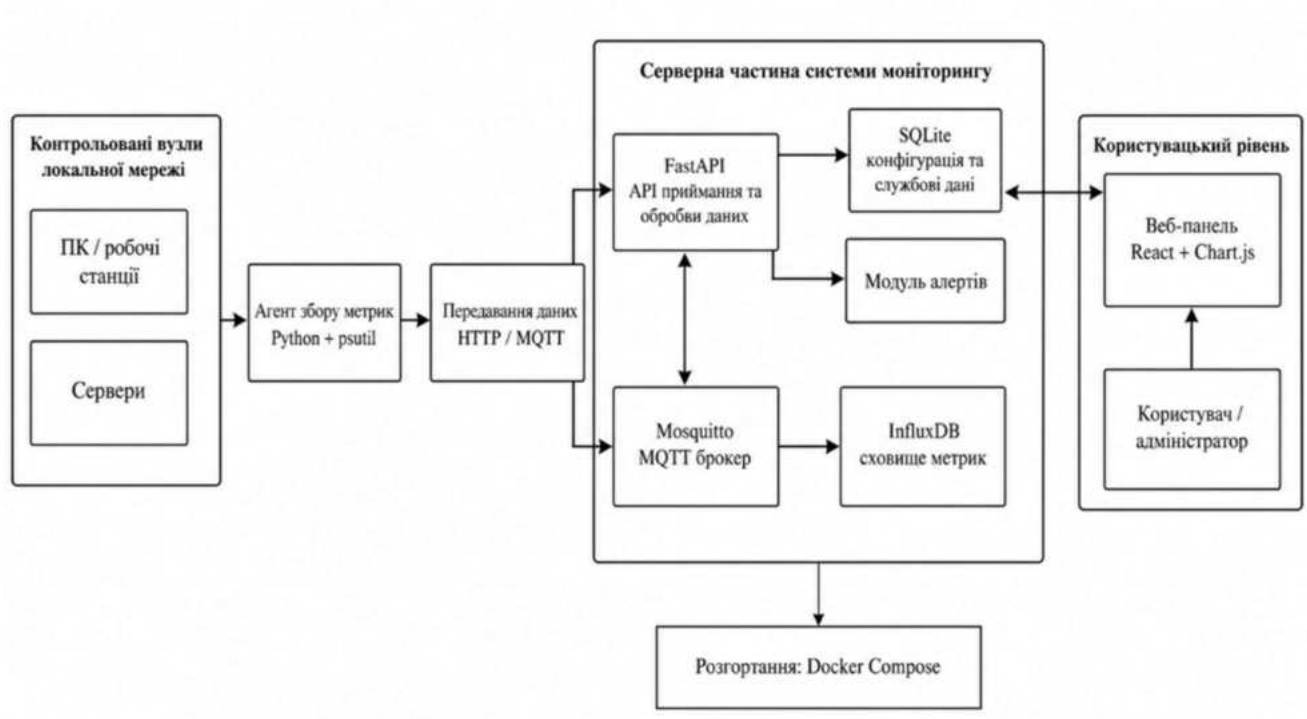



Рисунок 2.2 – Структура програмних та технічних засобів реалізації системи моніторингу

У підсумку вибір технічних та програмних засобів реалізації виконано з орієнтацією на практичні умови розгортання сервісу моніторингу в локальній мережі навчальної лабораторії. Обрані інструменти не створюють зайвої технологічної складності, але водночас забезпечують усі базові функції, необхідні для повноцінної роботи системи: збір системних метрик, стандартизовану передачу даних, розділене збереження часових рядів і конфігурацій, побудову наочного веб-інтерфейсу та зручне розгортання взаємопов’язаних сервісів. Це дозволяє вважати обраний стек достатньо обґрунтованим, технічно цілісним і придатним для подальшої реалізації наступних підсистем у межах кваліфікаційної роботи.

## 2.3 Розроблення структури даних та моделі зберігання метрик

У кваліфікаційній роботі розроблення структури даних та моделі зберігання метрик розглянуто як один із центральних етапів побудови системи моніторингу стану ПК і серверів у локальній мережі, оскільки саме від правильно організованого подання інформації залежить не лише коректність накопичення показників, а й подальша зручність їх аналізу, візуалізації, фільтрації та використання в механізмах сповіщень. Якщо архітектура системи визначає загальну логіку взаємодії між агентом, сервером і веб-панеллю, то модель даних задає внутрішній порядок усього цього процесу. Саме вона відповідає на питання, у якому вигляді повинні існувати метрики, як ідентифікуються вузли, яким способом поєднуються поточні показники з історичними записами, як зберігаються налаштування системи і яким чином різні типи інформації не змішуються між собою, але залишаються логічно пов’язаними.

Під час проєктування структури даних враховано, що система моніторингу працює не з одним типом інформації, а одразу з кількома сутнісно різними категоріями. Насамперед це самі метрики, тобто числові показники, що відображають стан конкретного вузла в конкретний момент часу. До цієї групи

належать завантаження процесора, використання оперативної пам’яті, стан дискової підсистеми, параметри мережевої активності, температури та інші значення, що можуть надходити від агента. Другою категорією виступають довідникові та конфігураційні дані, зокрема перелік вузлів, їх ідентифікатори, ролі, групи, назви, налаштування інтервалів збору, правила алертів, облікові записи користувачів, параметри доступу та службові журнали. Третьою категорією є похідні або аналітичні дані, що формуються вже на стороні серверної частини, наприклад агреговані значення за періоди, записи про спрацювання правил, технічні позначки статусу вузла або часові інтервали доступності. Усе це означає, що єдина універсальна таблиця або однаковий спосіб зберігання для всіх даних не був би доцільним. Через це модель побудовано за принципом розділення сутностей відповідно до їх природи та режиму використання.

Ключовим рішенням під час розроблення моделі зберігання стало відокремлення метрик як часових рядів від конфігураційної та довідкової інформації. Таке розділення має не формальний, а практичний характер. Метрики надходять часто, у великому обсязі, завжди мають часову прив’язку та зазвичай використовуються в запитах на кшталт «показати динаміку за останню годину», «знайти максимум за добу», «порівняти завантаження двох вузлів за тиждень». Натомість службові дані мають відносно стабільний характер: вузли реєструються, але не створюються щосекунди; правила сповіщень змінюються рідко; назви груп або ролі обладнання взагалі оновлюються нечасто. Саме тому в системі розроблено двокомпонентну модель зберігання: часові ряди метрик фіксуються окремо, а структуровані конфігураційні записи зберігаються в іншому логічному просторі. Це не лише зменшує змішування різнорідної інформації, а й дозволяє більш точно пристосувати спосіб доступу до кожної категорії даних.

На рівні логічної структури всієї системи центральним об’єктом виступає вузол моніторингу. Саме навколо нього формуються всі інші зв’язки. Під вузлом

у межах цієї моделі розуміється окремий ПК або сервер, з якого агент передає показники. Для кожного вузла задається унікальний ідентифікатор, який використовується як основний ключ зв'язку між метриками, подіями та конфігураційними параметрами. Окрім цього, вузол має описові характеристики: назву, тип, роль, розміщення, мережеву адресу, операційну систему, статус активності та інші властивості, що дозволяють інтерпретувати його місце в інфраструктурі. Наприклад, одна машина може бути визначена як робоча станція в конкретній лабораторії, інша - як сервер баз даних або вузол віртуалізації. Таке структурування є важливим не лише для зручного відображення у веб-панелі, а й для подальшої фільтрації, сортування, групування та призначення правил спостереження.

Наступною ключовою сутністю є запис метрики. У моделі зберігання він розглядається не як абстрактне число, а як точка спостереження, що містить кілька обов'язкових компонентів. По-перше, це ідентифікатор вузла, який дозволяє зрозуміти, до якого саме джерела належить запис. По-друге, це час отримання або формування метрики, без якого значення не може бути включене до часового ряду. По-третє, це назва або тип метрики, наприклад cpu_usage, memory_used, disk_free, network_rx, temperature_cpu. По-четверте, це числове значення. За потреби до цього запису додаються й інші поля, наприклад одиниця вимірювання, джерело, назва інтерфейсу або мітка групування. Завдяки такій структурі одна й та сама система може зберігати різнорідні показники в уніфікованому форматі, а серверна частина отримує можливість однаково працювати з усіма типами даних - незалежно від того, чи це навантаження процесора, чи температура.

Для підвищення логічної впорядкованості в межах моделі метрики доцільно групуються за категоріями. Окремо виділяються процесорні метрики, окремо - пам'ять, окремо - диски, мережа, температури та службові технічні показники. Це не означає, що для кожного виду обов'язково створюється зовсім ізольоване сховище. Йдеться про логічний рівень структурування, який надалі

використовується для побудови графіків, формування вкладок інтерфейсу та створення правил аналізу. Наприклад, дані про CPU можуть мати підвиди: загальне завантаження, завантаження по ядрах, середнє навантаження за інтервал, кількість активних процесів. Для пам'яті можуть зберігатися загальний обсяг, використаний обсяг, вільний обсяг, кеш, swap. Дискова підсистема може описуватися як у межах логічних розділів, так і у вигляді показників читання та запису. Така деталізація дозволяє не обмежуватись одним «плоским» значенням, а створити модель, у якій система поступово нарощує інформативність без руйнування загальної схеми.

Оскільки система орієнтована на роботу з часовими рядами, принцип часової впорядкованості закладено в модель як один із базових. Кожне значення не просто записується в базу, а розміщується в послідовності, яка дозволяє простежити зміну стану вузла в динаміці. Це означає, що навіть якщо два записи мають однаковий тип метрики та належать одному вузлу, їх розрізнення здійснюється передусім через часову мітку. Унаслідок цього можна будувати лінії змін, обчислювати середні значення на відрізку, знаходити піки навантаження, фіксувати моменти переходу до небезпечного режиму. У межах кваліфікаційної роботи це має принципове значення, бо сама ідея моніторингу полягає не лише у фіксації поточного числа, а в оцінці того, як цей показник поводиться з часом.

Під час розроблення моделі особливу увагу приділено тегам, які виконують роль допоміжних ознак для групування та фільтрації. На відміну від основного значення метрики, тег не описує сам показник, а уточнює контекст його виникнення. Наприклад, для мережевої активності тегом може бути назва інтерфейсу, для дискової підсистеми - позначення розділу або накопичувача, для температур - тип сенсора, для вузла - його група чи роль у мережі. Саме теги дозволяють побудувати більш гнучку модель відображення даних, коли користувач веб-панелі може переглянути не просто «мережу на вузлі», а активність конкретного мережевого інтерфейсу, або не просто «диск», а окремий

логічний том. У підсумку теги стають одним із механізмів деталізації інформації без ускладнення базової схеми.

Конфігураційна частина моделі даних побудована на окремих сутностях, які обслуговують логіку функціонування системи. Серед них важливе місце займає таблиця або еквівалентна структура вузлів, де зберігається інформація про всі зареєстровані ПК та сервери. Далі йде сутність користувачів, якщо система передбачає автентифікацію доступу до веб-панелі. Окремо зберігаються ролі, що визначають рівень доступу до перегляду, редагування або адміністрування.

Окремого значення набуває модель зберігання подій доступності вузла. У системі моніторингу стан вузла не завжди описується лише окремими метриками. Іноді сама відсутність нових значень уже є важливою інформацією, яка може свідчити про втрату зв’язку, вимкнення машини або аварійну зупинку агента. Саме тому в моделі даних передбачено поняття останнього контакту з вузлом, яке використовується для визначення його поточного статусу. Це може бути окрема службова мітка, що оновлюється з кожним прийнятим пакетом, або окремий запис про доступність. Завдяки цьому серверна частина має змогу не лише відображати метрики, а й оцінювати життєвий стан вузла в реальному часі.

Важливою частиною розроблення структури даних є й спосіб подання пакетів, що надходять від агента. На транспортному рівні вони мають бути достатньо компактними, але водночас однозначними з точки зору обробки. З цієї причини доцільно формувати пакет як об’єкт, у якому містяться загальні атрибути вузла та масив метрик. Такий підхід зменшує дублювання даних: ідентифікатор вузла, час формування пакета, службові теги передаються один раз, а далі список конкретних значень розшифровується серверною частиною і розкладається у сховище. У межах серверної моделі один транспортний пакет може породжувати одразу кілька записів часового ряду, кілька оновлень статусу вузла та, за потреби, одну або більше перевірок правил алертів. Отже, модель

даних охоплює не тільки кінцеве зберігання, а й логіку переходу від пакета агента до внутрішніх записів системи.

На рівні веб-інтерфейсу структура даних повинна бути придатною до швидкого отримання вибірок для відображення. Це означає, що модель зберігання проєктується не лише «для запису», а й «для читання». Наприклад, сторінка загального огляду повинна швидко показувати список вузлів, їхній статус, час останньої активності та кілька основних поточних показників. Сторінка конкретного вузла повинна отримувати історію певної метрики за вибраний інтервал без складних багатоступеневих перетворень. Блок сповіщень повинен мати змогу швидко отримати недавні події або критичні стани. Усе це вимагає такої моделі, де дані не тільки правильно зберігаються, а й логічно впорядковуються для подальшого використання інтерфейсом. Унаслідок цього модель зберігання формується не ізольовано від веб-панелі, а в безпосередньому зв’язку з нею. На рисунку 2.3 подано логічну модель даних системи моніторингу.

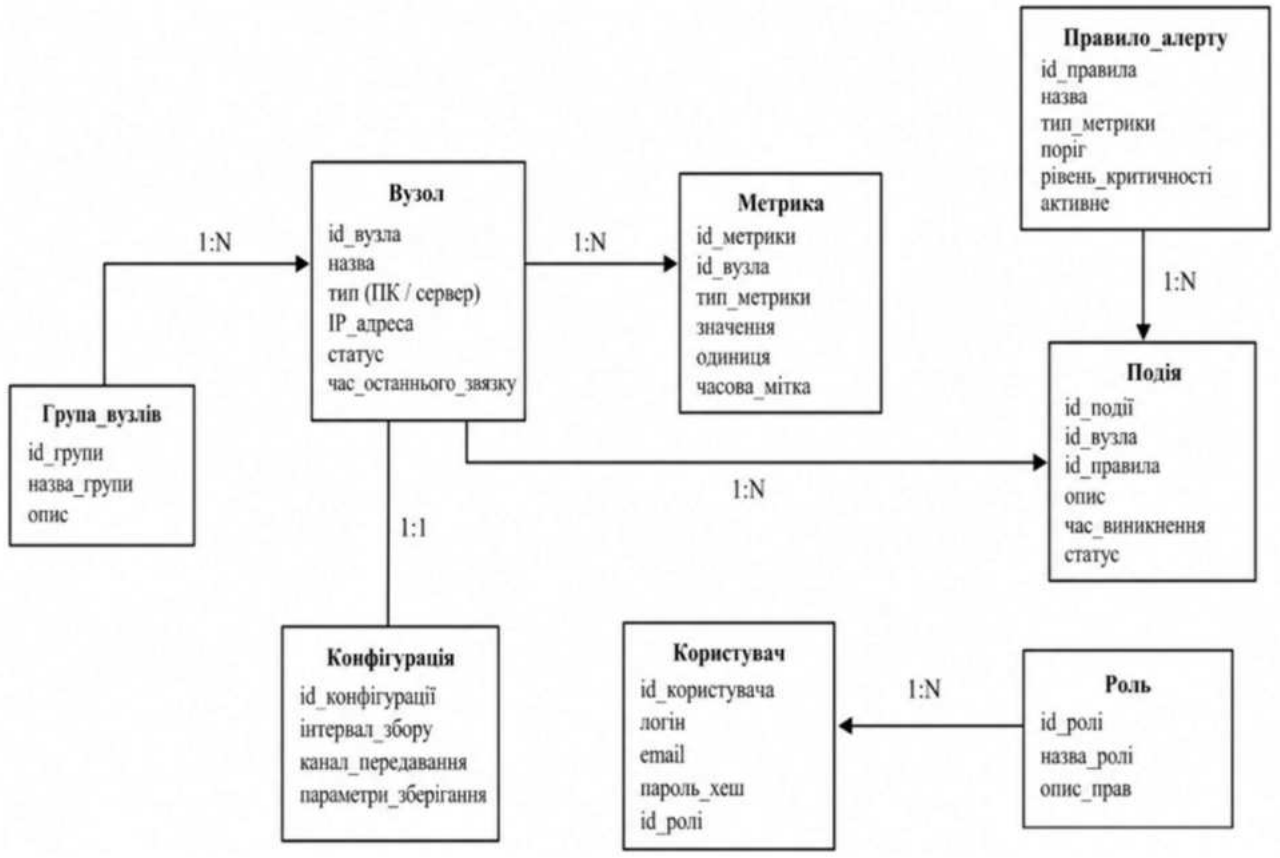



Рисунок 2.3 – Логічна модель даних системи моніторингу

Не менш важливим є й питання цілісності даних. У системі моніторингу особливо небезпечними є дві ситуації: накопичення дубльованих записів і поява «сирітських» даних, які не можна пов’язати з конкретним вузлом або правилом. Саме тому в логічній моделі для кожного типу записів передбачено чіткі зв’язки. Метрика не існує без вузла. Подія алерту не існує без правила та часової прив’язки. Користувач веб-панелі не існує без визначеної ролі. Такий підхід дає можливість підтримувати впорядкованість системи навіть тоді, коли кількість даних поступово зростає. У межах кваліфікаційної роботи ця властивість особливо важлива, оскільки вона демонструє, що система спроєктована не як тимчасовий прототип, а як повноцінне прикладне рішення з продуманою внутрішньою організацією.

У підсумку розроблення структури даних та моделі зберігання метрик виконано як побудову багаторівневої логічної схеми, у межах якої часові ряди, службові дані, конфігураційні сутності та події системи утворюють узгоджене інформаційне середовище. Центральними елементами цієї моделі визначено вузол моніторингу, запис метрики, правила сповіщень, журнали подій та конфігураційні таблиці, що описують користувачів, ролі та групи обладнання. Такий підхід дозволяє не лише надійно накопичувати показники стану ПК і серверів, а й створює основу для подальшого аналізу, побудови графіків, контролю доступності та формування алертів. Саме тому розроблена модель даних виступає одним із фундаментальних елементів усієї системи моніторингу, забезпечуючи її цілісність, масштабованість і придатність до практичного використання в умовах локальної мережі навчальної лабораторії.

## 2.4 Організація збору, обробки та передачі даних

Якщо в загальній архітектурі було визначено склад компонентів, а в моделі даних - внутрішню структуру інформації, то на цьому етапі показано, як саме

дані з’являються, у якому вигляді вони проходять через систему, які перетворення з ними виконуються та за рахунок чого досягається цілісність усього процесу. Для системи моніторингу стану ПК і серверів у локальній мережі цей аспект має принципове значення, оскільки корисність сервісу визначається не тільки наявністю баз даних чи графіків, а передусім тим, наскільки надійно і впорядковано організовано шлях даних від джерела до кінцевого відображення.

Початковим етапом цього циклу є збір метрик на стороні вузла. У межах реалізації системи цю функцію виконує локальний агент, який працює у фоновому режимі на кожному контрольованому ПК або сервері. Його основне завдання полягає у періодичному отриманні показників стану операційної системи та апаратних ресурсів без безпосереднього втручання користувача. Така модель є виправданою з практичного погляду, оскільки саме локальний запуск агента забезпечує доступ до актуальних значень завантаження процесора, використання оперативної пам’яті, дискового простору, мережевої активності й температурних сенсорів. Для реалізації цього рівня обрано бібліотеку psutil, офіційна документація якої прямо вказує, що вона є кросплатформним засобом для отримання інформації про CPU, пам’ять, диски, мережу та сенсори в Python. Це робить її природним інструментом для побудови агентної частини саме в задачі системного моніторингу.

Робота агента організована циклічно. Через заздалегідь визначений інтервал часу виконується послідовне опитування системних джерел даних, після чого отримані значення збираються в єдиний внутрішній набір показників. Такий підхід обрано не випадково. Для системи моніторингу важливо, щоб інтервал збору був достатньо малим для фіксації короткочасних піків навантаження, але водночас не створював надмірного навантаження на сам вузол і не породжував зайвого потоку телеметрії. Саме тому збір організовано не в безперервному режимі, а через періодичні ітерації, де кожна ітерація формує повний зріз поточного стану машини. У межах такого зрізу окремо зчитуються значення CPU, оперативної пам’яті, логічних дисків, мережевих інтерфейсів і,

якщо доступно, температурних параметрів. У результаті формується структура даних, яка описує не один ізольований параметр, а загальний стан вузла в конкретний момент часу.

Після отримання початкових системних значень агент переходить до етапу внутрішньої обробки. На цьому рівні виконується не складний аналітичний розрахунок, а базова підготовка метрик до подальшого передавання. Насамперед зібрані значення приводяться до єдиного формату. Це означає, що показники, які отримані з різних системних джерел, уніфікуються за іменами, одиницями вимірювання та способом подання. Наприклад, відносні величини подаються у відсотках, об'єми пам'яті - у байтах або у похідних величинах, мережеві лічильники - як обсяг переданих чи прийнятих даних, температури - у градусах. Паралельно до кожного набору метрик додається часовий штамп, а також ідентифікатор вузла, що дозволяє серверній частині однозначно пов'язати отримані дані з конкретною машиною. На цьому ж рівні може виконуватися фільтрація допоміжних або нестабільних значень, якщо вони не несуть практичної користі для моніторингового сценарію. Унаслідок цього агент не просто передає «сирі числа», а формує пакет даних, який уже готовий до стабільної серверної обробки.

Особливістю організації збору є те, що агент працює автономно від серверної частини. Це означає, що він не залежить від постійного зовнішнього опитування з боку сервера, а самостійно ініціює процес формування та надсилання даних. Такий підхід виявився найбільш доречним для умов локальної мережі навчальної лабораторії, оскільки окремі машини можуть бути недоступними, вимкненими або працювати в різні періоди часу. Якщо б система повністю спиралася на модель зовнішнього опитування, то сервер був би змушений постійно перевіряти доступність кожного вузла, а це ускладнювало б інфраструктуру і робило б її більш чутливою до змін мережевого стану. Натомість автономний агент дозволяє вузлу самому визначати момент відправлення пакета та підтримує природну модель роботи для гетерогенного

середовища, де не всі машини поводяться однаково. На рисунку 2.4 подано схему збору, обробки та передачі даних у системі моніторингу.

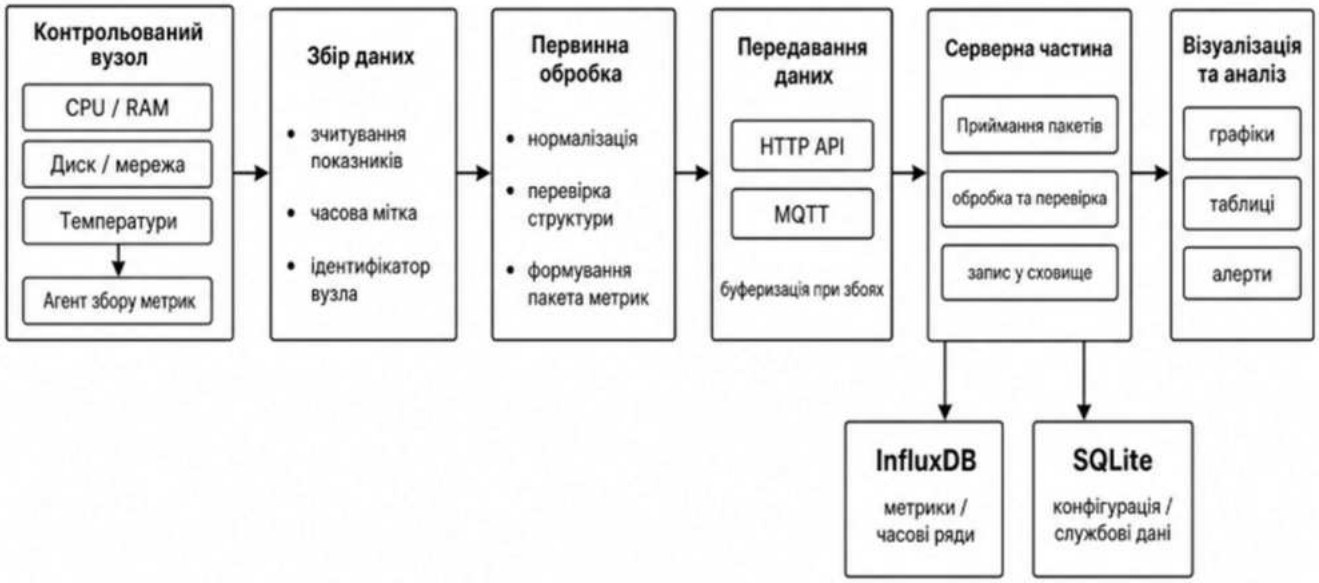



Рисунок 2.4 – Схема збору, обробки та передачі даних у системі моніторингу

Наступним етапом після формування пакета метрик є його передавання до серверної частини. У межах системи обґрунтовано два канали передавання - HTTP та MQTT. HTTP використовується як прямий канал обміну між агентом і серверним API. У такій моделі агент формує запит із тілом, що містить пакет метрик, а сервер приймає його як request body, виконує перевірку та повертає службову відповідь. Офіційна документація FastAPI прямо описує request body як дані, що надсилаються клієнтом до API, і це добре узгоджується з нашою схемою, де агент є клієнтом, а сервер - приймальною точкою. Така модель є зручною, прозорою та простою для відлагодження. Вона добре підходить для локального середовища, де важливими є зрозумілість маршрутизації, явна логіка запиту й можливість швидко перевірити вміст переданого пакета.

Поряд з HTTP у системі використовується MQTT як альтернативний або додатковий канал передавання телеметрії. Його застосування є особливо виправданим для сценарію, де багато вузлів регулярно надсилають короткі повідомлення з метриками, а серверна частина може бути розділена на кілька

логічних споживачів цих даних. Офіційний стандарт MQTT 5.0 визначає цей протокол як легкий клієнт-серверний publish/subscribe transport, простий у реалізації та орієнтований на ефективний обмін повідомленнями. У практичній реалізації це означає, що агент не повинен напряму взаємодіяти з кожним серверним компонентом. Він лише публікує повідомлення в потрібну тему, а далі брокер забезпечує їх доставку підписаним модулям. Такий підхід зменшує жорстку зв'язаність між елементами системи та спрощує масштабування серверної обробки. Крім того, MQTT підтримує різні рівні якості доставки, що важливо для моніторингової телеметрії, де баланс між продуктивністю та надійністю може налаштовуватися залежно від критичності повідомлень.

Незалежно від того, який транспортний канал використано, після надсилання агентом дані мають пройти етап контролю доставки. У локальній мережі не можна повністю виключити короткочасні перебої: сервер може перезавантажуватися, брокер може бути тимчасово недоступним, мережевий сегмент може втратити зв'язок, а вузол - тимчасово перейти в офлайн-стан. Саме тому організація передавання в системі побудована з урахуванням механізму буферизації. Якщо пакет не вдається передати відразу, він не відкидається, а тимчасово зберігається на стороні агента. Після відновлення з'єднання такі пакети відправляються повторно в тому самому логічному порядку. Цей підхід дозволяє уникнути невидимих провалів у часових рядах і робить систему стійкішою до короткочасних мережевих збоїв. Разом із цим буферизація має контрольовані межі, щоб накопичення недоставлених записів не перетворилося на окрему проблему для вузла, на якому працює агент.

Саме такий формат описує офіційна документація InfluxDB line protocol, і він добре відповідає задачі системного моніторингу. У практичному сенсі це означає, що назва типу метрики або групи метрик використовується як measurement, службові ознаки на кшталт host_id чи ролі вузла - як теги, а самі числові значення - як поля. Така модель дозволяє легко виконувати вибірки за вузлами, за видами метрик або за часовими інтервалами. Паралельно з цим

службові й конфігураційні дані зберігаються окремо, що дає змогу не перевантажувати часове сховище сутностями, які мають іншу природу й інші сценарії використання.

Окремою частиною серверної обробки є агрегація. Вона потрібна з двох причин. По-перше, система моніторингу повинна вміти показувати не лише сирі щосекундні або щохвилинні значення, а й усереднені показники за більші часові вікна. По-друге, надто детальне довготривале зберігання метрик призводить до швидкого росту обсягу даних. Через це організація обробки передбачає побудову запитів або додаткових механізмів, які дозволяють отримувати усереднені, максимальні чи мінімальні значення за інтервал. У практичній роботі це проявляється в тому, що для короткого періоду може використовуватись більш деталізована історія, а для довгого - вже агреговані значення. Такий підхід робить систему зручнішою для аналітики й одночасно більш стійкою з погляду ресурсів серверного сховища.

Паралельно із записом метрик виконується перевірка умов для можливих алертів. На цьому етапі окремі значення або їхні комбінації співставляються з правилами, заданими в конфігурації системи. Якщо, наприклад, завантаження процесора перевищує допустимий поріг протягом визначеного часу, або температура компонента зростає вище встановленого рівня, або сервер перестає надсилати дані, формується подія тривоги. Організація цього процесу важлива тим, що система перестає бути пасивним накопичувачем телеметрії й набуває властивостей активного засобу контролю. Саме в цьому полягає одна з практичних переваг розроблюваного рішення: воно не просто відображає показники, а дає змогу своєчасно виявляти проблемні стани.

Після запису й первинної серверної обробки дані стають доступними для веб-панелі. На цьому рівні організація процесу вже орієнтована не на телеметрію як таку, а на її відображення користувачеві. Сервер формує вибірки за вузлом, типом метрики та часовим інтервалом, після чого передає їх у фронтенд. Для відображення графіків і таблиць важливо, щоб дані були не лише збережені, а й

логічно впорядковані та готові до швидкої вибірки. Саме тому весь попередній цикл - від локального збору до запису в часове сховище - підпорядковано не лише задачі накопичення, а й задачі подальшої інтерпретації. Внаслідок цього користувач може побачити як загальну картину стану мережі, так і детальну історію конкретного вузла без необхідності працювати з «сирими» даними напряму.

У підсумку організація збору, обробки та передачі даних у системі моніторингу побудована як послідовний, багатоетапний і логічно зв’язаний процес. Агент на вузлі виконує локальне зчитування системних показників, формує уніфікований пакет і передає його через HTTP або MQTT. Серверна частина приймає цей пакет, виконує перевірку структури, нормалізує його відповідно до внутрішньої моделі, записує метрики у сховище часових рядів, оновлює службову інформацію та перевіряє умови для сповіщень. Після цього дані стають основою для графіків, таблиць і аналітики у веб-панелі. Така організація дозволяє забезпечити безперервність моніторингу, стійкість до короткочасних збоїв зв’язку, узгодженість усіх етапів обробки та придатність системи до реального використання в локальній мережі навчальної лабораторії.

## 2.5 Організація взаємодії з користувачем та візуалізації результатів

Якщо агентний рівень відповідає за отримання даних, серверний - за їх приймання, обробку та збереження, то користувацький рівень забезпечує сприйняття всієї цієї інформації в упорядкованому, наочному та придатному для інтерпретації вигляді. Для системи моніторингу це має принципове значення, тому що навіть технічно досконале сховище часових рядів не дає реальної користі, якщо адміністратор або відповідальна особа не може швидко побачити поточний стан мережі, визначити проблемний вузол, простежити динаміку зміни параметрів і зрозуміти, коли саме почало формуватися відхилення.

Під час проєктування взаємодії з користувачем враховано, що система призначена для роботи в умовах локальної мережі навчальної лабораторії, де потрібна не надлишково складна аналітична платформа, а зручний і зрозумілий інструмент щоденного контролю стану ПК і серверів. Через це в основу користувацької частини покладено принцип наочності, швидкого доступу до ключових показників та логічного переходу від загального огляду до детальної інформації. Такий підхід означає, що інтерфейс не повинен змушувати користувача працювати з сирими записами метрик або вручну формувати запити до бази даних. Навпаки, вся логіка побудована так, щоб результати роботи системи моніторингу відображалися у формі, яка одразу дає змогу оцінити ситуацію, виявити потенційно проблемні стани та перейти до глибшого перегляду лише тоді, коли це справді потрібно.

У межах розроблюваного рішення основним засобом взаємодії з користувачем визначено веб-панель, яка працює як централізований інтерфейс доступу до результатів моніторингу. Вибір саме веб-формату зумовлений кількома причинами. По-перше, такий підхід не вимагає встановлення окремого клієнтського програмного забезпечення на кожну машину, з якої може здійснюватися перегляд результатів. По-друге, веб-панель природно вписується в архітектуру системи, де серверна частина вже працює через API, а отже фронтенд може звертатися до неї стандартизованими HTTP-запитами. По-третє, веб-інтерфейс є зручним для централізованого доступу з різних вузлів локальної мережі, що особливо важливо для навчального середовища, де контроль за інфраструктурою може здійснюватися з різних робочих місць. Для побудови інтерфейсу доцільно використати React, оскільки офіційна документація React визначає його як засіб створення інтерфейсів через систему компонентів, які комбінуються у більші частини екрана. Саме така компонентна структура добре відповідає завданню моніторингової панелі, де окремими повторюваними блоками виступають списки вузлів, картки стану, графіки, таблиці подій та панелі фільтрації.

Логіка взаємодії з користувачем у системі вибудувана за ієрархічним принципом. На верхньому рівні користувач отримує загальний огляд стану всієї інфраструктури. На цьому екрані відображається список або набір карток вузлів, де для кожної машини показуються базові ознаки її стану: назва, роль, останній час зв'язку, статус доступності та кілька основних поточних показників, наприклад завантаження процесора, використання оперативної пам'яті або загальний рівень заповнення диска. Такий спосіб подання вибрано не випадково. Він дозволяє за короткий час охопити всю мережу поглядом і відразу зрозуміти, чи є вузли, що потребують уваги. Саме цей екран виконує роль точки входу в систему моніторингу й забезпечує те, чого зазвичай бракує при ручному контролі - цілісну картину всієї інфраструктури, а не уривчасті перевірки окремих машин.

Організація графічного представлення даних у межах системи передбачає, що кожна категорія метрик може мати власний тип відображення. Для CPU доцільними є лінійні графіки із часовою віссю, які показують динаміку загального навантаження або навантаження по ядрах. Для пам'яті зручно використовувати графіки зайнятого та вільного обсягу, що дає змогу візуально оцінювати стабільність або поступове зростання споживання ресурсу. Для дисків корисними є як абсолютні значення зайнятого простору, так і графіки читання/запису, якщо вони підтримуються агентом. Для мережі важливо показувати окремо приймання та передавання трафіку, а для температур - часову зміну значень по доступних сенсорах. Такий підхід дозволяє адаптувати форму візуалізації до природи конкретної метрики, а не намагатися показувати всі типи даних одним універсальним, але менш інформативним способом.

Суттєву роль у взаємодії з користувачем відіграє механізм вибору часових інтервалів. У моніторинговій системі одна й та сама метрика набуває різного значення залежно від того, за який період її розглядати. Для оперативного контролю зручними є короткі інтервали - кілька хвилин або година, де видно актуальну реакцію системи на поточне навантаження. Для пошуку тенденцій кориснішими стають проміжки у межах доби, кількох днів або тижня. Саме тому

в інтерфейсі доцільно передбачити швидкий вибір типових часових відрізків, наприклад «15 хвилин», «1 година», «24 години», «7 днів». Це суттєво підвищує зручність користування, оскільки дає можливість переходити між оперативним і ретроспективним режимами аналізу без складних ручних налаштувань.

Окрім графіків, важливим елементом візуалізації є табличне подання інформації. У системі моніторингу таблиці не замінюють графіки, а доповнюють їх. Графік добре показує тенденцію, але не завжди дозволяє швидко побачити точні значення або перелік останніх подій. Саме тому в межах панелі доцільно використовувати таблиці для відображення списку вузлів, журналу алертів, часу останньої активності, коротких зведень про стан системи та результатів фільтрації. Такий спосіб представлення особливо корисний тоді, коли потрібно швидко впорядкувати вузли за рівнем навантаження, знайти машини без зв’язку або переглянути останні критичні події. Унаслідок цього графічна та таблична візуалізація не конкурують між собою, а формують єдине середовище взаємодії, де кожен інструмент використовується у своєму найбільш придатному сценарії.

Окремого значення набуває візуальне відображення поточного статусу вузлів. Для швидкого сприйняття стану системи користувачу недостатньо просто бачити набір числових полів. Необхідно, щоб інтерфейс відразу давав зрозуміти, чи є вузол доступним, чи передає метрики, чи перебуває в нормальному режимі, чи потребує уваги. Через це в панелі доцільно використовувати статусні позначки, кольорові індикатори або маркери стану. Наприклад, вузол може відображатися як «активний», «недоступний», «попередження», «критичний». Таке подання значно прискорює сприйняття інформації, особливо коли кількість контрольованих машин зростає. У навчальній лабораторії це практично важливо, оскільки відповідальна особа повинна мати змогу швидко знайти проблемну машину без детального перегляду кожного окремого графіка.

Ще одним важливим аспектом взаємодії з користувачем є механізм сповіщень. У межах системи моніторингу взаємодія не повинна зводитися лише до сценарію, коли користувач сам відкриває веб-панель і шукає відхилення.

Частина інформації має подаватися у формі активних сигналів, коли система самостійно інформує про настання важливої події. Для цього в інтерфейсі організовується окремий блок алертів, де відображаються поточні й історичні спрацювання правил. Такий блок може містити назву вузла, тип проблеми, час виникнення, рівень критичності та стан алерту. Це особливо корисно у випадках, коли проблема вже завершилася, але потребує подальшого аналізу, або коли потрібно простежити, чи є відхилення повторюваними. У результаті користувач отримує не лише інформацію про «живий» стан системи, а й журнал подій, який допомагає осмислювати роботу інфраструктури в ретроспективі.

Важливим принципом організації взаємодії є й мінімізація зайвих дій користувача. У хорошій панелі моніторингу шлях до потрібної інформації не повинен бути довгим. Саме тому доцільно передбачити пошук вузлів за назвою, фільтрацію за роллю або групою, швидке сортування за станом чи рівнем навантаження. Це робить інтерфейс придатним до практичного використання в умовах, коли кількість контрольованих машин вже не обмежується двома-трьома прикладами. У навчальній лабораторії навіть кілька десятків вузлів можуть створити ситуацію, де без фільтрації або сортування панель втрачає оперативну зручність. Отже, взаємодія з користувачем повинна бути побудована не лише красиво, а й функціонально продумано.

З технічного боку організація взаємодії з користувачем безпосередньо пов’язана з API-рівнем серверної частини. Фронтенд отримує дані не напряму зі сховища, а через серверні маршрути, які повертають готові до відображення вибірки. Такий підхід є правильним з кількох причин. По-перше, він дозволяє приховати внутрішню логіку зберігання та обробки даних від користувацького рівня. По-друге, він централізує перевірку доступу та валідацію параметрів запитів. По-третє, він робить систему більш гнучкою: за потреби внутрішню структуру зберігання можна змінювати без повної перебудови інтерфейсу. Офіційна документація FastAPI підкреслює зручність побудови API для

структурованої взаємодії із фронтендом, що добре відповідає ролі серверної частини в нашій системі.

Окремо варто відзначити, що візуалізація результатів повинна бути не лише інформативною, а й стійкою до змін обсягу даних. У моніторинговій системі кількість точок на графіках може швидко зростати, особливо якщо користувач обирає довгі часові відрізки або якщо система збирає дані з невеликим інтервалом. Через це організація відображення повинна враховувати агрегацію, обмеження кількості точок, адаптацію часової шкали та комфортне масштабування. Документація Chart.js окремо описує time axis і timeseries axis як засоби для побудови часових графіків, що дозволяє природно працювати з метриками моніторингу та автоматично впорядковувати дані за часом. Саме це робить бібліотеку доречною для нашого сценарію, де основною одиницею візуалізації є не просто набір чисел, а часовий ряд поведінки ресурсу. На рисунку 2.5 подано схему взаємодії користувача з веб-панеллю системи моніторингу.

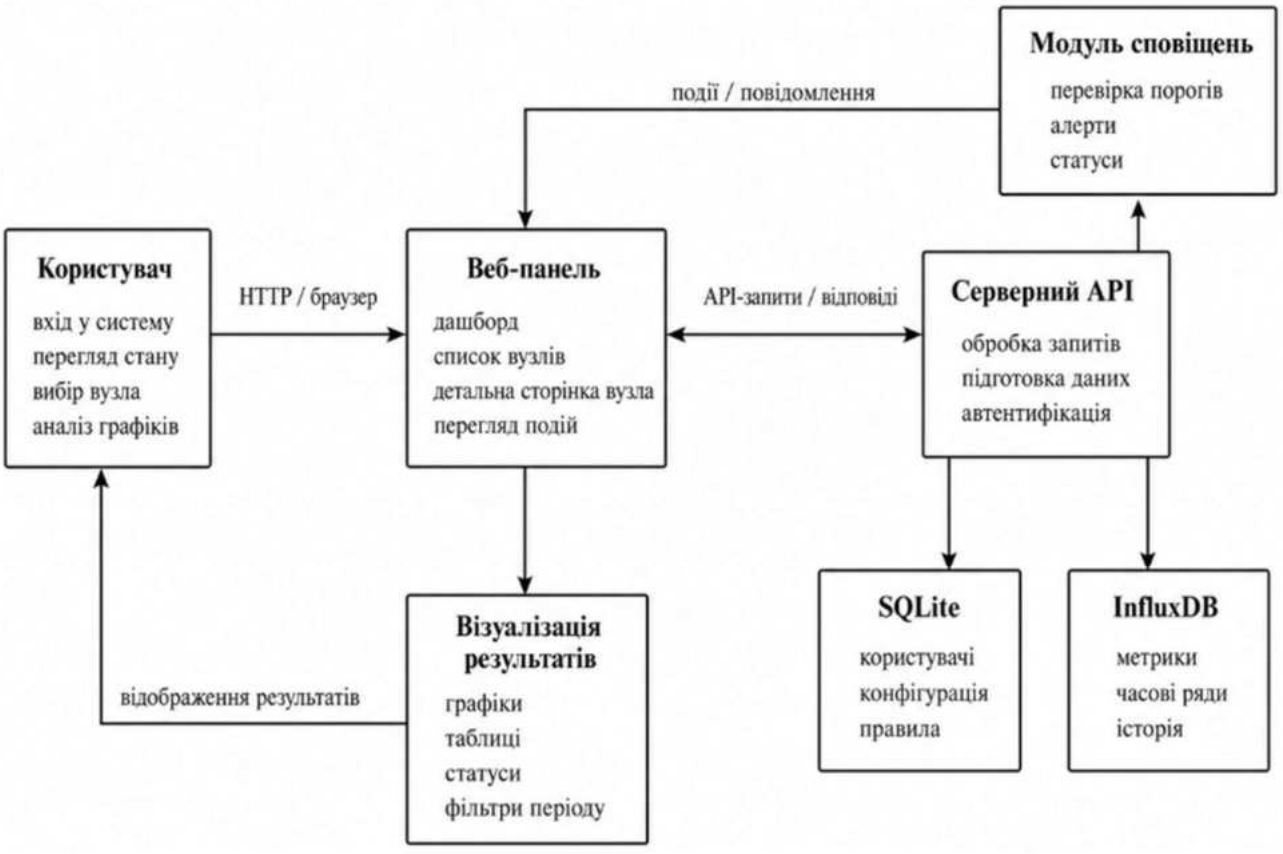



Рисунок 2.5 – Схема взаємодії користувача з веб-панеллю системи моніторингу

У підсумку організацію взаємодії з користувачем та візуалізації результатів у межах системи моніторингу побудовано як цілісну модель, де веб-панель виступає центральною точкою доступу до всієї інформації про стан ПК і серверів у локальній мережі. У цій моделі користувач спочатку отримує загальний огляд інфраструктури, далі переходить до детального перегляду конкретного вузла, аналізує історичні графіки, переглядає таблиці подій і за потреби реагує на спрацювання алертів. Використання React для організації інтерфейсу та Chart.js для побудови графіків є обґрунтованим, оскільки ці засоби добре відповідають компонентній природі веб-панелі та часовому характеру метрик. Унаслідок цього результати роботи системи моніторингу набувають зручної для сприйняття форми, а сама взаємодія з користувачем перетворюється на реальний інструмент щоденного контролю, аналізу та підтримки стабільної роботи обчислювальної інфраструктури.

## 2.6 Висновки до другого розділу

У другому розділі кваліфікаційної роботи спроєктовано та змістовно обґрунтовано основні складові системи моніторингу стану ПК і серверів у локальній мережі. На цьому етапі сформовано загальну архітектуру програмного рішення, визначено логіку взаємодії між агентним рівнем, серверною частиною, сховищами даних і веб-інтерфейсом, а також описано принципи функціонування всієї системи як єдиного сервісу. Унаслідок цього створено цілісне уявлення про те, яким чином відбувається рух інформації від контрольованого вузла до кінцевого відображення у веб-панелі та формування сповіщень.

У межах підрозділу, присвяченого загальній архітектурі, сформовано багаторівневу модель системи моніторингу, у якій кожен компонент виконує чітко визначену роль. Показано, що агентний модуль забезпечує локальний збір системних показників, транспортний рівень підтримує їх передавання, серверна

частина виконує приймання, перевірку та обробку, сховища забезпечують накопичення часових рядів і службових даних, а веб-інтерфейс формує зручне середовище для сприйняття результатів. Такий підхід дозволив побудувати архітектуру, що поєднує централізований контроль, достатню гнучкість, масштабованість та придатність до розгортання в умовах навчальної лабораторії.

Під час обґрунтування вибору технічних і програмних засобів реалізації визначено стек технологій, який найбільш повно відповідає поставленій задачі. Для реалізації агентного та серверного рівнів обрано Python, для збору системних метрик - psutil, для серверної взаємодії - FastAPI, для передавання даних - HTTP і MQTT, для збереження метрик - InfluxDB, для конфігураційної та службової інформації - SQLite, а для побудови веб-інтерфейсу - React і засоби графічної візуалізації. Таке рішення виявилося збалансованим з точки зору функціональності, простоти впровадження та практичної доцільності, оскільки воно не створює надлишкової складності, але водночас забезпечує всі необхідні можливості для реалізації повноцінної системи моніторингу.

У підрозділі, присвяченому організації збору, обробки та передачі даних, визначено повний цикл руху інформації в системі. Показано, що дані зчитуються агентом на вузлі, проходять етап первинної нормалізації, формуються в уніфіковані пакети, передаються до серверної частини через стандартизовані канали, після чого проходять перевірку, запис у сховища й підготовку до подальшої аналітики. Окремо враховано буферизацію при короткочасних збоях зв’язку, що суттєво підвищує надійність сервісу та дозволяє зменшити ризик втрати метрик. У результаті описано не окремі ізольовані операції, а повноцінну логіку роботи всієї системи моніторингу як єдиного ланцюга обробки даних.

У підрозділі, де розглянуто організацію взаємодії з користувачем та візуалізації результатів, сформовано підхід до побудови веб-панелі як завершального функціонального рівня системи. Показано, що саме через веб-інтерфейс користувач отримує загальний огляд мережі, доступ до детальної інформації про окремі вузли, графіки зміни показників, таблиці подій і блоки

сповіщень. Такий підхід дозволив перетворити накопичені метрики на практично корисний інструмент контролю стану інфраструктури. Завдяки цьому система моніторингу виступає не лише як механізм накопичення числових значень, а як повноцінний програмний сервіс, орієнтований на сприйняття, аналіз і підтримку прийняття технічних рішень.

# 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА СИСТЕМИ МОНІТОРИНГУ

## 3.1 Реалізація програмного агента для збору системних показників

У межах кваліфікаційної роботи реалізовано програмний агент, призначений для безперервного збору системних показників на контрольованих вузлах локальної мережі. Основне призначення цього компонента полягало в тому, щоб забезпечити регулярне отримання актуальної інформації про стан ПК і серверів без потреби у ручному запуску перевірок або безпосередньому втручанні користувача в процес моніторингу. Саме агент став початковою точкою всієї системи, оскільки від стабільності його роботи безпосередньо залежить повнота даних, які надалі передаються на сервер, зберігаються в базі та використовуються для побудови графіків, відображення станів і формування сповіщень.

Під час реалізації агента закладено принцип фонової роботи з мінімальним впливом на ресурси контрольованого вузла. Це рішення виявилося особливо важливим для умов навчальної лабораторії, де на одних і тих самих машинах одночасно виконуються прикладні та навчальні задачі, а тому додаткове програмне навантаження має залишатися якомога менш помітним. Через це агент реалізовано як окремий програмний модуль, який після запуску переходить у циклічний режим функціонування, у межах якого послідовно зчитуються показники системи, формується структурований пакет метрик і виконується підготовка до його подальшого передавання на серверну частину.

У процесі реалізації визначено перелік базових параметрів, які доцільно збирати на кожному вузлі. До таких показників віднесено завантаження центрального процесора, використання оперативної пам’яті, стан дискової підсистеми, параметри мережевої активності та температурні значення апаратних компонентів у тих випадках, коли доступ до них підтримується операційною системою та конфігурацією пристрою. Такий набір показників

виявився достатнім для формування узагальненої та водночас інформативної картини стану вузла, оскільки саме ці параметри найчастіше відображають появу перевантажень, нестачу ресурсів, зростання температури або інші відхилення, що можуть впливати на стабільність роботи обладнання.

Функціональну основу програмного агента побудовано навколо послідовності повторюваних дій, які виконуються через певний часовий інтервал. На кожній ітерації агент ініціює отримання поточних значень системних ресурсів, після чого виконує їх впорядкування у внутрішній структурі. Такий підхід дозволив відмовитися від безперервного опитування без пауз, що створювало б зайве навантаження, і водночас забезпечив достатню оперативність отримання інформації.

Значну увагу під час реалізації приділено стійкості роботи агента в разі короткочасних збоїв. У реальному середовищі не можна гарантувати, що канал передавання даних на сервер буде постійно доступним. Можливе тимчасове відключення мережі, перезапуск серверної частини або локальні збої на самому контрольованому вузлі. Саме тому в агенті реалізовано механізм буферизації, який дозволяє тимчасово накопичувати підготовлені пакети метрик до моменту відновлення зв’язку. Унаслідок цього дані не втрачаються відразу після невдалої спроби передавання, а можуть бути надіслані пізніше. Такий підхід суттєво підвищив практичну надійність усієї системи моніторингу, оскільки зменшив кількість прогалин у часових рядах і зробив агента менш залежним від коротких збоїв зовнішнього середовища.

Передавання даних з агента організовано так, щоб воно залишалося прозорим і контрольованим. Після завершення циклу збору та формування пакета виконується підготовка до надсилання через визначений канал обміну. У межах цієї роботи передбачено підтримку передавання через HTTP API або через MQTT, що дозволило зробити агента придатним до використання в різних мережевих сценаріях. На практичному рівні це означає, що одна й та сама логіка збору системних показників не прив’язується жорстко до єдиного способу

комунікації із сервером. Така гнучкість є важливою, оскільки дозволяє адаптувати розроблене рішення до конкретних умов локальної мережі та особливостей організації інфраструктури. На рисунку 3.1 подано структуру роботи програмного агента збору системних показників.

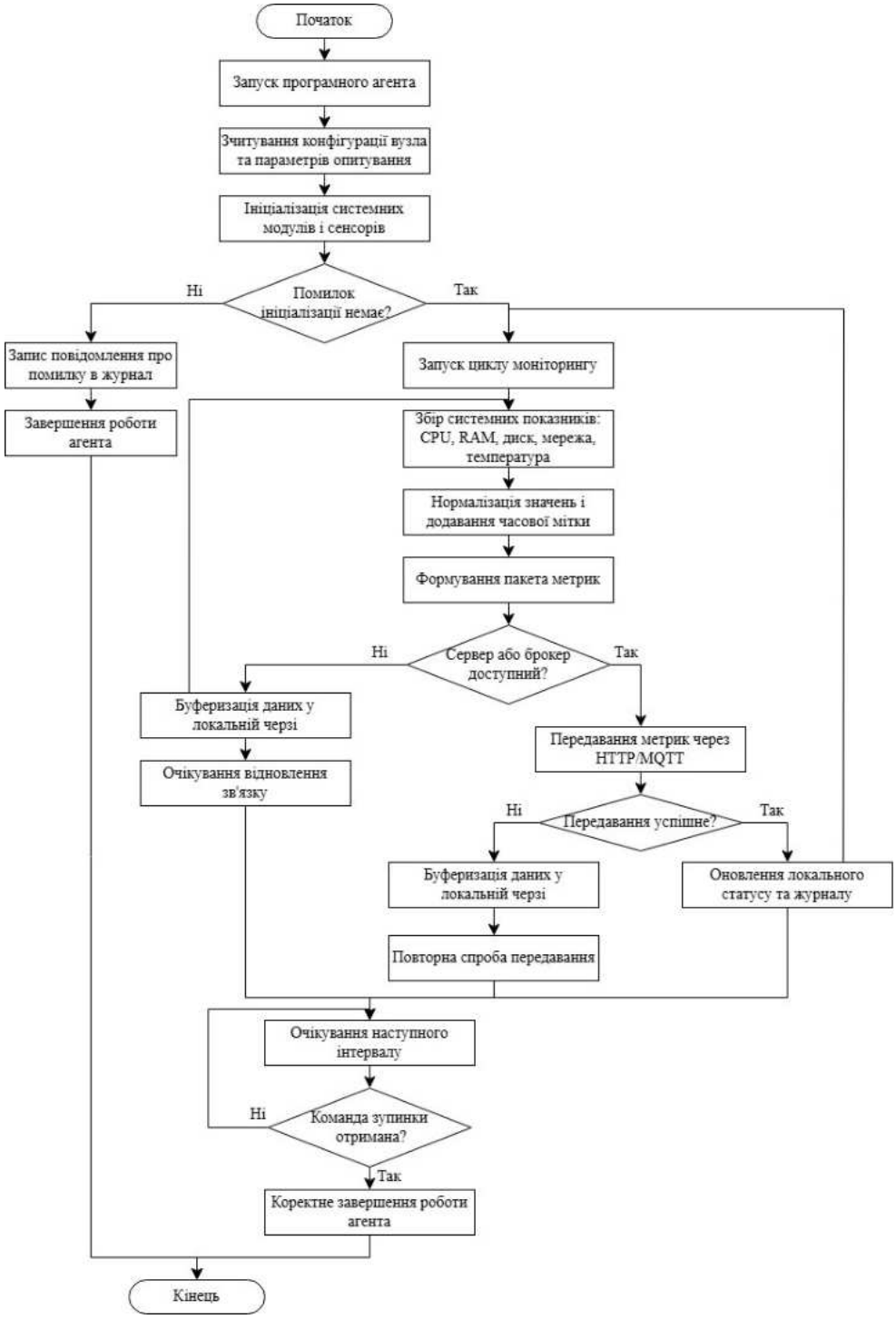



Рисунок 3.1 – Структура роботи програмного агента збору системних показників.

Під час реалізації програмного агента також враховано необхідність підтримки тривалого безперервного функціонування. Агент не розглядався як утиліта для разового запуску, а відразу проєктувався як довготривалий сервіс, здатний працювати у фоновому режимі протягом значного часу. Саме тому логіку його функціонування побудовано так, щоб окремі помилки під час отримання певного показника або тимчасові проблеми з окремим системним джерелом не зупиняли роботу всього модуля. У разі виникнення локальної помилки агент продовжує збір доступних метрик, фіксує службову інформацію про збої та переходить до наступного циклу. Це дозволяє уникнути ситуації, коли моніторинг повністю припиняється через одиничний нестандартний випадок.

Важливою частиною реалізації стало й забезпечення простоти подальшого супроводу коду. Оскільки агент виконує базову, але дуже відповідальну роль, його програмна структура побудована модульно: окремо виділено отримання системних показників, окремо - формування внутрішньої структури метрик, окремо - логіку буферизації та передавання. Такий підхід полегшує подальше розширення функціональності, наприклад у разі потреби додати новий тип показників або змінити спосіб передавання даних. Це має і практичне, і навчальне значення, оскільки розроблене рішення залишається придатним до розвитку без необхідності повного переписування вже реалізованого модуля.

У підсумку в межах цього підрозділу реалізовано програмний агент, який забезпечує автоматизований збір системних показників на контрольованих вузлах локальної мережі, формує уніфіковані пакети метрик і підтримує їх подальше передавання до серверної частини системи. Реалізований підхід поєднує фоновий режим роботи, конфігурований інтервал опитування, підтримку різних груп системних показників, стійкість до короткочасних збоїв і придатність до подальшого розширення. Це створило практичну основу для подальшої реалізації серверного модуля приймання, обробки та зберігання метрик, який розглядається в наступному підрозділі.

## 3.2 Реалізація серверного модуля приймання, обробки та зберігання метрик

У межах кваліфікаційної роботи реалізовано серверний модуль, який забезпечує централізоване приймання, обробку та зберігання метрик, що надходять від програмних агентів із контрольованих вузлів локальної мережі. Саме цей компонент сформував логічне ядро всієї системи моніторингу, оскільки через нього організовано основний потік телеметричних даних, підтримано зв’язок між агентною частиною, сховищами та веб-панеллю, а також забезпечено узгоджену роботу сервісів, що відповідають за накопичення показників і подальше відображення результатів. Якщо агентний модуль виконав роль джерела первинних значень, то серверний рівень уже забезпечив перетворення цих значень на впорядковане інформаційне середовище, придатне для аналізу, візуалізації та формування подій сповіщення.

Під час реалізації серверної частини закладено принцип єдиної точки приймання даних. Такий підхід виявився доцільним з практичного погляду, оскільки дозволив уникнути розосередження логіки обробки між кількома незалежними модулями та сформував чітко визначений вхідний рівень для всіх пакетів, що передаються від агентів. Унаслідок цього незалежно від того, з якого саме вузла надійшов пакет, його подальший шлях залишився однаковим: виконано приймання, перевірку структури, нормалізацію, розподіл даних між відповідними сховищами та оновлення службового стану вузла. Така організація стала особливо важливою в умовах локальної мережі навчальної лабораторії, де одночасно функціонують машини різного призначення, а частота надсилання пакетів може змінюватися залежно від ролі конкретного вузла та поточного режиму навантаження.

Разом із розбором метрик реалізовано механізм оновлення службового стану вузла. Кожне нове надходження пакета дозволило не лише записати поточні значення показників, а й зафіксувати факт актуальної активності контрольованої машини. Саме тому у серверному модулі оновлено час

останнього зв’язку з вузлом, що далі використовується для визначення його поточного статусу у веб-панелі. Таке рішення має практичну цінність, оскільки відсутність нових пакетів упродовж певного часу вже розглядається як окремий індикатор можливих проблем, пов’язаних із мережею, вимкненням вузла або збоєм у роботі агента.

Під час реалізації серверного модуля особливу увагу приділено розділенню телеметричних та конфігураційних даних. У межах цієї роботи реалізовано окреме сховище для метрик і окреме сховище для службової інформації. Телеметричні дані, що мають характер часових рядів і постійно поповнюються новими записами, збережено в InfluxDB. Таке рішення виявилося найбільш придатним для системи моніторингу, оскільки дозволило ефективно працювати з послідовностями значень, побудованими у часовому вимірі, виконувати вибірки за інтервалами та готувати дані для побудови графіків. Натомість службові й конфігураційні записи, до яких належать відомості про вузли, правила алертів, параметри доступу, ролі та допоміжні журнали, збережено в SQLite. Це дозволило зберегти логічний порядок у структурі системи та не змішувати дані різної природи в межах одного сховища.

На рисунку 3.2 подано структуру роботи серверного модуля приймання, обробки та зберігання метрик.

Після перевірки та нормалізації пакетів реалізовано запис показників у сховище часових рядів. Кожен запис збережено разом із часовою міткою, типом метрики, ідентифікатором вузла та необхідними тегами, що використовуються для подальшої фільтрації й побудови вибірок. Такий підхід дозволив не лише накопичувати поточні значення, а й формувати повну історію зміни стану вузлів у часі. У результаті серверна частина забезпечила можливість подальшого аналізу динаміки навантаження, виявлення пікових станів, побудови статистичних зрізів і відображення тенденцій у роботі як окремих машин, так і всієї локальної мережі загалом. На рисунку 3.2 подано структуру роботи серверного модуля приймання, обробки та зберігання метрик.

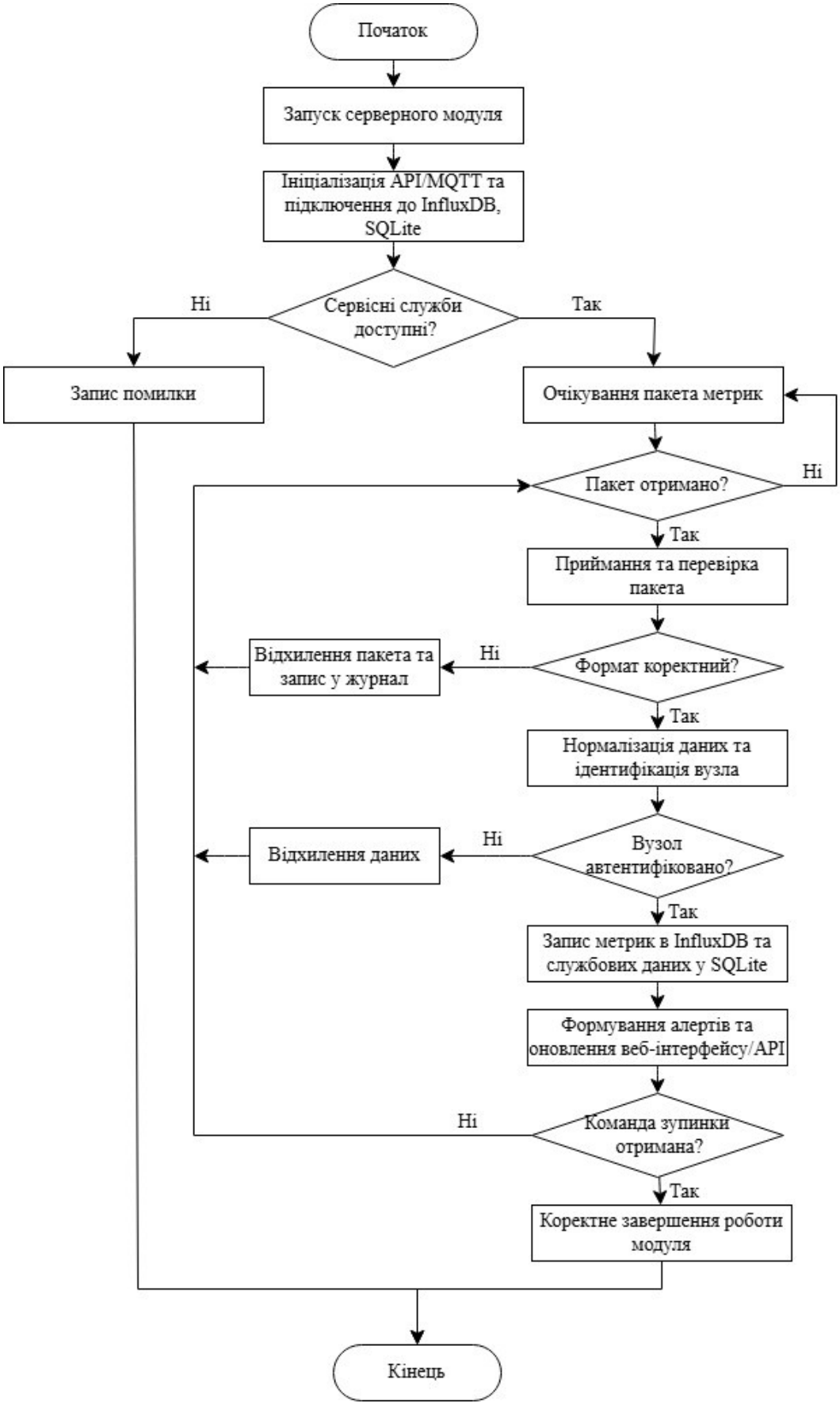



Рисунок 3.2 – Структура роботи серверного модуля приймання, обробки та зберігання метрик.

Паралельно зі збереженням метрик реалізовано роботу з конфігураційною базою, у межах якої обслуговуються записи про самі вузли, параметри системи, ролі користувачів, умови сповіщень та службові події. Такий рівень логіки виявився не менш важливим, ніж запис телеметрії, оскільки саме він забезпечив функціонування всієї сервісної інфраструктури. Без нього неможливо було б визначати, до якої групи належить вузол, які порогові значення застосовуються до конкретного типу метрик, хто має доступ до перегляду або адміністрування веб-панелі, а також які події мають відображатися як критичні.

Окремим напрямом реалізації став механізм перевірки умов для формування алертів. На цьому рівні серверний модуль виконав не лише функцію приймання й запису даних, а й роль активного інтерпретатора стану системи. Отримані показники співставлено з правилами, заданими в конфігурації, після чого у разі перевищення порогових значень або виявлення інших небажаних ситуацій сформовано відповідні події. Наприклад, серверний модуль фіксує тривале перевантаження процесора, критичне зростання температури, нестачу вільного місця на диску або тривалу відсутність метрик від конкретного вузла. Унаслідок цього система переходить від простого накопичення числових значень до реального контролю стану інфраструктури.

Під час реалізації також враховано потребу в підтримці стабільної роботи серверного модуля за умов одночасного надходження пакетів від кількох вузлів. Навіть якщо в навчальній лабораторії кількість машин є відносно помірною, серверна частина повинна залишатися придатною до ситуації, коли кілька агентів майже одночасно передають нові порції даних. Саме тому внутрішню логіку побудовано як послідовність чітко розмежованих етапів: приймання, перевірка, обробка, запис, оновлення стану вузла та аналіз умов для алертів. Це дозволило зробити роботу сервера більш передбачуваною, а також спростило супровід і діагностику можливих проблем у його функціонуванні.

У межах серверного модуля реалізовано також журналювання службових подій. Це означає, що система фіксує не лише надходження коректних пакетів, а

й випадки помилок структури, збоїв під час запису, порушення правил доступу або інших нестандартних ситуацій. Такий підхід має прикладне значення, оскільки дозволяє оцінювати не тільки стан контрольованих вузлів, а й стабільність роботи самої серверної частини. У результаті серверний модуль набуває ознак повноцінного програмного компонента з внутрішньою логікою самоконтролю, а не просто пасивного каналу передавання інформації до сховища.

Важливою рисою реалізації стало модульне розділення внутрішньої логіки серверного рівня. Окремо виділено блок приймання пакетів, окремо – перевірку та нормалізацію, окремо – запис у сховище часових рядів, окремо – роботу зі службовою базою, а також окремо – перевірку правил алертів. Таке рішення спростило супровід коду, підвищило прозорість його структури та створило кращі умови для подальшого розширення функціональності. У разі потреби змінити формат пакета, модифікувати логіку перевірки або додати новий тип подій не виникає потреби перебудовувати весь серверний модуль повністю, оскільки зміни можуть бути локалізовані в межах конкретного функціонального блоку.

## 3.3 Реалізація веб-інтерфейсу системи моніторингу та засобів відображення результатів

Реалізовано веб-інтерфейс системи моніторингу, який забезпечує зручний доступ до зібраних метрик, станів контрольованих вузлів і результатів аналітичної обробки даних. Саме цей компонент сформував користувацький рівень усієї системи, оскільки через нього забезпечено перегляд поточних показників, аналіз історії змін, контроль доступності вузлів і сприйняття сповіщень про відхилення в роботі інфраструктури. Якщо агентна частина забезпечила отримання даних, а серверний модуль виконав їх приймання,

перевірку та зберігання, то веб-інтерфейс уже перетворив ці дані на наочне й зручне для сприйняття представлення.

Під час реалізації веб-інтерфейсу закладено принцип централізованого доступу до результатів моніторингу через браузер. Такий підхід виявився найбільш доцільним для умов локальної мережі навчальної лабораторії, оскільки не потребує встановлення окремого клієнтського програмного забезпечення на кожну машину, спрощує розгортання та забезпечує швидкий доступ до системи з різних робочих місць. Унаслідок цього користувацьку частину побудовано як веб-панель, що взаємодіє із серверним API та отримує від нього вже підготовлені вибірки даних для відображення.

У процесі розроблення веб-інтерфейсу визначено, що його структура повинна забезпечувати два основні режими роботи: загальний огляд усієї інфраструктури та детальний перегляд окремого вузла. Саме тому початкову сторінку реалізовано як інформаційну панель, на якій відображаються всі контрольовані ПК і сервери з короткими відомостями про їхній поточний стан. На цьому рівні користувач отримує можливість швидко побачити, які вузли є активними, коли саме від них востаннє надходили метрики, а також які з них працюють у нормальному режимі, а які вже демонструють ознаки перевантаження або втрати доступності. Така організація інтерфейсу дозволила перейти від фрагментарного перегляду окремих показників до цілісного сприйняття стану всієї локальної мережі.

Для кожного вузла в межах веб-інтерфейсу реалізовано окремий режим деталізації. Після вибору конкретного ПК або сервера користувач переходить до сторінки, де подано розширений набір показників за основними категоріями. Окремо відображено блоки, пов’язані з процесором, оперативною пам’яттю, дисковою підсистемою, мережею та температурними параметрами. Такий поділ дозволив зробити інтерфейс більш впорядкованим і зрозумілим, оскільки дані не подаються суцільним масивом чисел, а розміщуються у логічно відокремлених

зонах. У результаті користувач не лише бачить набір поточних значень, а й швидше орієнтується в тому, який саме ресурс потребує уваги.

Особливу увагу під час реалізації приділено засобам візуалізації результатів, оскільки саме вони формують практичну цінність системи моніторингу. У веб-інтерфейсі реалізовано графічне відображення часових рядів, що дозволяє аналізувати динаміку змін системних параметрів, а не лише сприймати окремі поточні значення. Такий підхід є принципово важливим для теми цієї роботи, оскільки навантаження на процесор, зміна використання оперативної пам’яті, заповнення диска чи зростання температури значно інформативніше оцінюються саме в часовому розрізі. Через це в інтерфейсі реалізовано графіки, які дають змогу простежити зміну стану вузла за обраний проміжок часу та побачити не тільки сам факт відхилення, а й момент його виникнення, тривалість і характер розвитку.

Для відображення результатів реалізовано поєднання графічного та табличного подання. Графіки використовуються для демонстрації динаміки, тоді як таблиці забезпечують перегляд точних значень, переліку подій, часових позначок і коротких службових відомостей про вузли. Така комбінація виявилася найбільш зручною, оскільки графіки добре показують загальну тенденцію, а табличні блоки дозволяють уточнити конкретні числові значення без потреби в додатковому аналізі осей або шкал. Унаслідок цього веб-інтерфейс забезпечив не лише візуальну наочність, а й достатню точність подання результатів.

У процесі реалізації інтерфейсу передбачено можливість швидкого вибору періоду спостереження. Це рішення виявилося важливим з практичного погляду, оскільки одна й та сама метрика може мати різний зміст залежно від часової глибини перегляду. Для коротких проміжків часу зручним є перегляд оперативних змін, що відображають поточне навантаження на вузол. Для довших інтервалів більшу цінність має виявлення тенденцій, повторюваних піків або поступового наближення системи до критичного стану. Саме тому в інтерфейсі реалізовано вибір типових часових діапазонів, що дозволяє швидко

переходити від короткострокового до довгострокового аналізу без ручного формування складних запитів.

На рисунку 3.3 подано структуру веб-інтерфейсу системи моніторингу та засобів відображення результатів.

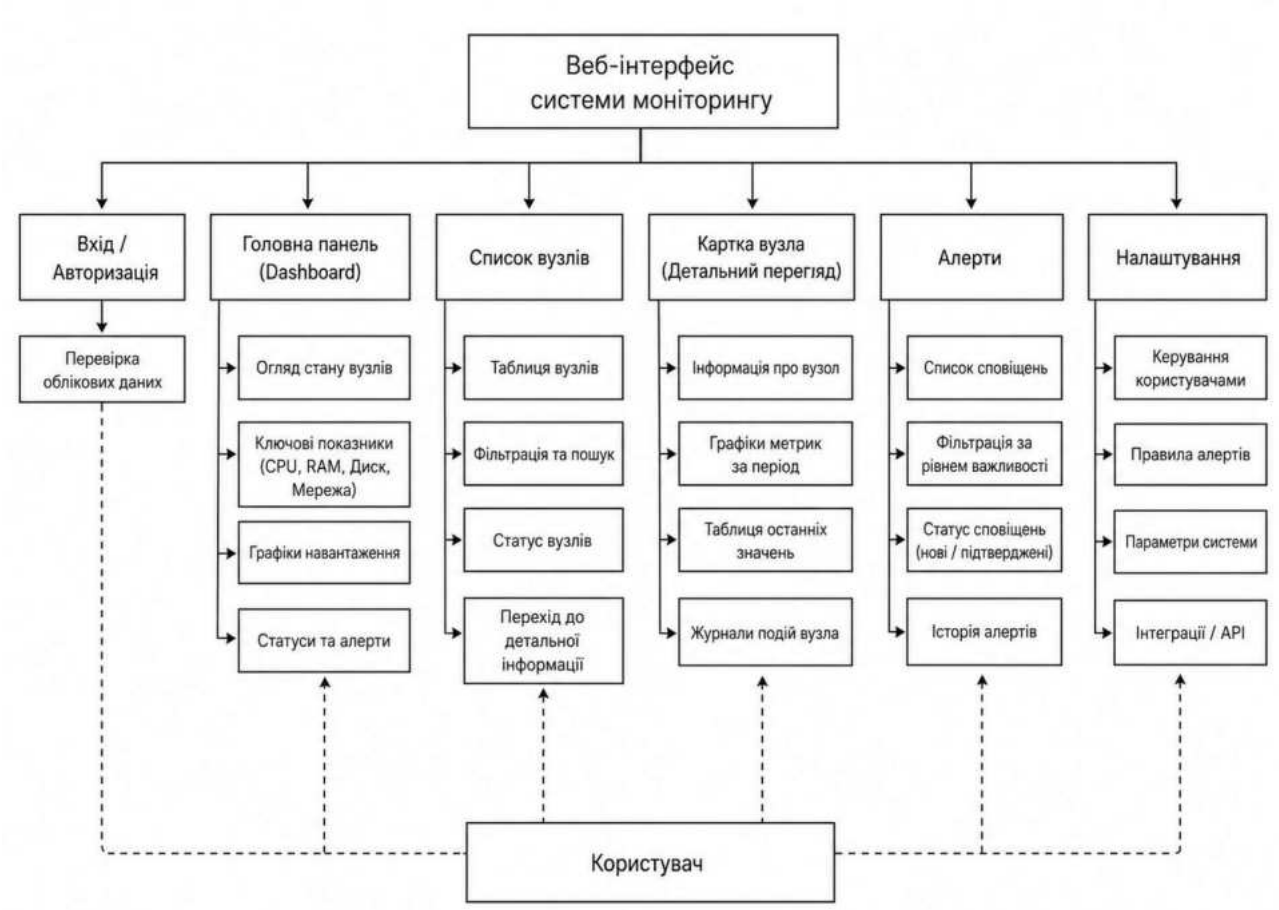



Рисунок 3.3 – Структура веб-інтерфейсу системи моніторингу та засобів відображення результатів

Окремою частиною веб-інтерфейсу реалізовано блок подій і сповіщень. У цьому блоці відображаються результати перевірки правил алертів, що були сформовані серверним модулем на основі отриманих метрик. Такий компонент став важливим доповненням до графіків і таблиць, оскільки він дозволяє не лише спостерігати за показниками, а й одразу бачити, які саме стани були розпізнані системою як небажані або потенційно критичні. У межах цього блоку подано

назву вузла, тип проблеми, рівень критичності, час спрацювання та поточний статус події. Це дає змогу значно швидше орієнтуватися в проблемних ситуаціях і зменшує потребу в ручному аналізі великої кількості окремих значень.

Під час реалізації взаємодії з користувачем враховано і те, що система повинна залишатися зручною не лише за наявності кількох вузлів, а й у разі поступового розширення інфраструктури. Саме тому у веб-інтерфейсі реалізовано базові механізми впорядкування та фільтрації інформації. Це дозволило швидко знаходити потрібний вузол, відокремлювати сервери від робочих станцій, переглядати тільки активні або тільки проблемні машини, а також отримувати більш структуровану картину поточного стану мережі. Такий підхід підвищив практичну придатність інтерфейсу до реального використання, оскільки без фільтрації навіть наочна панель із часом перетворюється на перевантажений інформаційний екран.

Окрему увагу приділено і тому, щоб засоби відображення результатів залишалися придатними до роботи з великими масивами даних. Оскільки система накопичує історію метрик у часовому розрізі, кількість точок на графіках з часом зростає. Саме тому в реалізації інтерфейсу враховано потребу в адаптивному відображенні часових рядів залежно від обраного проміжку спостереження. Це дозволило зберегти читабельність графіків та уникнути ситуації, коли інтерфейс стає перевантаженим через надмірну деталізацію. Унаслідок цього результати моніторингу залишаються зручними для аналізу як у короткому часовому вікні, так і при перегляді довготривалої історії показників.

## 3.4 Тестування системи моніторингу в умовах експериментальної експлуатації

Виконано тестування системи моніторингу в умовах експериментальної експлуатації з метою перевірки стабільності функціонування всіх реалізованих компонентів, правильності передавання метрик, коректності їх збереження,

відображення у веб-інтерфейсі та спрацювання механізмів сповіщення. Цей етап виявився необхідним, оскільки саме практична перевірка дозволяє не лише підтвердити працездатність окремих модулів, а й оцінити, наскільки узгоджено система функціонує як єдиний сервіс у реальному середовищі локальної мережі. Якщо у попередніх підрозділах реалізовано агент збору метрик, серверний модуль і веб-панель, то на цьому етапі вже перевірено, як усі ці складові поводяться під час спільної роботи в умовах наближених до реальної експлуатації.

Під час організації тестування виходили з того, що система повинна не лише запускатися без помилок, а й забезпечувати стійке функціонування впродовж тривалого часу, реагувати на зміну стану контрольованих вузлів, фіксувати навантаження на ресурси та відображати результати без істотних затримок. Саме тому експериментальну перевірку побудовано не як разове відкриття веб-панелі чи одиничне надсилання тестового пакета, а як послідовність сценаріїв, у межах яких оцінено поведінку системи за різних умов роботи. До таких умов віднесено нормальний режим функціонування, зростання навантаження на окремі ресурси, переривання зв’язку між агентом і сервером, повторне відновлення передавання даних, зміну статусу вузлів та спрацювання правил алертів.

Паралельно перевірено роботу системи в умовах зміни параметрів дискової підсистеми та мережевої активності. Для цього змодельовано сценарії запису й читання даних, а також мережевого обміну, після чого простежено коректність зчитування відповідних показників і їх відображення в інтерфейсі моніторингу. У результаті підтверджено, що система коректно фіксує динаміку використання дискових ресурсів, мережевий трафік та зміну навантаження на інтерфейси, що дозволяє використовувати її не лише для контролю процесора й пам’яті, а й для більш комплексного спостереження за станом вузлів.

У процесі тестування окремо перевірено обробку температурних показників на тих вузлах, де відповідні дані є доступними. На практичному рівні

це дозволило оцінити, наскільки система придатна до виявлення небажаного теплового режиму роботи обладнання. У результаті підтверджено, що в разі наявності доступу до сенсорних джерел агент коректно включає температурні значення до пакета метрик, серверна частина зберігає ці показники у часових рядах, а веб-панель відображає їх у межах відповідного блоку. Це суттєво розширює прикладну корисність системи, оскільки дозволяє спостерігати не лише за логічним навантаженням ресурсів, а й за фізичними умовами роботи апаратної складової.

На рисунку 3.4 подано схему тестування системи моніторингу в умовах експериментальної експлуатації.

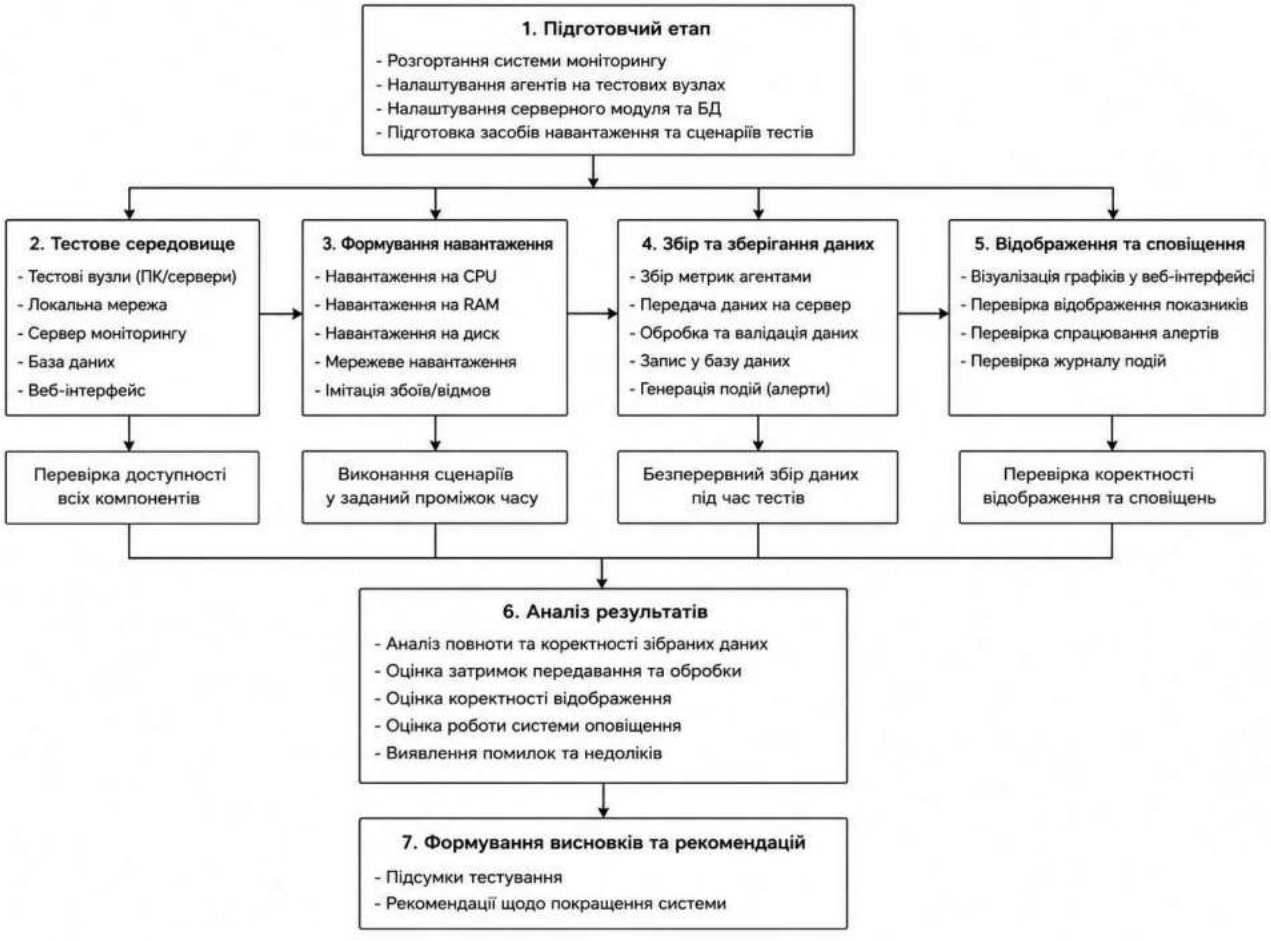



Рисунок 3.4 – Схема тестування системи моніторингу в умовах експериментальної експлуатації

Важливим етапом перевірки стало тестування системи в умовах тимчасового порушення зв’язку між агентом і серверною частиною. Для цього змодельовано ситуацію, у якій під час чергового циклу передавання агент не має змоги негайно доставити сформований пакет метрик на сервер. У такому режимі перевірено роботу механізму буферизації, який раніше реалізовано на рівні агентного модуля. У результаті підтверджено, що за короткочасної недоступності сервера або каналу зв’язку дані не втрачаються відразу, а тимчасово накопичуються на стороні вузла. Після відновлення з’єднання передавання було продовжено, а буферизовані пакети успішно доставлено до серверної частини. Це дозволяє зробити висновок, що система залишається функціонально стійкою навіть у разі коротких збоїв мережі.

Під час тестування також перевірено роботу механізму алертів. Для цього змодельовано ситуації, у яких певні показники досягають або перевищують задані порогові значення. У межах цих сценаріїв простежено, чи формує серверна частина відповідні події, чи відображаються вони у веб-панелі та чи відповідає їхній статус фактичному стану контрольованого вузла. У результаті показано, що система коректно виявляє перевищення порогів, фіксує події в журналі та відображає їх як окремі сигнали, придатні для подальшого аналізу. Це підтверджує, що реалізований сервіс виконує не лише функцію пасивного накопичення даних, а й завдання активного інформування про потенційно небезпечні відхилення.

Додатково перевірено, наскільки зручно сприймаються результати тестування через веб-інтерфейс. Для цього проаналізовано роботу сторінки загального огляду, сторінки окремого вузла, блоків графіків, таблиць і журналу подій. У результаті підтверджено, що реалізовані засоби візуалізації дають змогу швидко виявляти зміни в роботі системи, переходити від загальної картини до деталізації конкретного ресурсу та зіставляти події з часовою динамікою показників. Це важливо з практичного погляду, оскільки будь-яка система

моніторингу набуває реальної цінності лише тоді, коли зібрані нею дані легко сприймаються й інтерпретуються без додаткової ручної обробки.

У межах експериментальної експлуатації простежено і загальну стабільність взаємодії між компонентами системи. Зокрема, оцінено, чи не виникають критичні конфлікти між агентами, серверним модулем, сховищами даних і веб-панеллю, чи не спостерігаються суттєві затримки між фактичним зчитуванням показників і їх появою в інтерфейсі, а також чи не порушується цілісність даних за тривалішої роботи системи. У результаті підтверджено, що реалізована архітектура підтримує узгоджене функціонування всіх рівнів і не виявляє принципових проблем у межах експериментального середовища.

Ще одним важливим результатом тестування стало підтвердження придатності системи до подальшого масштабування. Хоча перевірка виконувалася в умовах експериментальної експлуатації з обмеженою кількістю вузлів, сама логіка функціонування сервісу показала, що додавання нових агентів не потребує перебудови принципу роботи серверної частини чи веб-панелі. Це дозволяє розглядати розроблене рішення не лише як демонстраційний прототип, а як основу для подальшого впровадження в реальніші або більш розгалужені локальні мережі.

## 3.5 Висновки до третього розділу

У третьому розділі кваліфікаційної роботи реалізовано основні програмні компоненти системи моніторингу стану ПК і серверів у локальній мережі та виконано їх експериментальну перевірку в умовах, наближених до реальної експлуатації. У межах практичної частини реалізовано програмний агент збору системних показників, серверний модуль приймання, обробки та зберігання метрик, а також веб-інтерфейс, призначений для централізованого перегляду результатів моніторингу. Унаслідок цього сформовано завершений цикл роботи системи, у межах якого показники стану вузлів зчитуються, передаються,

накопичуються, аналізуються та відображаються у зручній для сприйняття формі. У процесі реалізації агента забезпечено автоматизований збір базових системних показників, до яких віднесено завантаження процесора, використання оперативної пам’яті, параметри дискової підсистеми, мережеву активність і температурні значення, якщо вони доступні на конкретному вузлі. Агентний модуль побудовано з урахуванням фонової роботи, циклічного режиму функціонування, формування уніфікованого пакета метрик і підтримки буферизації в разі короткочасних збоїв зв’язку. Це дозволило створити стійкий механізм збирання телеметрії, придатний до використання в межах навчальної локальної мережі без суттєвого впливу на продуктивність контрольованих машин. У межах користувацького рівня реалізовано веб-інтерфейс, який забезпечив зведений перегляд усіх контрольованих вузлів, деталізацію стану окремого ПК або сервера, відображення графіків зміни системних параметрів, перегляд таблиць подій і сприйняття алертів. Такий підхід дозволив перетворити накопичені дані на практичний інструмент контролю інфраструктури, придатний до щоденного використання. За рахунок графічного подання часових рядів та відображення поточних статусів забезпечено не лише перегляд окремих значень, а й аналіз тенденцій зміни навантаження, що є важливим для своєчасного виявлення нестабільних або проблемних режимів роботи.

Під час експериментальної перевірки підтверджено працездатність усіх основних складових системи. Встановлено, що агент коректно зчитує системні показники, серверна частина приймає та обробляє пакети без порушення структури, а веб-інтерфейс відображає поточні й історичні дані у зручному вигляді. Окремо підтверджено, що система фіксує зміну навантаження на ресурси, відображає втрату доступності вузла, підтримує буферизацію при тимчасових збоях каналу передавання та формує події при перевищенні заданих порогових значень. Це свідчить про те, що реалізований сервіс виконує не лише функцію спостереження, а й забезпечує активний контроль стану інфраструктури.

# ВИСНОВКИ

У кваліфікаційній роботі розглянуто актуальну задачу створення системи моніторингу стану ПК і серверів у локальній мережі, орієнтованої на використання в умовах навчальної лабораторії. У процесі виконання роботи проаналізовано особливості експлуатації комп'ютерної інфраструктури, для якої характерними є змінні режими навантаження, різнорідність апаратної бази, висока залежність стабільності роботи від своєчасного виявлення відхилень і обмежені можливості ручного контролю. Проведений аналіз предметної області показав, що традиційний підхід до перевірки стану вузлів не забезпечує достатньої оперативності, не дозволяє накопичувати історію показників і ускладнює пошук причин нестабільної роботи окремих машин або всієї інфраструктури загалом.

У першому розділі систематизовано основні теоретичні положення, що стосуються моніторингу обчислювальної інфраструктури, розкрито роль таких систем у підтримці працездатності ПК і серверів, а також проаналізовано існуючі підходи й готові рішення у цій сфері. Показано, що сучасні платформи моніторингу мають значні функціональні можливості, проте для умов навчальної лабораторії доцільним виявляється створення власного сервісу, який поєднує простоту розгортання, достатню функціональність, централізований збір метрик, збереження часових рядів, веб-візуалізацію та підтримку сповіщень. На цій основі сформульовано постановку задачі, що передбачає розроблення агента збору системних показників, серверного модуля приймання та обробки даних, сховища метрик, веб-панелі та механізму виявлення критичних станів.

У другому розділі спроєктовано загальну архітектуру системи моніторингу та обґрунтовано вибір технічних і програмних засобів реалізації. Архітектуру побудовано за розподіленим принципом, де на контрольованих вузлах функціонує агент збору метрик, а в центрі системи працює серверна частина, що відповідає за приймання, перевірку, збереження, аналіз та подальше

відображення даних. Для реалізації агента і серверного модуля обрано Python, для збору системних показників - бібліотеку psutil, для серверної взаємодії - FastAPI, для передавання даних - HTTP API та MQTT, для збереження часових рядів - InfluxDB, для конфігураційних і службових записів - SQLite, а для веб-інтерфейсу - React і засоби побудови графіків. Окремо розроблено логічну модель даних, у межах якої розмежовано метрики, службову інформацію, вузли, правила алертів і події. Таке проєктне рішення сформувало технічну основу для створення цілісної системи моніторингу, придатної до практичного впровадження.

У третьому розділі реалізовано основні компоненти розроблюваної системи та виконано їх експериментальну перевірку. Реалізовано програмний агент, який у фоновому режимі виконує циклічний збір показників процесора, оперативної пам’яті, дискової підсистеми, мережі та температурних параметрів, формує уніфікований пакет даних і забезпечує його подальше передавання на сервер. Реалізовано серверний модуль, який приймає пакети, перевіряє їх структуру, нормалізує показники, записує метрики у сховище часових рядів, оновлює стан вузлів і формує події для системи сповіщень. Також реалізовано веб-інтерфейс, що забезпечує централізований перегляд усіх контрольованих вузлів, відображення історії метрик, аналіз графіків, перегляд таблиць подій і сприйняття алертів у зручній формі.

У підсумку поставлену мету кваліфікаційної роботи досягнуто. Розроблено та перевірено систему моніторингу стану ПК і серверів у локальній мережі, яка забезпечує автоматизований збір системних показників, їх надійне передавання, централізоване збереження, зручну візуалізацію та своєчасне виявлення відхилень у роботі контрольованих вузлів. Отриманий результат підтверджує доцільність використання такого підходу в умовах навчальної інфраструктури та засвідчує практичну обґрунтованість запропонованого програмного рішення.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ


1. Zabbix. Templates out of the box. Zabbix Documentation. 2026. URL: https://www.zabbix.com/documentation/devel/en/manual/config/templates_out_of_the_box (дата звернення: 29.05.2026).
2. Checkmk. The Official Checkmk User Guide. Checkmk Documentation. 2026. URL: https://docs.checkmk.com/latest/en/ (дата звернення: 29.05.2026).
3. Paessler. PRTG Manual. Paessler PRTG Documentation. 2026. URL: https://www.paessler.com/manuals/prtg (дата звернення: 29.05.2026).
4. Netdata. Netdata Agent Installation. Netdata Documentation. 2026. URL: https://learn.netdata.cloud/docs/netdata-agent/installation (дата звернення: 29.05.2026).
5. LibreNMS. LibreNMS Documentation. LibreNMS Docs. 2026. URL: https://docs.librenms.org/ (дата звернення: 29.05.2026).
6. Prometheus Authors. Prometheus Documentation. Prometheus Documentation. 2026. URL: https://prometheus.io/docs/ (дата звернення: 29.05.2026).
7. Prometheus Authors. Monitoring Linux Host Metrics with the Node Exporter. Prometheus Documentation. 2026. URL: https://prometheus.io/docs/guides/node-exporter/ (дата звернення: 29.05.2026).
8. Prometheus Authors. Node Exporter. GitHub. 2026. URL: https://github.com/prometheus/node_exporter (дата звернення: 29.05.2026).
9. Grafana Labs. Dashboards. Grafana Documentation. 2026. URL: https://grafana.com/docs/grafana/latest/visualizations/dashboards/ (дата звернення: 29.05.2026).
10. Grafana Labs. Data Sources. Grafana Documentation. 2026. URL: https://grafana.com/docs/grafana/latest/datasources/ (дата звернення: 29.05.2026).
11. InfluxData. InfluxDB Documentation. InfluxDB Documentation. 2026. URL: https://docs.influxdata.com/influxdb/ (дата звернення: 29.05.2026).

12. InfluxData. Telegraf Documentation. InfluxData Documentation. 2026. URL: https://docs.influxdata.com/telegraf/v1/ (дата звернення: 29.05.2026).

13. InfluxData. Telegraf: Open Source Server Agent for Collecting Metrics. InfluxData. 2026. URL: https://www.influxdata.com/time-series-platform/telegraf/ (дата звернення: 29.05.2026).

14. Giampaolo R. psutil Documentation. psutil Documentation. 2026. URL: https://psutil.readthedocs.io/ (дата звернення: 29.05.2026).

15. Python Software Foundation. Python 3 Documentation. Python Documentation. 2026. URL: https://docs.python.org/3/ (дата звернення: 29.05.2026).

16. Python Software Foundation. sqlite3 — DB-API 2.0 Interface for SQLite Databases. Python Documentation. 2026. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 29.05.2026).

17. Tiangolo S. FastAPI Documentation. FastAPI. 2026. URL: https://fastapi.tiangolo.com/ (дата звернення: 29.05.2026).

18. Pydantic Team. Pydantic Documentation. Pydantic Documentation. 2026. URL: https://docs.pydantic.dev/ (дата звернення: 29.05.2026).

19. Encode. Uvicorn Documentation. Uvicorn Documentation. 2026. URL: https://www.uvicorn.org/ (дата звернення: 29.05.2026).

20. SQLite Consortium. SQLite Documentation. SQLite. 2026. URL: https://sqlite.org/docs.html (дата звернення: 29.05.2026).

21. React Team. React Documentation. React. 2026. URL: https://react.dev/ (дата звернення: 29.05.2026).

22. Docker Inc. Docker Documentation. Docker Docs. 2026. URL: https://docs.docker.com/ (дата звернення: 29.05.2026).

23. Docker Inc. Docker Compose. Docker Docs. 2026. URL: https://docs.docker.com/compose/ (дата звернення: 29.05.2026).

24. Docker Inc. Compose File Reference. Docker Docs. 2026. URL: https://docs.docker.com/reference/compose-file/ (дата звернення: 29.05.2026).

25. Eclipse Foundation. Eclipse Mosquitto. Eclipse Mosquitto. 2026. URL: https://mosquitto.org/ (дата звернення: 29.05.2026).

26. Eclipse Foundation. mosquitto.conf Man Page. Eclipse Mosquitto Documentation. 2026. URL: https://mosquitto.org/man/mosquitto-conf-5.html (дата звернення: 29.05.2026).

27. OASIS. MQTT Version 5.0. OASIS Standard. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (дата звернення: 29.05.2026).

28. OpenTelemetry Authors. OpenTelemetry Specification. OpenTelemetry Documentation. 2026. URL: https://opentelemetry.io/docs/specs/otel/ (дата звернення: 29.05.2026).

29. OpenAPI Initiative. OpenAPI Specification Version 3.1.1. OpenAPI Specification. 2024. URL: https://spec.openapis.org/oas/v3.1.1.html (дата звернення: 29.05.2026).

30. JSON Schema. JSON Schema Validation: A Vocabulary for Structural Validation of JSON. JSON Schema. 2020. URL: https://json-schema.org/draft/2020-12/json-schema-validation (дата звернення: 29.05.2026).

31. OWASP Foundation. OWASP API Security Top 10 – 2023. OWASP. 2023. URL: https://owasp.org/API-Security/editions/2023/en/0x00-header/ (дата звернення: 29.05.2026).

32. OWASP Foundation. Logging Cheat Sheet. OWASP Cheat Sheet Series. 2026. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html (дата звернення: 29.05.2026).

33. Fielding R., Nottingham M., Reschke J. HTTP Semantics. *RFC 9110*. 2022. DOI: 10.17487/RFC9110

34. Fielding R., Nottingham M., Reschke J. HTTP/1.1. *RFC 9112*. 2022. DOI: 10.17487/RFC9112

35. Thomson M., Nottingham M., Pauly T. Building Protocols with HTTP. *RFC 9205*. 2022. DOI: 10.17487/RFC9205

36. Nottingham M., Wilde E., Dalal S. Problem Details for HTTP APIs. *RFC 9457*. 2023. DOI: 10.17487/RFC9457

37. Souppaya M., Scarfone K., Dodson D. Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities. *NIST Special Publication 800-218*. 2022. DOI: 10.6028/NIST.SP.800-218

38. National Institute of Standards and Technology. The NIST Cybersecurity Framework (CSF) 2.0. *NIST Cybersecurity White Paper 29*. 2024. DOI: 10.6028/NIST.CSWP.29

39. ISO/IEC. ISO/IEC 25010:2023 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Product Quality Model. International Organization for Standardization. 2023. URL: https://www.iso.org/standard/78176.html (дата звернення: 29.05.2026).

40. Stetsenko I., Myroniuk A. Software for Collecting and Analyzing Metrics in Highly Loaded Applications Based on the Prometheus Monitoring System. *Information, Computing and Intelligent Systems*. 2024. No. 5. DOI: 10.20535/2786-8729.5.2024.316366

41. Parvathy M., Antony Balasingam J., Sanjith E. S. Real-Time Web Server Monitoring System Using Python. *Journal of Artificial Intelligence and Capsule Networks*. 2024. Vol. 6, No. 3. P. 332–339. DOI: 10.36548/jaicn.2024.3.006

42. Saputra M. Y. E., Noprianto, Noor Arief S., Wijayaningrum V. N., Syaifudin Y. W. Real-Time Server Monitoring and Notification System with Prometheus, Grafana, and Telegram Integration. *2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS)*. 2024. P. 1808–1813. DOI: 10.1109/ICETSIS61505.2024.10459488

43. Lobur M., Maliar M. Selecting a Monitoring Technology for a Control System of Distributed Oil Production Facilities. *Energy Engineering and Control Systems*. 2024. Vol. 10, No. 1. P. 28–34. DOI: 10.23939/jeecs2024.01.028

44. Skorin Y., Zolotaryova I., Lystopad Y. The Management of Scalability in Cloud-Based Applications. *Computer Systems and Information Technologies*. 2024. No. 3. P. 58–66. DOI: 10.31891/csit-2024-3-8

45. Rak T. Performance Evaluation of an API Stock Exchange Web System on Cloud Docker Containers. *Applied Sciences*. 2023. Vol. 13, No. 17. 9896. DOI: 10.3390/app13179896

46. Liu X., Wei Z., Yu W., Liu S., Wang G., Liu X., Li Y. Khronos: A Real-Time Indexing Framework for Time Series Databases on Large-Scale Performance Monitoring Systems. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2023. P. 1607–1616. DOI: 10.1145/3583780.3614944

47. Chan N. A Resource Utilization Analytics Platform Using Grafana and Telegraf for the Savio Supercluster. *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines*. 2019. P. 1–6. DOI: 10.1145/3332186.3333053

48. Bello A. W. Designing a Real-Time Monitoring System for the AWS Cloud. CEUR Workshop Proceedings. 2025. URL: https://ceur-ws.org/Vol-4036/Paper9.pdf (дата звернення: 29.05.2026).

49. Choudhury S. Cloud-Based Observability in Distributed Systems. KTH Royal Institute of Technology. 2025. URL: https://www.diva-portal.org/smash/get/diva2%3A1964791/FULLTEXT02.pdf (дата звернення: 29.05.2026).

50. Elradi M. D. Prometheus & Grafana: A Metrics-focused Monitoring Stack. *Journal of Computer Allied Intelligence*. 2025. Vol. 3, No. 3. P. 28–39. DOI: 10.69996/jcai.2025015

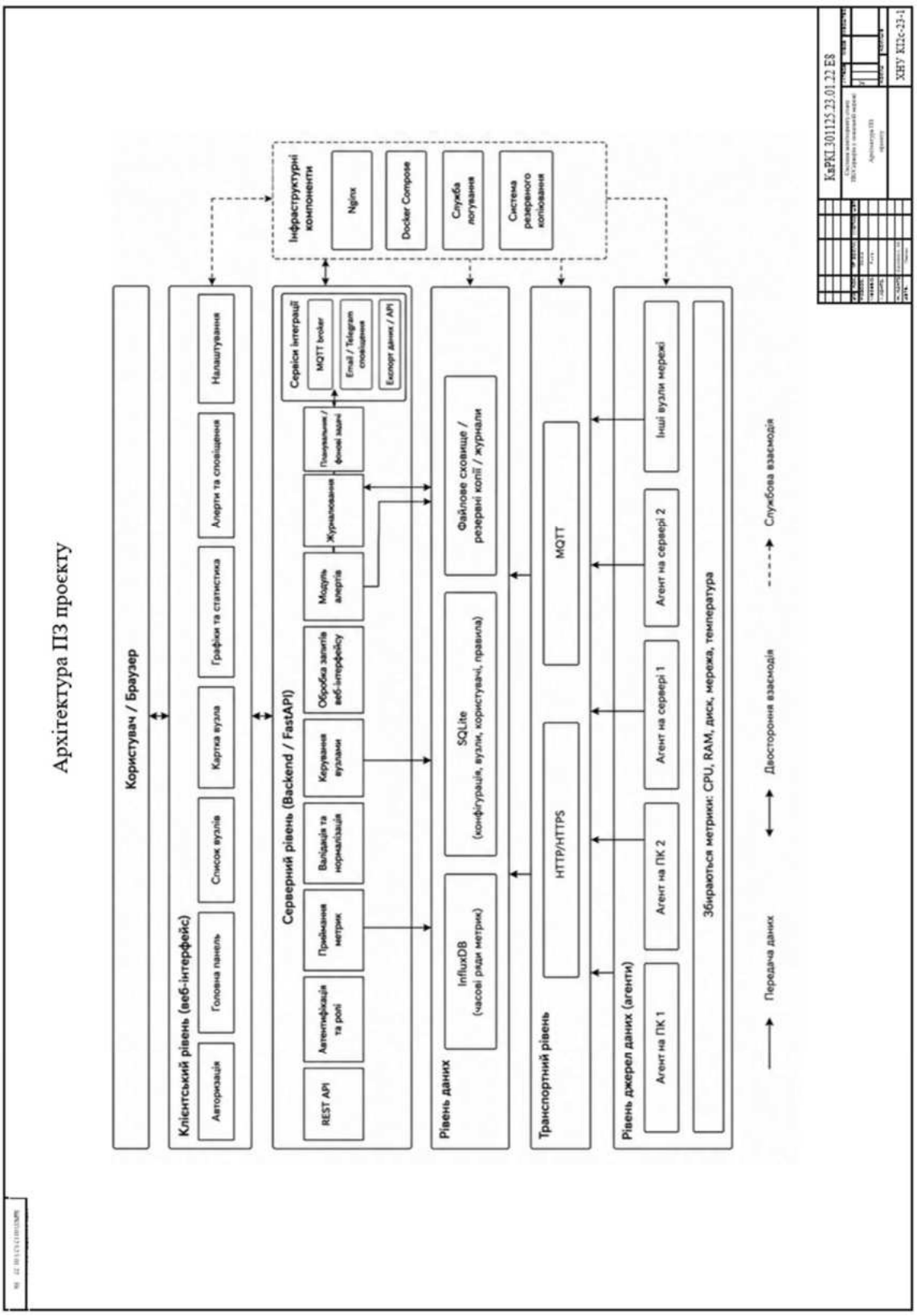
51. Інтерфейс Grafana. URL: https://www.google.com/imgres?q=%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%20Grafana&imgurl=https%3A%2F%2Fgrafana.com%2Fmedia%2Fproducts%2Fcloud%2Fgrafana%2Fgrafana-dashboard-english.png&imgrefurl=https%3A%2F%2Fgrafana.com%2Fgrafana%2F&docid=grJLQi2QJ0_CLM&tbnid=FxxO3YbJO6nloM&vet=12ahUKEwiC442UsfWSAxVcLBAIHRJPFmgQnPAOegQIHBAB..i&w=3360&h=1878&hcb=2&ved=2ahUKEwiC442UsfWSAxVcLBAIHRJPFmgQnPAOegQIHBAB

52. Інттерфейс InfluxDB. URL: https://siliconangle.com/2021/10/26/influxdb-updates-make-easier-build-time-series-database-apps/

# ДОДАТОК А

## (обов’язковий)

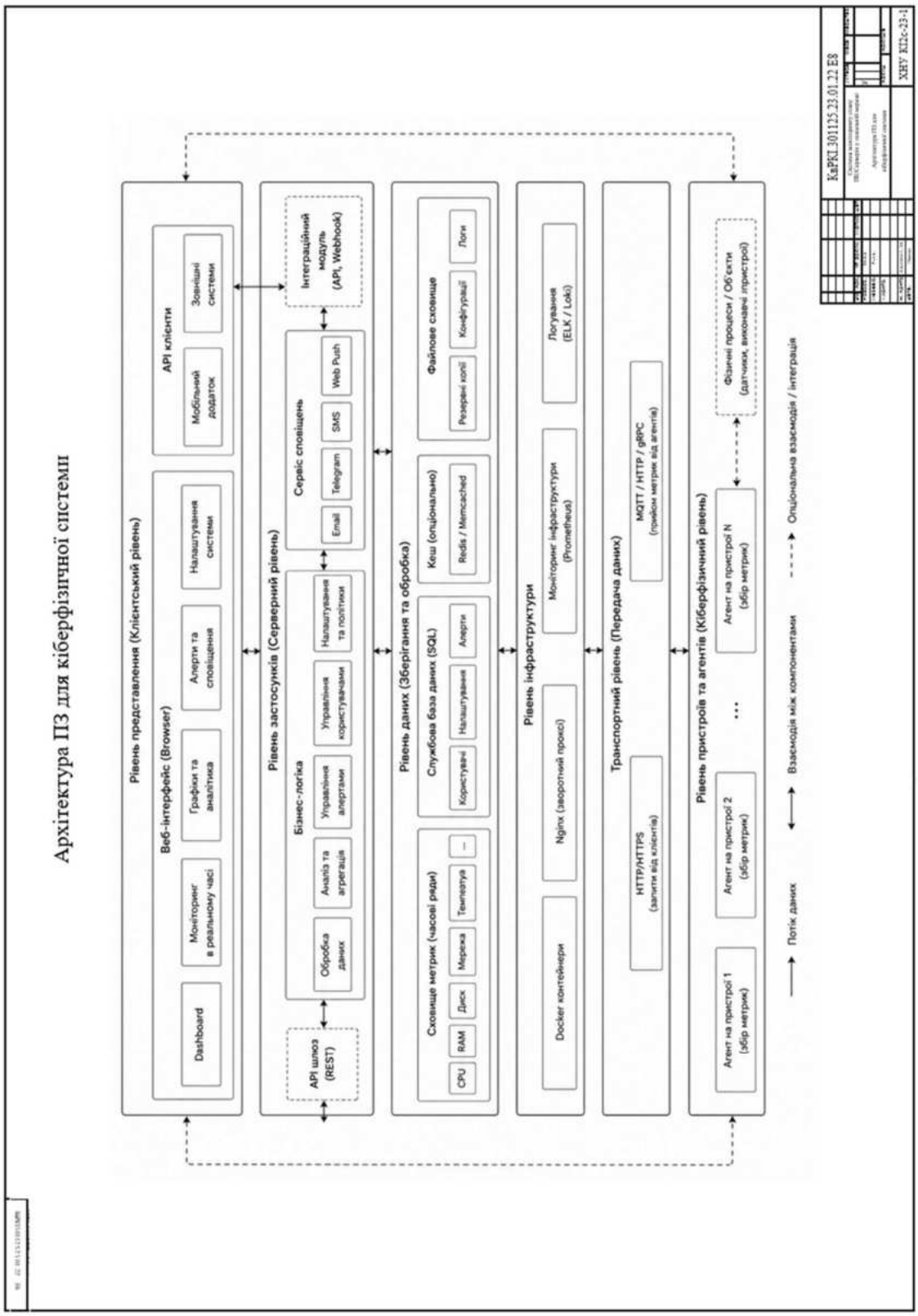
## Копія креслення «Архітектура ПЗ проєкту



## ДОДАТОК Б

**(обов’язковий)**

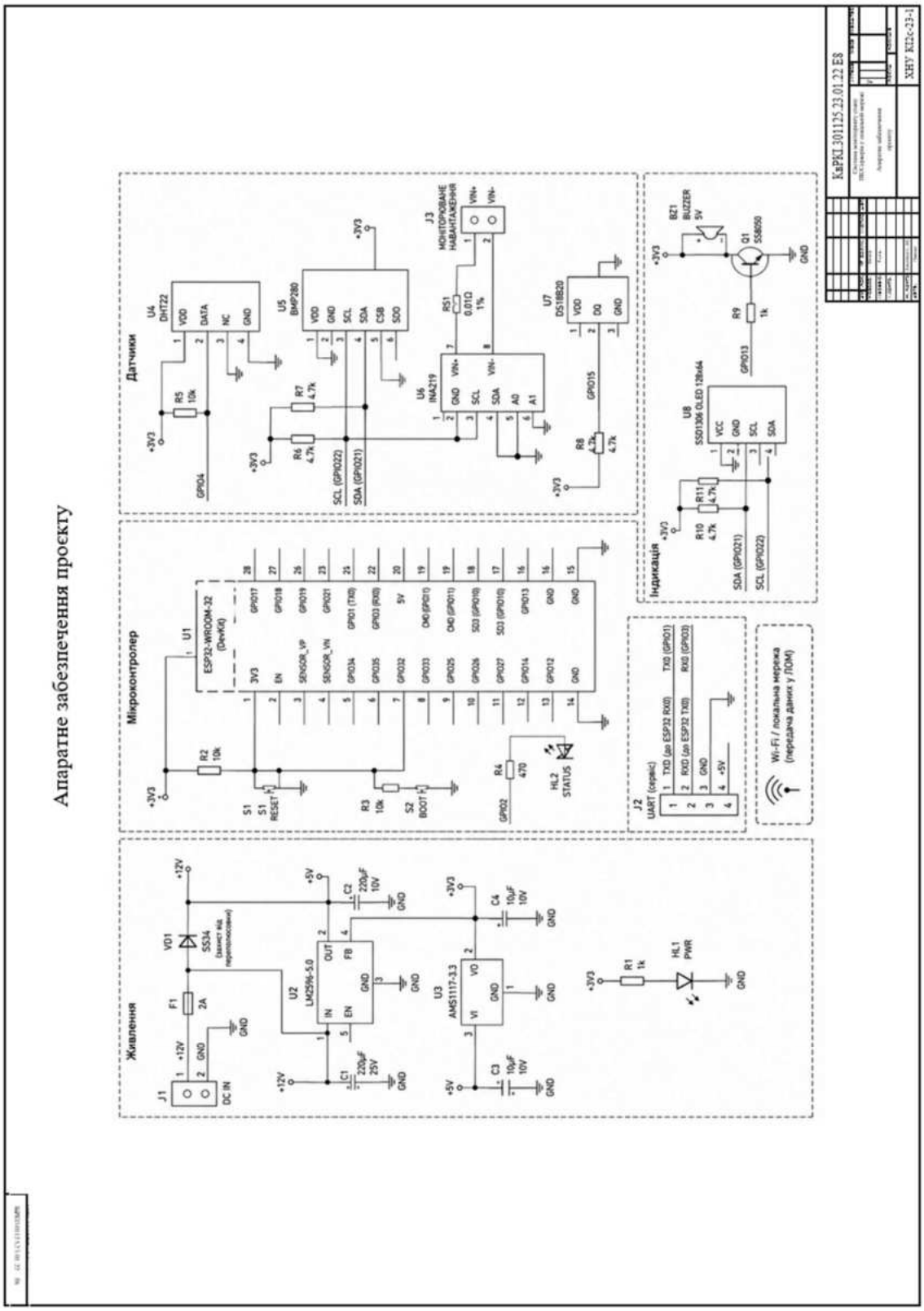
**Копія креслення «Архітектура ПЗ для кіберфізичної системи»**



# ДОДАТОК В

## (обов’язковий)

## Копія креслення «Апаратне забезпечення проєкту»



**Протокол аналізу звіту подібності експертом**

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

**Автор:** Владислав ЯРИШ

**Співавтор:**

**Назва:** Система моніторингу стану ПК/Серверів у локальній мережі

**Експерт:** Василь ЯЦКІВ

**Підрозділ:** Кафедра комп'ютерної інженерії та інформаційних систем

**Коефіцієнт подібності 1:**3.22%

**Коефіцієнт подібності 2:**1.49%

**Мікропробіли:** 3

**Заміна букв:** 0

**Інтервали:** 0

**Білі знаки:** 0

**Дата створення звіту:** 2026-06-01 09:15:26.0

**Після аналізу Звіту подібності констатую наступне:**

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-06-01 Доцент Андрій Нічепорук

*Дата* експерт

# Anti-Plagiarism (http://ap.km.ua) v-15.701

**Максимальне співпадіння з одним документом 1.0%**

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 8%**

| | Документ | | Сумарний збіг по Базі Даних | |
|---|---|---|---|---|
| ID: 272896<br>Назва: БКР Система моніторингу стану ПК/Серверів у локальній мережі<br>Додано в БД: 2026-06-01<br>Автора: Владислав ЯРИШ<br>Керівники: Василь ЯЦКІВ<br>Консультанти:<br>Опоненти: | Символи | Лексеми | Символи | Лексеми |
| | 140248 | 891 | 2778 (2%) | 37 (4%) |

Джерело плагіату

| ID | Опис | Наявність плагіату в документі | |
|---|---|---|---|
| | | Символи | Лексеми |

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

# РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Яриш Владислав Романович

Тема: Система моніторингу стану ПК/Серверів у локальній мережі

Спеціальність**:** 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___74___

1. Короткий зміст роботи та прийнятих рішень: Метою роботи є проєктування, реалізація та тестування програмної системи, призначеної для автоматизованого збору, передавання, обробки, збереження та візуалізації системних показників ПК і серверів у локальній мережі. Для досягнення поставленої мети проаналізовано предметну область, обґрунтовано архітектуру системи, вибрано технічні та програмні засоби реалізації, розроблено модель зберігання метрик, реалізовано агент збору системних показників, серверний модуль приймання й обробки даних та веб-інтерфейс для перегляду результатів моніторингу. У результаті створено програмне рішення, яке забезпечує централізований контроль стану обчислювальних вузлів, підтримує накопичення історії метрик і дає змогу своєчасно виявляти відхилення в роботі інфраструктури.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню. Усі вимоги вказані у завданні на кваліфікаційну роботу, а саме аналіз предметної області, проєктування та практична реалізація кіберфізичної системи керування якістю повітря в приміщеннях, виконані в повному обсязі.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі проаналізовано існуючі рішення для моніторингу обчислювальних систем та обґрунтовано доцільність розробки спеціалізованого комплексу для локальних мереж. В другому розділі спроєктовано загальну архітектуру системи, обґрунтовано вибір технологічного стеку для програмного агента та розроблено модель бази даних

системних метрик. В третьому розділі успішно виконано програмну реалізацію комплексу, налаштовано веб-інтерфейс візуалізації телеметрії та протестовано роботу рішення на базі комп'ютерних вузлів навчальної лабораторії..

4. Позитивні сторони роботи: Висока практична цінність роботи полягає у створенні гнучкого програмно-технічного комплексу, який дозволяє здійснювати централізований та узгоджений моніторинг розподілених обчислювальних вузлів у межах локальної мережі. Також варто відзначити якісно реалізовану модель бази даних часових рядів для тривалого збереження системних метрик та наочний веб-інтерфейс користувача для оперативного аналізу накопиченої телеметрії.

5. Негативні сторони роботи: Недоліком роботи є відсутність інтегрованої системи миттєвих автоматизованих сповіщень адміністратора у разі критичного перевищення допустимих метрик на обчислювальних вузлах. Наразі відстеження показників здійснюється виключно через веб-інтерфейс, що обмежує можливості оперативного реагування на аварійні ситуації в реальному часі без постійного візуального моніторингу панелі керування.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно згідно з діючими стандартами оформлення документації. Графічний матеріал, а саме архітектура програмного забезпечення, схема електрична принципова та схема логіки роботи, виконаний на високому технічному рівні.

7. Відгук про роботу в цілому: Робота виконана на високому науково-технічному рівні. Здобувач продемонстрував ґрунтовні знання у сфері проєктування сучасних кіберфізичних систем, програмування мікроконтролерів та інтеграції комплексних мережевих рішень.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре B (85)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Федула Михайло Васильович, к.т.н., доцент кафедри автоматизації, комп'ютерно-інтегр. техн. НТУ

"8" червень 2026 р. _____ (підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Владислав ЯРИШ

*ПІБ здобувача вищої освіти*

ФІТ, 3 курсу, групи КІ2с-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

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

1 травня 2026 року

# РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

## КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Система моніторингу стану ПК/Серверів у локальній мережі
Автор Владислав ЯРИШ
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.т.н., проф Василь ЯЦКІВ

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Ознаки академічного плагіату | |
| 1.1 | Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту. | відповідає |
| 1.2 | Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. | |
| 1.3 | Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 1.4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |
| 2 | Інші види порушень академічної доброчесності | |

Підтвердження:

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

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 3,22%; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри — Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми — Підпис — Андрій НІЧЕПОРУК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи — Підпис — Василь ЯЦКІВ (Ім'я, ПРІЗВИЩЕ)