НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

комп'ютерних наук

_____ Голуб Б.Л., доц., к.т.н

(підпис) (ПІБ)

"___"_____2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Система моніторингу та керування мережевими ресурсами»

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми

Д.е.н., професор _____ Руденський Р.А.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ Даков С.Ю.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Довгаль М.В.

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
_____ Голуб Б.Л., доц., к.т.н
(підпис) (ПІБ)
"___" _____2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи

студенту Довгалю Максиму Вікторовичу

Спеціальність 122 – «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи «Система моніторингу та керування мережевими ресурсами»

затверджена наказом ректора НУБіП України від "16" Грудня 2024 р. №2246 С

Термін подання завершеної роботи на кафедру 2025 . 06 . 02
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи: опис програмного забезпечення

Перелік питань, які потрібно розробити:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновок

Дата видачі завдання "_____"_____20___р.

Керівник бакалаврської кваліфікаційної роботи_____ Даков С.Ю.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Довгаль М.В.
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
1	Аналіз предметної області, постановка задачі та вивчення технічних вимог	03.02.2025 – 09.02.2025	Завершено
2	Проєктування логічної структури бази даних та побудова ER-діаграми	10.02.2025 – 16.02.2025	Завершено
3	Створення інформаційної бази та реалізація ключових таблиць, зв’язків та обмежень	17.02.2025 – 23.02.2025	Завершено
4	Розробка архітектури програмного забезпечення	24.02.2025 – 02.03.2025	Завершено
5	Вибір та налаштування інструментарію	03.03.2025 – 09.03.2025	Завершено
6	Розробка серверної логіки збору даних, моніторингу та збереження інформації	10.03.2025 – 23.03.2025	Завершено
7	Реалізація клієнтської частини	24.03.2025 – 06.04.2025	Завершено
8	Інтеграція з базою даних, налаштування авторизації, ролей та прав доступу	07.04.2025 – 13.04.2025	Завершено
9	Реалізація системи сповіщень, подій, тригерів, логування дій користувача	14.04.2025 – 20.04.2025	Завершено
10	Тестування системи, пошук та виправлення помилок, оптимізація роботи	21.04.2025 – 27.04.2025	Завершено
11	Написання пояснювальної записки	28.04.2025 – 04.05.2025	Завершено
12	Опис функціоналу системи	05.05.2025 – 11.05.2025	Завершено
13	Підготовка додатків	12.05.2025 – 18.05.2025	Завершено
14	Остаточне редагування записки, форматування, підготовка до подання	19.05.2025 – 25.05.2025	Завершено

АНОТАЦІЯ

У межах виконаної бакалаврської кваліфікаційної роботи здійснено проєктування та реалізацію системи моніторингу та керування мережевими ресурсами. Об’єктом дослідження є процеси адміністрування мережевої інфраструктури організації. Предметом дослідження виступають методи, інструменти та технології, що дозволяють здійснювати автоматизований контроль за станом мережевих пристроїв. У роботі було використано методи системного аналізу, логічного проєктування, моделювання інформаційних структур, а також методи побудови веборієнтованого прикладного програмного забезпечення.

У роботі розглядаються основні проблеми та підходи до організації моніторингу пристроїв і сервісів у сучасних комп’ютерних мережах (розділ 1). Проведено аналіз існуючих систем та оцінено їх переваги й недоліки. Описується процес побудови логічної моделі інформаційної бази, вибір системи управління базами даних та формування фізичної структури даних, яка забезпечує зберігання інформації про пристрої, параметри, події, користувачів і сповіщення (розділ 2). Аналізуються програмні модулі, алгоритми їх взаємодії, а також обґрунтовується вибір інструментів реалізації (розділ 3). Наведено етапи тестування, умови впровадження та вимоги до апаратного забезпечення (розділ 4).

Результатом проведеного дослідження стало створення повнофункціональної системи, яка дозволяє здійснювати моніторинг мережевих пристроїв у режимі реального часу. Реалізовано базу даних, веб інтерфейс користувача, механізм створення подій, журнал подій і формування умовних спрацювань. Система може бути використана в організаціях для забезпечення стабільної роботи мережевої інфраструктури, підвищення рівня керованості та своєчасного реагування на технічні збої. Також її можливо адаптувати до індивідуальних потреб завдяки гнучкій архітектурі.

ANNOTATION

As part of this bachelor's qualification work, a system of monitoring and management of network resources was designed and implemented. The object of the study is the processes of administering the organization's network infrastructure. The subject of the study is methods, tools and technologies that allow for automated monitoring of the state of network devices. The work used methods of system analysis, logical design, modeling of information structures, as well as methods for building web-based application software.

The work considers the main problems and approaches to organizing monitoring of devices and services in modern computer networks (section 1). An analysis of existing systems was conducted, and their advantages and disadvantages were assessed. The process of building a logical model of the information base, the selection of a database management system and the formation of a physical data structure that provides storage of information about devices, parameters, events, users and notifications (section 2) are described. Software modules, algorithms for their interaction are analyzed, and the choice of implementation tools is justified (section 3). The testing stages, implementation conditions and hardware requirements are presented (section 4).

The result of the research was the creation of a fully functional system that allows monitoring of network devices in real time. A database, a web user interface, an event creation mechanism, an event log and the formation of conditional triggers have been implemented. The system can be used in organizations to ensure stable operation of the network infrastructure, increase the level of manageability and timely response to technical failures. It can also be adapted to individual needs thanks to flexible architecture.

ЗМІСТ

ВСТУП

Інформаційні технології стали фундаментальною складовою економічного, соціального та технічного розвитку. Практично всі сфери людської діяльності покладаються на безперебійну роботу інформаційно-комунікаційних систем. У зв’язку з цим питання надійності, ефективності та безпеки цифрової інфраструктури набувають особливого значення.

Одним із ключових аспектів підтримки стійкості інформаційних систем є моніторинг та керування мережевими ресурсами. Це дозволяє в реальному часі спостерігати за станом мережевої інфраструктури, виявляти неполадки, реагувати на інциденти та забезпечувати стабільну роботу критичних сервісів. Поява нових протоколів, мережевих стандартів і моделей взаємодії лише посилює потребу у створенні гнучких, адаптивних та масштабованих рішень у сфері моніторингу.

Загальновідомо, що традиційні засоби обслуговування IT-інфраструктури часто виявляються громіздкими, складними у налаштуванні або надмірними за функціональністю. У відповідь на ці виклики актуальним стає розробка власних інструментів, які враховують обмежені ресурси середовища розгортання та забезпечують прозорий, інтуїтивно зрозумілий інтерфейс для адміністратора.

Разом із технічним прогресом зростають і ризики. Уразливості в мережевих пристроях, атаки на сервіси, DDoS, витоки даних — це лише частина проблем, що потребують своєчасного виявлення. Саме тому системи моніторингу є не лише засобом зручності, а інструментом забезпечення кіберстійкості й інформаційної безпеки. У світовій практиці спостерігається тенденція до розробки легких, спеціалізованих рішень, які не обтяжені зайвими модулями, легко інтегруються у хмарні середовища й адаптуються до нестандартних потреб. Подібні рішення можуть бути як локальними, так і розподіленими, що відкриває широкі можливості для їх застосування в умовах децентралізованої інфраструктури.

Метою даної бакалаврської кваліфікаційної роботи є проєктування та реалізація програмної системи моніторингу й керування мережевими ресурсами, яка функціонує автономно, підтримує базові вимоги до збирання метрик, фіксації подій, аналізу інцидентів і взаємодії з користувачем через веб-інтерфейс.

Для реалізації поставленої мети необхідно було:

- здійснити аналіз вимог до систем цього типу;
- побудувати структурну архітектуру компонентів майбутнього рішення;
- реалізувати функціональні модулі з використанням сучасних веб- і серверних технологій;
- створити інтерфейс для керування та перегляду подій;
- реалізувати базові механізми інформаційної безпеки та захисту доступу;
- провести тестування програмного комплексу;
- впровадити систему в експлуатаційне середовище.

Для реалізації проєкту використано такі інструменти: PostgreSQL як система управління базами даних, nginx – як сервер для маршрутизації запитів, Docker та docker-compose – для модульного розгортання усіх компонентів системи, HTML/CSS/JS та PHP – для реалізації веб-інтерфейсу. Розгортання системи виконано на базі Linux.

Система, розроблена у рамках цього дослідження, орієнтована на спрощене, але надійне розгортання, забезпечує базові можливості моніторингу без прив’язки до сторонніх рішень та може служити основою для подальшого розширення функціоналу згідно з потребами конкретного користувача чи організації.

Пояснювальна записка містить 61 сторінку основного тексту, ілюстрації, графічні моделі та додатки, які детально висвітлюють як процес розробки, так і результат впровадження створеної системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

У межах виконання дипломного проєкту було створено програмну систему моніторингу та керування мережевими ресурсами, призначену для забезпечення ефективного контролю за функціонуванням комп'ютерної мережі, своєчасного виявлення проблем та оптимізації роботи мережевої інфраструктури. Необхідність створення такої системи зумовлена постійним зростанням складності мережевих структур, кількості пристроїв, а також підвищеними вимогами до стабільності та надійності їх роботи.

Розроблена система оперує інформацією про активні пристрої у мережі, їхній статус, завантаженість, рівень трафіку та інші технічні характеристики. Також вона зберігає історію подій, включаючи зміну статусу пристроїв та інші ключові події, що впливають на функціонування мережі.

В процесі роботи системи автоматизовано виконуються рутинні операції, зокрема: перевірка доступності пристроїв, збір метрик, надсилання сповіщень про проблеми. Результати її функціонування відображаються у вигляді графіків, таблиць і текстових звітів, доступних для перегляду адміністратором у веб-інтерфейсі. Також реалізовано генерацію звітної документації за вибраний період (наприклад, щоденної або щомісячної статистики).

Розроблена система за своєю суттю належить до класу інформаційно-аналітичних систем, які забезпечують підтримку прийняття рішень на основі обробки та аналізу даних. За масштабом вона орієнтована на використання у локальних або корпоративних мережах малого та середнього розміру, з можливістю подальшого розширення. Система підтримує стандартні протоколи мережевого управління, такі як ICMP, SNMP, TCP/IP, що забезпечує її сумісність із широким спектром мережевих пристроїв. Комунікація реалізована за допомогою сучасних веб-технологій із використанням клієнт-серверної

архітектури, що дозволяє розгортати систему як у внутрішній мережі організації, так і в хмарному середовищі.

За ступенем автоматизації система є напівавтоматизованою: вона виконує основні функції без участі користувача, водночас надаючи адміністратору інтерфейс для ручного перегляду, аналізу та управління ресурсами. Завдяки цьому забезпечується гнучкість у налаштуванні та адаптації під специфіку конкретного середовища.

До основних вимог, які були реалізовані під час розробки, належать: безперебійна робота в режимі 24/7, висока швидкодія при обробці запитів, зручний та інтуїтивно зрозумілий інтерфейс користувача, підтримка розмежування прав доступу. Усі ці вимоги були враховані на етапах проєктування, розробки та тестування програмної системи.

1.2 Огляд інформаційних джерел та існуючих рішень

У процесі розробки будь-якої інформаційної системи важливою складовою є проведення ґрунтовного аналізу наявних інформаційних джерел, а також дослідження вже існуючих технічних і програмних рішень у вибраній предметній області. Це дозволяє не лише краще зрозуміти сучасні тенденції в галузі, а й виявити прогалини, які доцільно заповнити під час реалізації власного проєкту. Аналіз інформаційних джерел також сприяє формуванню більш чіткого бачення архітектури майбутньої системи, вибору доцільних технологій і обґрунтуванню технічних рішень.

Сучасна практика управління IT-інфраструктурою вимагає постійного моніторингу стану мережі, апаратного та програмного забезпечення, а також своєчасного реагування на інциденти. З огляду на це, в рамках дипломного проєкту було розглянемо існуючі системи моніторингу та керування мережевими ресурсами.

Аналіз сучасного стану предметної області показує, що для вирішення завдань моніторингу IT-інфраструктури сьогодні існує значна кількість як комерційних, так і вільно поширюваних програмних рішень. Основні вимоги до

таких систем — це забезпечення безперервного збору даних про стан об'єктів, надання зручного інтерфейсу для аналізу та своєчасне повідомлення про події.

1.2.1 Prometheus + Grafana

Prometheus у поєднанні з Grafana є одним із найпоширеніших рішень для моніторингу в хмарних та контейнеризованих середовищах. Основним завданням Prometheus є збір метрик у вигляді часових рядів, що дозволяє отримувати високоточні дані про стан системи з мінімальною затримкою. Grafana, у свою чергу, забезпечує гнучку візуалізацію цих метрик через дашборди, які користувач може налаштовувати відповідно до власних потреб [9].

Проте архітектура Prometheus передбачає децентралізований підхід, у якому кожен компонент виконує окрему функцію. Це робить систему гнучкою та масштабованою, однак потребує налаштування додаткових сервісів (наприклад, Alertmanager для обробки сповіщень), що може ускладнити впровадження для користувачів без досвіду в системному адмініструванні чи DevOps-практиках [4].

1.2.2 PRTG Network Monitor

PRTG Network Monitor — це готове до використання комерційне рішення для моніторингу мережі, серверів та іншого мережевого обладнання. Його основною цільовою аудиторією є малі та середні підприємства, які потребують швидкого впровадження та простого управління системою моніторингу без суттєвих витрат часу на навчання персоналу [6].

PRTG має сучасний веб-інтерфейс, який дозволяє зручно переглядати стан інфраструктури, формувати звіти та налаштовувати сповіщення. Завдяки вбудованим шаблонам для різних типів пристроїв і можливості автоматичного виявлення вузлів мережі, система швидко адаптується до нових середовищ. Також підтримується інтеграція з Active Directory, що спрощує керування доступом у корпоративних мережах.

Функціональність PRTG спрямована на надання максимально повної картини стану мережі з мінімальними зусиллями з боку адміністратора.

1.2.3 Nagios Core

Nagios Core є одним із найдавніших представників систем моніторингу, який і до сьогодні активно використовується завдяки своїй простій, модульній архітектурі. Основна ідея полягає у створенні плагінів, кожен з яких виконує перевірку певного ресурсу або сервісу. Такий підхід дозволяє адаптувати систему під конкретні потреби, використовуючи як готові рішення, так і власні скрипти [5].

Завдяки своїй відкритій структурі Nagios підтримує широкий спектр типів моніторингу — від доступності хостів до стану сервісів, затримок у мережі, використання ресурсів тощо. Система широко підтримується спільнотою, що забезпечує наявність великої кількості перевірених розширень та документації.

Однак з точки зору сучасного користувача, Nagios часто сприймається як консервативне рішення. Його конфігурація здійснюється через текстові файли, що може бути незручним при масштабуванні системи. Візуальна частина системи також поступається новішим рішенням, тому для покращення юзер-інтерфейсу часто використовуються додаткові надбудови або зовнішні панелі керування.

Розглянуті системи моніторингу демонструють різні підходи до організації контролю за станом мережевих ресурсів. Кожне рішення має свою специфіку, що зумовлює сферу його доцільного використання: від масштабованих хмарних середовищ до локальних мереж підприємств. Аналіз функціональних можливостей цих систем дозволяє краще зрозуміти сучасні вимоги до моніторингу IT-інфраструктури та слугує основою для обґрунтованого вибору технологій у рамках реалізації власного проєкту.

1.3 Моделювання предметної області

Проєктування будь-якої інформаційної системи, в тому числі системи моніторингу та керування мережевими ресурсами, доцільно починати з формування її моделі. Моделі допомагають краще зрозуміти предметну область, які об’єкти в ній беруть участь, як вони взаємодіють між собою, та які процеси

потребують автоматизації. Це дозволяє на ранньому етапі уникнути помилок і чітко визначити структуру та поведінку системи.

Для моделювання найчастіше використовуються стандартизовані графічні засоби, які забезпечують зручне візуальне подання інформації. Одним із найбільш поширених підходів є використання мови моделювання UML (Unified Modeling Language). Ця мова підтримується більшістю сучасних інструментів для проєктування та дозволяє створювати різні типи діаграм для опису як структури системи, так і процесів, які в ній відбуваються.

До найуживаніших засобів для побудови моделей відносяться такі програмні продукти, як draw.io, StarUML, Visual Paradigm, Lucidchart, Microsoft Visio, а також онлайнові платформи типу PlantUML, які дозволяють створювати діаграми на основі текстового опису. Зазначені інструменти використовуються як у навчанні, так і у професійній практиці.

У рамках цієї роботи було створено базовий комплекс моделей, що відображають ключові аспекти розроблюваної системи. До таких моделей належать:

- діаграма прецедентів, що демонструє основні сценарії взаємодії користувача із системою;
- діаграма діяльності, яка описує алгоритм виконання операцій у системі;
- діаграма послідовності, що показує обмін повідомленнями між компонентами системи в часі.

Створені моделі дозволяють краще зрозуміти логіку роботи майбутньої програми, перевірити повноту функціональності, виявити залежності між окремими частинами системи, а також забезпечують основу для наступного етапу — розробки структури інформаційної бази та прикладного програмного забезпечення.

1.3.1 Діаграма прецедентів

Діаграма прецедентів є одним з основних засобів моделювання функціональних можливостей інформаційної системи. Вона відображає, як

користувач (актор) взаємодіє з програмним забезпеченням. Такий тип діаграми використовується на ранніх етапах проєктування, щоб узагальнено описати сценарії роботи з системою та визначити перелік функцій, які має реалізовувати інтерфейс. Основною метою цієї діаграми є визначення того, які дії виконує користувач у системі, та опис взаємозв’язку між цими діями.

Серед елементів діаграми прецедентів:

1. Актори (Actors) – зовнішній користувач, який взаємодіє з системою.

2. Прецеденти (Use Cases) – конкретні функціональні сценарії, які реалізує система у відповідь на взаємодію з актором. Кожен прецедент описує окрему одиницю поведінки (наприклад, "авторизація", "формування звіту", "перевірка стану пристрою").

3. Зв’язки (Associations) – відображають взаємозв’язки між акторами та прецедентами (асоціації), а також між самими прецедентами (використання, розширення, включення).

Діаграма прецедентів не описує внутрішню реалізацію функцій, проте є дуже корисною на початкових етапах проєктування системи, оскільки:

- допомагає зафіксувати вимоги до системи;
- забезпечує розуміння між технічною та бізнес-сторонами;
- дає змогу легко виявити та усунути прогалини у вимогах;
- формує базу для створення інших типів моделей (послідовностей, класів, діяльності тощо).

Першим кроком у побудові діаграми стало визначення основного користувача системи — Адміністратора мережі. Це єдиний актор, який взаємодіє з розроблюваною системою. Він виконує всі дії, пов’язані з налаштуванням, моніторингом та керуванням мережевими ресурсами.

На наступному етапі було визначено ключові функціональні можливості системи, які представляються у вигляді прецедентів:

- Перегляд дашборду – базова функція, що дозволяє отримувати інформацію в узагальненому візуальному вигляді;

- Птворення дашборду – є складовою частиною попереднього прецеденту, оскільки для перегляду спочатку необхідно сформувати дашборд;
- Додавання вузлів мережі – забезпечує розширення системи шляхом додавання нових пристроїв;
- Створення тригерів – функція, яка може виконуватись як розширення базового сценарію додавання вузлів;
- Перевірка наявності проблем – дає змогу ідентифікувати несправності у функціонуванні мережі або її елементів;
- Перевірка доступності вузлів мережі – дозволяє адміністратору переконатися в тому, що всі вузли доступні і працюють коректно.
- Перегляд останніх даних – забезпечує доступ до найсвіжіших отриманих метрик і параметрів роботи мережі.

Отже, діаграма прецедентів (рис. 1.1) дозволяє наочно відобразити основні функціональні можливості системи моніторингу та керування мережевими ресурсами, а також взаємодію користувача з нею. Вона слугує логічною основою для подальшого моделювання програмної системи, забезпечуючи цілісне розуміння вимог та сценаріїв використання майбутнього програмного продукту.

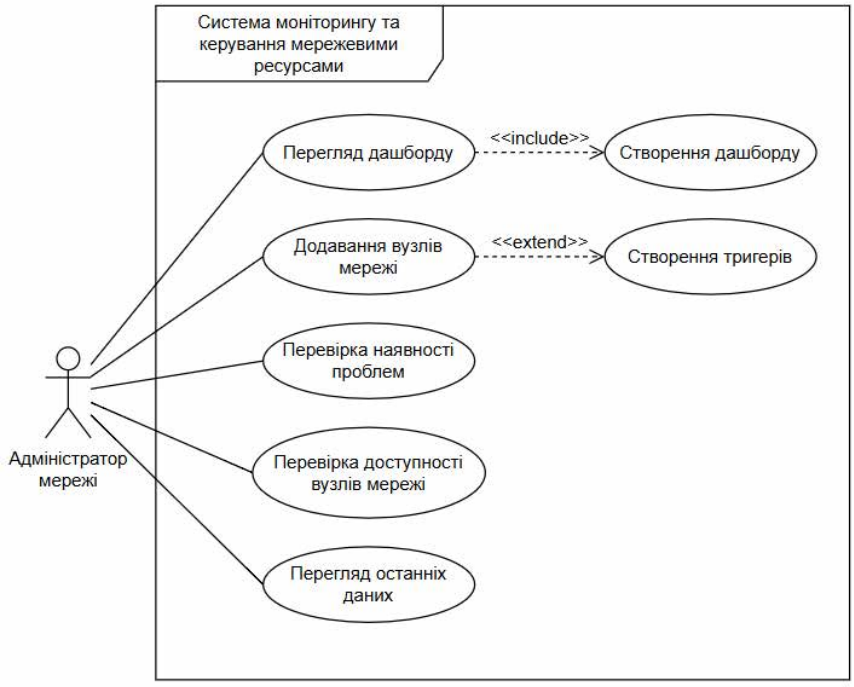

Рис. 1.1 Діаграма прецедентів

1.3.2 Діаграма діяльності

Діаграма діяльності (англ. Activity Diagram) є одним з основних інструментів в UML, який використовується для моделювання робочих процесів і алгоритмів у системах. Вона дозволяє відобразити послідовність дій, умовні переходи та логіку виконання процесів. Діаграма діяльності широко застосовується для опису складних процесів, що складаються з кількох етапів, у тому числі для ілюстрації сценаріїв користувача, бізнес-процесів або внутрішніх алгоритмів роботи програмних систем.

Цей тип діаграм дозволяє детально представити внутрішню логіку виконання окремих прецедентів або операцій, зображуючи не лише дії, а й можливі розгалуження, паралельні гілки та завершення процесів.

Основними елементами діаграми діяльності є:

- Дія (Action): одинична операція або завдання, яке виконується в межах процесу;
- Розгалуження (Decision): точка, де процес розділяється на кілька варіантів виконання в залежності від умов;
- Об'єкти (Object): використовуються для позначення об'єктів, з якими взаємодіє процес;
- Початок і кінець процесу (Start/End): точки, що визначають початок і завершення процесу;
- Потоки (Flow): лінії, які показують порядок виконання дій.

Таким чином, діаграма діяльності дозволяє чітко уявити процес, визначити можливі варіанти його виконання та оптимізувати його для покращення ефективності системи.

Побудова діаграми (рис. 1.2) починається зі збору діагностичної інформації, що дозволяє системі отримати повну картину про стан мережевої інфраструктури. Ці дані є основою для подальших дій – без них неможливо ні виявити проблеми, ні прийняти обґрунтовані рішення.

Після збору даних система перевіряє доступність вузлів — чи працює обладнання, чи є з'єднання. Якщо якась частина мережі недоступна, це критична

ситуація, що одразу викликає тривогу й сповіщення адміністратора. Якщо ж доступність підтверджена, запускається глибша діагностика, що охоплює перевірку помилок, формування візуальних графіків і збір метрик для подальшого аналізу.

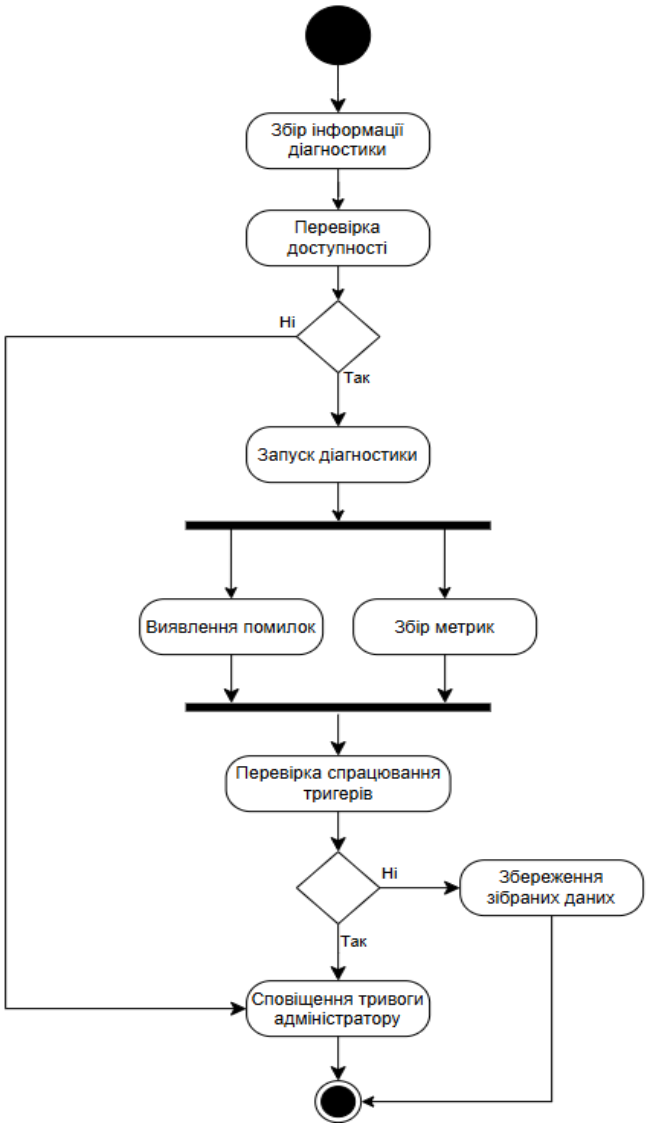

Рис. 1.2 Діаграма діяльності (Збір даних)

Після завершення аналізу система оцінює, чи були виявлені критичні події – тобто чи спрацювали налаштовані тригери. У разі виявлення таких подій адміністратор миттєво отримує сповіщення.

Якщо ж критичних відхилень не виявлено, система просто зберігає зібрані дані для історичного аналізу та подальшого прийняття рішень. Таким чином, навіть у нормальному режимі роботи система продовжує накопичувати

інформацію, що дозволяє будувати дашборди, виявляти довгострокові тренди та підвищувати ефективність мережі.

У результаті, діаграма діяльності системи демонструє не просто послідовність дій, а цілісний цикл адаптивного контролю за станом мережі. Вона ілюструє, як система автоматично поєднує аналіз, реагування та інформування — забезпечуючи стабільну роботу мережевої інфраструктури без постійної ручної участі адміністратора.

1.3.3 Діаграма послідовності

Діаграма послідовності (англ. Sequence Diagram) є важливою складовою інструментарію UML, яка використовується для моделювання динамічної поведінки системи. Її основне завдання — показати, яким чином об’єкти чи компоненти взаємодіють між собою в часовій площині, тобто в якій саме послідовності відбувається обмін повідомленнями у межах певного сценарію. Це не просто схема технічних взаємодій, а спосіб відтворення логіки поведінки системи в реальних умовах її використання.

На відміну від більш загальних структурних моделей, таких як діаграми класів або компонентів, діаграма послідовності акцентує увагу саме на подієвому аспекті. Вона дозволяє зрозуміти, як реагує система на конкретну дію користувача, які процеси запускаються у відповідь, які компоненти між собою комунікують, і в якій черговості відбувається ця комунікація. Таким чином, вона стає незамінною під час проєктування логіки складних сценаріїв, які включають кілька етапів взаємодії між підсистемами, або ж коли важливо прослідкувати умови виконання певної функції.

Особливо корисною така модель є для систем, де критично важливою є не просто наявність певного функціоналу, а саме порядок і синхронізація його виконання. Наприклад, у системах моніторингу, де потрібно не лише зібрати інформацію, а й оперативно її обробити, прийняти рішення на основі метрик, активувати тригери, сформувати сповіщення й у відповідний момент передати ці дані користувачу або іншій системі. Усе це потребує чіткого розуміння, в якій послідовності відбуваються дії, що й відображається в діаграмі.

Діаграма послідовності також допомагає виявити слабкі місця у логіці обміну — наприклад, надмірну залежність одного компонента від іншого, потенційні затримки, дублювання запитів або відсутність підтвердження відповіді. Це надзвичайно цінно під час оптимізації системи, коли важливо не лише реалізувати функціональність, а й зробити її ефективною, надійною та передбачуваною в роботі.

Таким чином, діаграма послідовності – це не просто формальність чи ілюстрація, а засіб осмислення логіки взаємодії у живому сценарії, що значно підвищує якість планування, впровадження й підтримки системи в майбутньому.

У структурі діаграми послідовності можна виділити кілька ключових елементів, кожен з яких виконує специфічну роль у моделюванні взаємодії між об’єктами системи:

1. Актори та об’єкти:

Розташовуються горизонтально у верхній частині діаграми. Це можуть бути користувачі (актори), програмні модулі, сервіси або бази даних. Кожен учасник має свою життєву лінію, яка вертикально простягається вниз, символізуючи часову вісь існування об’єкта протягом сценарію.

2. Життєва лінія:

Це пунктирна вертикальна лінія, що відходить вниз від кожного учасника. Вона показує існування об’єкта протягом часу. Взаємодії відображаються саме між цими лініями.

3. Повідомлення:

Стрілки, які з’єднують життєві лінії учасників. Вони показують, як один об’єкт надсилає запит іншому або отримує відповідь.

- Синхронні повідомлення (запит з очікуванням відповіді) позначаються суцільною стрілкою;
- Асинхронні повідомлення (наприклад, події або сповіщення) — стрілкою з половинною головкою;
- Відповіді часто не підписуються, якщо їх зміст очевидний, або можуть позначатися зворотною стрілкою.

4. Активність:

Це вузький прямокутник на життєвій лінії, який вказує, що в даний момент об’єкт виконує певну дію або метод. Його тривалість відображає період, коли об’єкт «активний».

5. Повернення:

Позначається пунктирною стрілкою знизу догори – це повернення управління від одного об’єкта до іншого після завершення дії.

6. Альтернативи та умовні блоки:

У разі складніших сценаріїв діаграма може містити фрагменти – умовні блоки (alt, opt, loop), які імітують логічні розгалуження (наприклад, якщо вузол недоступний – одне повідомлення, якщо доступний – інше).

В розроблюваній системі головним ініціатором усіх подій є мережевий адміністратор, який виступає в ролі користувача системи, відповідального за моніторинг, налаштування та оцінку стану мережевих вузлів. Його взаємодія з сервером – центральним компонентом програмного забезпечення – є основою усієї логіки діаграми.

У процесі побудови діаграми послідовності (Рис. 1.3) було змодельовано типову логіку взаємодій, яка охоплює кілька сценаріїв використання: перевірку доступності мережевого пристрою, налаштування сповіщень через тригери, а також формування дашбордів для візуалізації отриманих метрик. Така діаграма дозволяє детально простежити, як саме відбувається обробка запитів адміністратора, які відповіді генерує сервер, та у якій послідовності виконуються дії.

Зокрема, у першій частині діаграми показано, як адміністратор ініціює перевірку доступності пристрою, після чого сервер виконує діагностичний скрипт і повертає результат. Далі демонструється налаштування тригерів – користувач визначає умови спрацювання, а система, у разі їхнього виконання, надсилає сповіщення. У підсумку, відображено процес створення нового дашборду, що дає змогу адміністратору аналізувати зібрані дані у зручному форматі.

Таким чином, наведена діаграма не лише фіксує технічні деталі взаємодії, а й дозволяє виявити критичні точки у логіці обміну даними, що надзвичайно важливо при розробці надійного та гнучкого програмного забезпечення.

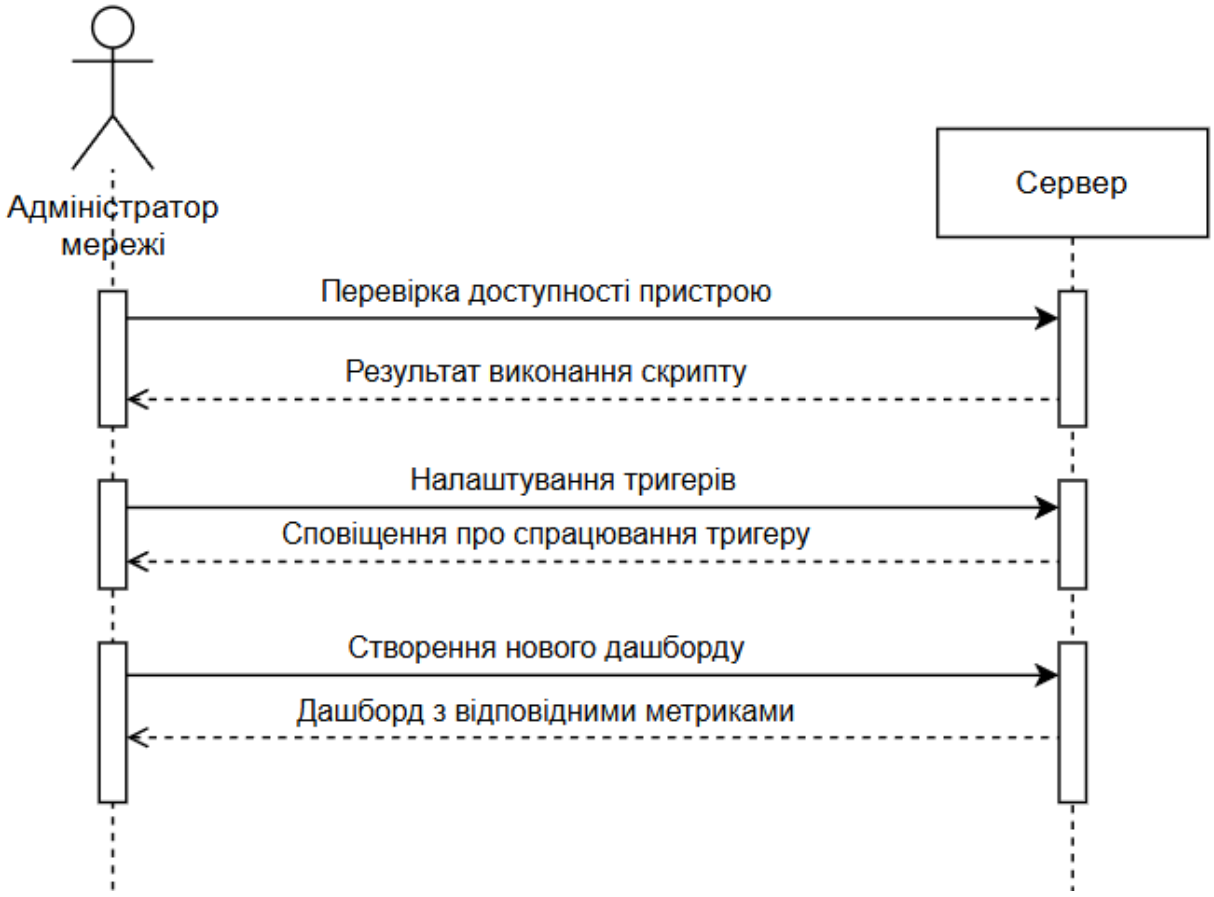

Рис. 1.3 Діаграма послідовності

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

У будь-якій автоматизованій системі інформаційне забезпечення відіграє велику роль. Саме воно визначає, які саме дані зберігаються, як вони організовані, структуровані, передаються між компонентами системи та як до них здійснюється доступ. Від якості побудови інформаційної бази залежить стабільність роботи системи, її масштабованість, зручність у користуванні та безпечність. Іншими словами, інформаційне забезпечення – це основа, на якій базується вся робота майбутньої системи.

В основі інформаційного забезпечення лежить інформаційна база. Під нею розуміють сукупність даних, що логічно пов’язані між собою та організовані у певну структуру, яка дозволяє ефективно зберігати, знаходити та опрацьовувати інформацію. Дані, що зберігаються в інформаційній базі, є основним джерелом для прийняття рішень, побудови звітів, відображення інформації в інтерфейсі та фіксації змін, які відбуваються в системі.

Розробка інформаційного забезпечення завжди починається зі створення логічної моделі – абстрактної структури, яка показує, які сутності (таблиці) існують, які між ними є зв’язки, та які атрибути має кожна сутність. Логічна модель дозволяє ще до реалізації на СУБД побачити загальну картину, виявити зайві повтори, логічні протиріччя. Це дуже важливий етап, оскільки саме тут закладаються основи для майбутньої роботи з базою даних.

Зазвичай логічна модель даних представляється у вигляді ER-діаграми (від англ. Entity–Relationship), що ілюструє взаємозв’язки між сутностями. У сучасних умовах така діаграма може бути побудована за допомогою відповідного програмного інструментарію, наприклад, у середовищі. ER-діаграма дозволяє візуалізувати структуру даних: таблиці, поля, первинні та зовнішні ключі, типи зв’язків («один до одного», «один до багатьох» тощо), а також забезпечує зручність подальшого перенесення цієї структури у фізичну базу даних.

Наступним кроком після створення логічної моделі є вибір конкретної системи управління базами даних (СУБД), у якій ця модель буде реалізована. Це

програмне середовище, яке відповідає за зберігання, зміну, пошук і захист даних. Вибір СУБД залежить від багатьох факторів: обсягу даних, необхідного рівня захисту, вимог до швидкості роботи, особливостей розгортання, сумісності з іншими компонентами системи.

Після вибору СУБД виконується створення інформаційної бази – тобто реалізація фізичної структури даних. Це включає створення таблиць, визначення полів, типів даних, ключів, обмежень, зв’язків між таблицями, а також налаштування механізмів доступу до інформації. Якщо система працює через інтерфейс (наприклад, веб-інтерфейс), то також потрібно налаштувати рівні доступу для користувачів, механізми авторизації, збереження логів та інші супутні речі.

Таким чином, інформаційне забезпечення в даній системі охоплює цілий комплекс завдань: від логічного проєктування структури даних — до реалізації таблиць, налаштування зв’язків і забезпечення їх надійного зберігання. Саме ця частина роботи створює підґрунтя для всієї функціональності програмного продукту та визначає, наскільки стабільною, ефективною та безпечною буде його робота.

2.1 Логічна модель даних

Логічна модель даних–фундаментальний компонент в процесі розробки програмного забезпечення для системи. Вона представляє собою абстрактне подання даних, визначаючи організацію інформації, перелік сутностей (об’єктів), присутніх в системі, їхні властивості (атрибути), а також логічні взаємозв'язки між ними.

Створення логічної моделі дозволяє ще на етапі проєктування системи чітко уявити, як виглядатиме база даних, яка інформація буде в ній зберігатись, як її можна буде знайти, змінити або проаналізувати. Це своєрідна «карта» майбутньої бази, яка ще не реалізована фізично, але вже дозволяє оцінити її структуру та повноту.

Логічна модель не прив’язана до конкретної системи керування базами даних (СУБД), мови програмування чи платформи. Її можна розглядати як нейтральну та універсальну форму представлення даних. Саме завдяки такому підходу логічна модель може бути використана незалежно від того, чи буде система реалізована у MySQL, PostgreSQL, SQLite або іншому середовищі.

Для даної роботи логічна модель даних була сформована з урахуванням завдань, що виконує система. Основна ідея полягає в тому, що система має вміти зберігати відомості про певні мережеві об’єкти, які вона контролює, а також фіксувати події, що відбуваються під час моніторингу (наприклад, недоступність вузла, відновлення зв’язку, затримки тощо). Крім того, система має забезпечити контроль доступу користувачів через авторизацію, що також передбачає зберігання відповідної інформації.

Таким чином, у логічна модель (рис. 2.1) охоплює декілька пов’язаних між собою блоків:

1. Пристрої (Протокол, Порт, Тип пристрою, Статус пристрою, Пристрій):

Центральним елементом є таблиця Пристрій, яка зберігає основну інформацію про об’єкти, що підлягають моніторингу. Для кожного пристрою фіксується його назва, IP-адреса, версія прошивки та модель. Пристрій має зв’язки з таблицею Тип пристрою, яка дозволяє класифікувати обладнання за функціональним призначенням (наприклад, сервер, маршрутизатор, точка доступу тощо), та з таблицею Статус пристрою, де зберігається поточний стан (наприклад, “активний”, “неактивний”, “відсутній”).

Крім того, пристрої мають прив’язку до використовуваних мережевих протоколів та портів, що реалізовано через таблиці Протокол і Порт. Це дозволяє зберігати інформацію про те, через які канали відбувається взаємодія з пристроєм, і забезпечити можливість гнучкого налаштування перевірок.

2. Елементи даних

Для кожного пристрою може бути визначено один або кілька моніторингових елементів, які представляють собою конкретні параметри, що

підлягають перевірці. Це реалізовано через таблицю Моніторингові дані, в якій фіксуються: назва елемента, його значення, час останнього оновлення, період оновлення, тип даних та ключ (який виступає унікальним ідентифікатором параметра).

Кожен моніторинговий елемент має зв’язок із таблицею Тип даних, де визначено одиницю виміру (наприклад, мілісекунди, відсотки, байти тощо). Це дозволяє уніфікувати представлення параметрів та забезпечити правильність математичних операцій (середнє значення, порівняння, нормалізація).

3. Умови спрацювання

Кожен моніторинговий елемент може бути пов’язаний з одним або кількома тригерами – умовами, що визначають ситуацію, яка вимагає реагування. Таблиця Тригер містить інформацію про те, до якого елемента належить тригер, який його поточний стан, яка умова використовується для порівняння (наприклад, “> 80%”) та який поріг задано.

Тригери пов’язані з таблицею Умова спрацювання, де зазначено назву умови та тип порівняння (більше, менше, дорівнює тощо), а також із таблицею Стан тригера, яка дозволяє відслідковувати, чи був тригер активований, та в якому він стані зараз.

4. Події

Якщо значення параметра перевищує встановлений поріг і умова спрацювала, система створює запис у таблиці Подія. Подія містить опис ситуації, час виникнення, код тригера, а також статус (активна/усунена). Кожна подія також може мати призначене сповіщення, яке фіксується в окремій таблиці Сповіщення – тут зберігається адреса електронної пошти, на яку надіслано повідомлення.

Зв’язок між подією та її статусом реалізовано через таблицю Статус події, що дозволяє класифікувати всі ситуації за рівнем серйозності (наприклад, “інформаційна”, “попередження”, “критична”).

Уся модель побудована з дотриманням вимог реляційної нормалізації та відповідає третій нормальній формі. Вона є логічно завершеною, охоплює як

статичні параметри (пристрої, типи, протоколи), так і динамічні (значення, події, стани), а також реалізує повний цикл – від збору даних до сповіщення користувача про виявлену проблему.

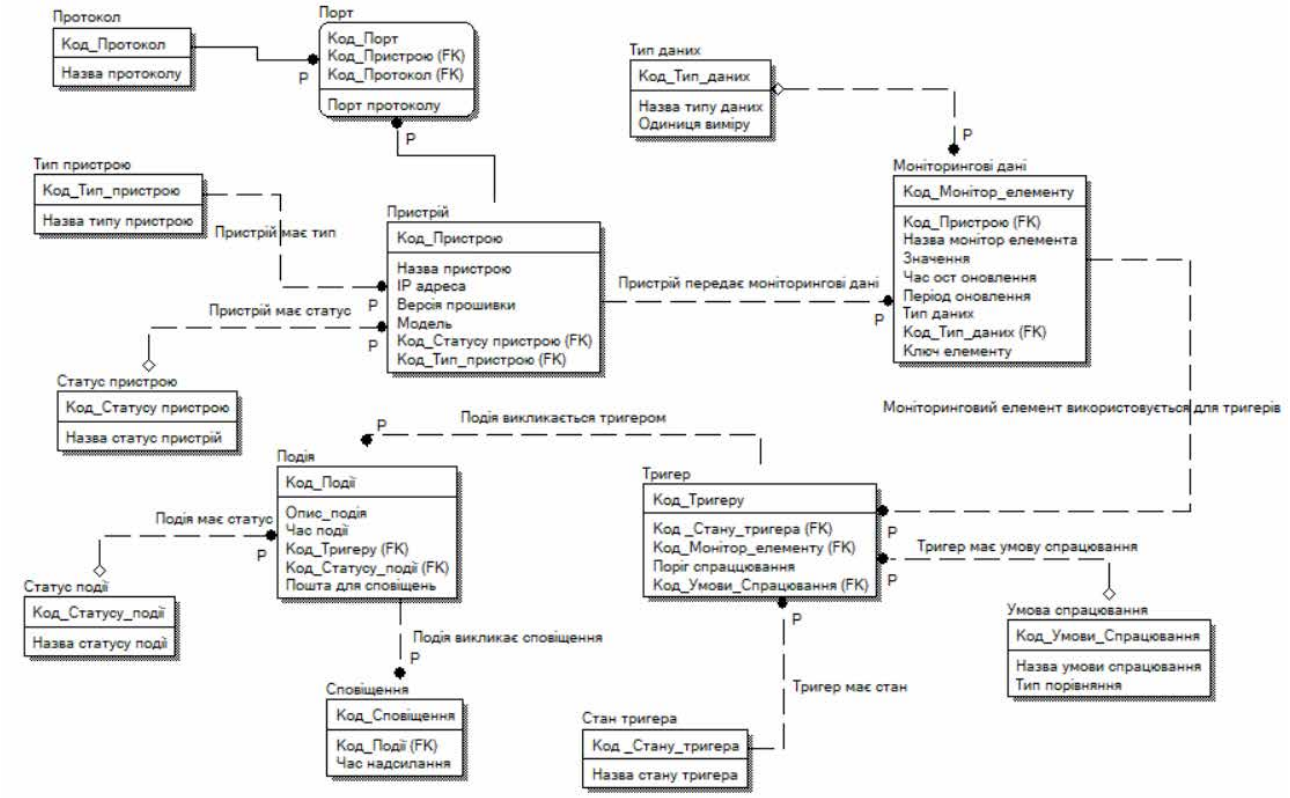

Рис. 2.1 Логічна модель

2.2 Вибір системи управління інформаційною базою

Під час проєктування будь-якої системи велику роль відіграє вибір системи управління базою даних (СУБД). Адже вона забезпечує зберігання, обробку, структурування та надійний доступ до даних, які є основою функціонування будь-якого застосунку, особливо якщо йдеться про системи, що потребують постійного обміну інформацією та збереження історичних записів.

Від якості та можливостей обраної СУБД залежить не лише загальна продуктивність системи, а й її масштабованість, стабільність, зручність адміністрування, легкість інтеграції з іншими сервісами, рівень безпеки та обсяг технічного супроводу. Оскільки інформаційна база є «серцем» будь-якої сучасної цифрової системи, недооцінка цього вибору може призвести до серйозних проблем у майбутньому, зокрема: до зниження швидкості обробки

запитів, втрати даних, складнощів з міграцією чи розширенням функціоналу.

Існує велика кількість СУБД, які широко використовуються в проєктах різного масштабу – від невеликих внутрішніх сервісів до комплексних веб-застосунків, які обробляють мільйони записів щодня. Серед найвідоміших і найчастіше застосовуваних СУБД:

- MySQL – одна з найпопулярніших реляційних СУБД, яка активно використовується у веб-розробці. Підтримує більшість SQL-операцій, має відкритий код, працює стабільно на багатьох платформах. Переважно застосовується у зв’язці з PHP або легкими CMS. Підтримує реплікацію, індексацію, але має деякі обмеження у роботі зі складними транзакціями та структурами.
- SQLite – компактна вбудована СУБД, що зберігає базу у вигляді одного файлу. Перевагою є простота, незалежність від сервера та відсутність потреби в адмініструванні. Однак ця система не підходить для великих обсягів даних або паралельного доступу з декількох клієнтів.
- Oracle Database – потужне комерційне рішення, яке використовується у великих корпораціях. Має дуже широкі можливості, але складне у налаштуванні й потребує ліцензії. Не підходить для невеликих проєктів через складність супроводу.
- PostgreSQL – розвинена СУБД з відкритим кодом, яка повністю підтримує стандарти SQL, транзакційність, реляційні зв’язки, тригери, індекси, ролі, механізми безпеки. Активно використовується як у невеликих, так і у великих проєктах. Підтримує роботу з JSON, розширення, геодані, реплікацію та резервне копіювання. Вважається однією з найнадійніших СУБД з відкритим кодом.

Під час вибору оптимальної СУБД для розробки системи моніторингу та керування мережевими ресурсами враховувалися такі аспекти, як характер даних, інтенсивність їх зміни, вимоги до безпеки, можливість розгортання в контейнеризованому середовищі, рівень автоматизації, а також перспективи масштабування.

У підсумку було обрано PostgreSQL — сучасну, надійну та функціонально багату СУБД, яка чудово підходить для побудови інформаційних систем як середнього, так і великого масштабу. Це рішення було обґрунтовано рядом технічних та організаційних причин, зокрема:

1. Відповідність функціональним вимогам:

Система моніторингу працює з великою кількістю періодично оновлюваних записів (елементи даних, результати перевірок, події). PostgreSQL добре працює з частими INSERT-операціями та дозволяє ефективно обробляти тисячі записів, що надходять з інтервалом у десятки секунд.

2. Підтримка складної логіки:

PostgreSQL дозволяє створювати складні запити, представлення (view), тригери та умови, які можуть бути використані, наприклад, для побудови звітів, графіків, агрегацій тощо. Це важливо для аналітичної частини інтерфейсу.

3. Цілісність і структурованість даних:

PostgreSQL повністю підтримує зовнішні ключі, каскадні оновлення та індексацію, що гарантує цілісність, швидкість, унеможливлює появу неповних або суперечливих записів.

4. Вбудована безпека:

У PostgreSQL реалізовано гнучке керування правами доступу, авторизація користувачів, шифрування підключень, підтримка журналів змін. Це дозволяє захистити дані навіть у відкритому середовищі.

5. Підтримка у Docker:

Враховуючи, що вся система розгортається у контейнеризованому середовищі, важливо, щоб СУБД без проблем працювала як окремий сервіс. PostgreSQL має офіційний образ, простий у розгортанні, оновленні, резервному копіюванні.

6. Активна спільнота та підтримка розширень:

Вибір відкритої СУБД знімає потребу у додаткових ліцензіях, а також дає змогу швидко знайти відповіді на технічні питання завдяки великій кількості документації та прикладів, а також підтримує велику кількість розширень.

На практиці використання PostgreSQL дозволяє легко створити фізичну модель бази даних, яка безпосередньо відображає логічну структуру, описану раніше. Вона підтримує всі необхідні типи даних, дозволяє створювати ключі, обмеження, індекси, а також розширює можливості в напрямку подальшого розвитку системи: впровадження збережених процедур, оптимізації запитів, підключення зовнішніх аналітичних засобів.

У результаті, вибір PostgreSQL як СУБД для даної системи є обґрунтованим з технічної, функціональної та організаційної точки зору. Вона задовольняє усі поточні потреби, забезпечує стабільну роботу та високу продуктивність.

2.3 Створення інформаційної бази

Після побудови логічної моделі даних та вибору системи управління базами даних наступним етапом стало створення безпосередньо інформаційної бази, тобто реалізація фізичної структури, яка використовується в роботі системи. Створення бази даних передбачає переведення абстрактної логічної структури у конкретну форму: визначення таблиць, полів, типів даних, ключів, обмежень, а також зв’язків між таблицями.

Реалізація інформаційної бази здійснювалась у середовищі СУБД PostgreSQL, яке було обране на підставі функціональних та технологічних вимог до системи. Створення бази здійснюється на етапі запуску контейнерів за допомогою docker-compose, що дозволяє автоматизувати ініціалізацію структури та забезпечити швидке розгортання системи на будь-якому сервері.

Інформаційна база включає кілька функціональних блоків: вузли мережі, групи, елементи даних, події, сповіщення, користувачі та історія даних. Усі таблиці пов’язані між собою за допомогою зовнішніх ключів, що дозволяє гарантувати логічну цілісність даних.

У структурі інформаційної бази можна виділити кілька основних таблиць:

1. Таблиці для обліку вузлів мережі:

Ця таблиця містить інформацію про всі пристрої, які підлягають

моніторингу. Вузол мережі – це базова сутність системи, до якої прив’язуються всі параметри перевірок. Тут зберігається його унікальний ідентифікатор, IP-адреса, тип, статус, опис та інші характеристики.

Для організації пристроїв за логічними або функціональними ознаками реалізовано таблицю hosts_groups, яка встановлює зв’язок багато-до-багатьох між пристроями та групами. Це дозволяє, наприклад, відображати лише певну категорію пристроїв в інтерфейсі або застосовувати масові зміни до групи.

2. Таблиці елементів моніторингу:

Однією з ключових таблиць у структурі бази даних є таблиця items, яка відповідає за зберігання усієї інформації про параметри, що підлягають регулярному моніторингу в межах окремих пристроїв. Кожен запис у цій таблиці відповідає окремому елементу даних — тобто певному показнику, який система періодично опитує з використанням заданого механізму.

Кожен елемент даних обов’язково має бути прив’язаний до певного пристрою через зовнішній ключ. Така прив’язка дозволяє будувати індивідуальний набір перевірок для кожного вузла моніторингу. При цьому окремі вузли можуть мати абсолютно різні параметри перевірки, з різною частотою, різними способами збору, одиницями вимірювання та тривалістю зберігання історичних значень.

3. Таблиці тригерів і умов:

Для оцінки результатів перевірок у базі створюється таблиця triggers, яка відповідає за формулювання умов спрацювання (логічних виразів), які базуються на значеннях одного або декількох елементів даних.

Кожен тригер містить вираз, опис, рівень критичності, поточний статус та інші атрибути. За допомогою тригерів система аналізує, чи знаходиться значення параметра в межах норми. При виявленні відхилення система створює подію.

Для зберігання типів умов (операторів порівняння) використовується довідкова таблиця condition_types, яка містить перелік можливих операторів: "більше", "менше", "дорівнює", "не дорівнює" тощо.

4. Таблиця подій

Таблиця events зберігає усі зафіксовані системою події – це записи про спрацювання тригерів. Події мають дату і час виникнення, посилання на тригер, який їх викликав, опис, статус (активна, вирішена) та рівень критичності. Ця таблиця є основою для побудови журналу подій та формування сповіщень.

Кожна подія — це не просто запис про інцидент, а ключовий об’єкт, який фіксує момент часу, що сигналізує про зміну стану системи. Після усунення причини подія автоматично переходить у стан «вирішено».

5. Таблиці сповіщень

Функціональність автоматичного інформування реалізується через таблицю alerts, яка зберігає інформацію про надіслані або заплановані сповіщення. Для кожного повідомлення фіксується, кому воно адресоване, яким способом доставлене (email, push), дата та час надсилання, пов’язана подія, а також статус доставки (успішно, помилка).

Сповіщення є важливою складовою системи, яка дозволяє оперативно реагувати на критичні події, навіть якщо користувач не перебуває в інтерфейсі програми.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Прикладне програмне забезпечення (ППЗ) системи моніторингу та керування мережевими ресурсами є ключовим компонентом у загальній архітектурі проєкту. Воно виконує основну функцію збирання, обробки, збереження та візуалізації інформації про стан мережевих об’єктів, а також забезпечує механізми реагування на виникнення різноманітних подій у мережі. Правильна організація структури ППЗ є запорукою його ефективної роботи, масштабованості, гнучкості та зручності подальшої підтримки.

З огляду на особливості функціонування та вимоги до системи, ППЗ розроблено за модульним принципом, що дозволяє розділити його на окремі складові частини – пакети. Кожен з цих пакетів відповідає за певну частину логіки системи, що підвищує зрозумілість архітектури та сприяє полегшенню подальшої розробки і модернізації системи.

Основними складовими частинами програмного забезпечення є:

- Інтерфейс користувача:

Інтерфейс користувача є тим компонентом, з яким безпосередньо взаємодіє користувач системи. Він надає зручні засоби для моніторингу стану мережевих пристроїв, налаштування параметрів системи, перегляду повідомлень про події та формування різноманітних звітів. Для реалізації інтерфейсу використовується веб-технологія, що забезпечує доступність системи з будь-якого пристрою, підключеного до мережі. Використання веб-сервера Nginx дозволяє організувати швидку, стабільну та безпечну роботу веб-інтерфейсу.

- Модуль роботи з базою даних:

Цей пакет відповідає за взаємодію з реляційною базою даних, в якій зберігається вся необхідна інформація про мережеві пристрої, їх типи, налаштування моніторингу, історія подій, сповіщення, тригери, а також інші параметри, важливі для функціонування системи. Для реалізації зберігання та обробки даних використовується сучасна система керування базами даних

PostgreSQL, яка забезпечує надійність, цілісність даних, а також масштабованість системи при зростанні обсягу інформації.

- Обробка даних і логіка моніторингу (:

Цей компонент відповідає за збір інформації з мережевих пристроїв, її аналіз та обробку. Він реалізує механізми контролю стану ресурсів за допомогою встановлених правил, які перевіряють відповідність поточних показників заданим порогам (тригерам). У разі виявлення відхилень чи аномалій система генерує події та повідомлення, які надсилаються відповідальним особам для оперативного реагування. Модуль також виконує агрегування даних, що дозволяє аналізувати тенденції і формувати статистичні звіти.

Отже, обрана організаційна структура програмного забезпечення забезпечує розділення функціональних обов’язків між окремими модулями, що підвищує загальну надійність та ефективність роботи системи. Водночас така архітектура дозволяє легко впроваджувати нові функції, підтримувати існуючі модулі і адаптувати систему під зміни в мережевому середовищі.

3.2 Вибір інструментарію для створення ППЗ

Процес розробки прикладного програмного забезпечення вимагає ретельного підходу до вибору інструментальних засобів. Вибір мови програмування та середовища розробки є важливим етапом, що безпосередньо впливає на ефективність розробки, продуктивність роботи програмного забезпечення, а також зручність його підтримки й масштабування в майбутньому.

Необхідно взяти до уваги, що вибір інструментарію для реалізації програмного забезпечення був здійснений на основі технічних вимог проєкту та специфіки завдань, які вирішуються в рамках системи. Це відповідає принципам індивідуального підходу до проєктування програмних рішень, згідно з яким кожен розробник самостійно обирає найбільш зручні та доцільні засоби для реалізації поставлених цілей.

Вибір інструментів став критично важливим фактором, який суттєво

вплинув на загальну архітектуру системи, організацію командної взаємодії, забезпечення безпеки та ефективність використання ресурсів.

Загальна архітектура програмного забезпечення базується на принципах клієнт-серверної взаємодії, де серверна частина відповідає за збір, обробку, збереження даних та логіку сповіщень, а клієнтська частина – за інтерфейс користувача та відображення інформації.

Виходячи з цього методологія вибору базувалась за наступними критеріями:

- Функціональна достатність – вибрані засоби повинні повністю відповідати вимогам щодо реалізації необхідного функціоналу.
- Масштабованість – можливість розширення системи без істотної перебудови її компонентів.
- Продуктивність – забезпечення високої швидкодії при роботі з великими обсягами даних.
- Надійність і безпека – стійкість до збоїв, захист від зовнішніх атак, підтримка ролей доступу.
- Документованість та підтримка – наявність офіційної документації, активних спільнот, часті оновлення.
- Простота розгортання і підтримки – швидка адаптація на різних серверах та можливість автоматизації встановлення.

На основі аналізу вищевказаних критеріїв було сформовано остаточний стек технологій, який використано у проєкті. У наступних підрозділах наводиться розгорнуте обґрунтування кожного інструменту, мови програмування, середовища розробки та допоміжних технологій.

3.2.1 Середовище розгортання

Однією з ключових передумов створення стабільної та незалежної інфраструктури є вибір операційної системи та платформи для хостингу усіх компонентів системи. У межах цієї роботи як базову було обрано операційну систему Linux (дистрибутив Ubuntu Server), що дозволяє реалізувати повністю безкоштовне, гнучке та надійне серверне середовище з можливістю тонкого

налаштування [7].

Платформа Linux надає доступ до великої кількості системних інструментів, працює стабільно навіть на відносно слабкому обладнанні, а також чудово інтегрується з іншими сучасними засобами розгортання, зокрема Docker.

Для контейнеризації всіх компонентів системи було обрано технологію Docker. Контейнеризація дозволяє запакувати кожну частину системи (наприклад, веб-інтерфейс, базу даних, бекенд, моніторингові служби) у власне ізольоване середовище, яке не залежить від конфігурації основної системи. Це дає змогу гарантувати однакову поведінку програмного забезпечення незалежно від того, на якому сервері чи хості воно розгортається [1].

Контейнери Docker є легкими, швидкими у запуску та простими в обслуговуванні. Вони запускаються з готових образів, у яких вже містяться всі необхідні залежності. У ході роботи над проєктом було створено власні Dockerfile-файли, які описують, як зібрати потрібне середовище для кожної служби.

Крім самих контейнерів, для координації їхньої роботи було використано утиліту Docker Compose. Цей інструмент дозволяє описати конфігурацію всієї багатокомпонентної системи у єдиному YAML-файлі. В ньому задається, які саме служби необхідно запустити, які порти мають бути відкриті, які змінні середовища використовуються, як сервіси пов’язані між собою (наприклад, веб-інтерфейс підключається до бази даних за іменем postgres).

Завдяки Docker Compose ініціалізація системи зводиться до виконання однієї команди – docker-compose up – після чого вся інфраструктура автоматично запускається. Це значно прискорює розгортання на новому сервері, спрощує налагодження та дає змогу повторно використовувати ту саму конфігурацію в майбутньому.

До основних переваг такого підходу можна віднести:

- Відокремлення середовищ: кожна служба працює у своєму контейнері;
- Повторюваність: розгортання завжди відбувається однаково;
- Портативність: систему легко перенести з одного сервера на інший;

- Простота обслуговування: оновлення та відновлення конфігурації займає мінімум часу;
- Масштабованість: можна підняти декілька екземплярів одного сервісу за потреби.

Таким чином, вибір Linux + Docker + Docker Compose як базового інструментарію для хостингу системи є технічно обґрунтованим, сучасним і таким, що забезпечує гнучкість, безпеку та готовність до розширення в майбутньому.

3.2.2 Система управління базами даних

Збереження, обробка та цілісність даних у системі моніторингу відіграють ключову роль. Враховуючи складність структури інформаційної бази, наявність великої кількості таблиць із логічними зв’язками, а також потребу у стабільному зберіганні як поточних, так і історичних даних, було прийнято рішення використовувати реляційну систему керування базами даних PostgreSQL.

PostgreSQL — це одна з найпотужніших сучасних СУБД з відкритим кодом, яка активно розвивається і підтримується спільнотою. Вона поєднує в собі класичні реляційні можливості, повну підтримку SQL-стандартів, а також численні розширення, включаючи роботу з JSON-типами, CTE-запити, підтримку збережених процедур, зовнішніх ключів, перевірок цілісності та тригерів [2].

Причини вибору PostgreSQL:

- Підтримка складних зв’язків між таблицями, що важливо для структури з багатьма зовнішніми ключами;
- Можливість виконання великої кількості одночасних запитів без суттєвої втрати продуктивності;
- Вбудовані механізми захисту від конкурентного доступу;
- Простота інтеграції з Docker та використання готових офіційних образів;

- Повноцінна підтримка транзакцій, що важливо при оновленні одразу кількох таблиць;
- Висока надійність та масштабованість (до сотень мільйонів записів).

У межах проєкту база даних PostgreSQL була розгорнута у вигляді окремого контейнера з використанням офіційного образу postgres. Контейнер отримував окремий том з постійним зберіганням даних (volume), що гарантує збереження всієї інформації навіть у разі оновлення або перезапуску контейнера. Налаштування доступу до СУБД реалізоване через змінні середовища у docker-compose.yml, включаючи встановлення пароля, назви бази та користувача за замовчуванням.

Взаємодія між службами системи та базою даних здійснюється через мережевий інтерфейс Docker, що забезпечує безпечне внутрішнє з'єднання. Для обміну даними застосовуються SQL-запити, які реалізуються через мову програмування інтерфейсу.

Особливу увагу в системі приділено забезпеченню цілісності даних. Це реалізовано за допомогою таких інструментів:

- Первинні ключі (PRIMARY KEY) — унікально ідентифікують записи;
- Зовнішні ключі (FOREIGN KEY) — забезпечують логічну зв’язаність між таблицями;
- NOT NULL — запобігає відсутності обов’язкових значень;
- CHECK — контроль діапазону допустимих значень (наприклад, статус 0 або 1);
- Унікальні обмеження (UNIQUE) — для логінів користувачів, IP-адрес пристроїв тощо.

Також передбачено індексування полів, які найчастіше використовуються в запитах: це item_key, event_time, trigger_id, host_id, name, email тощо. Наявність індексів дозволяє суттєво зменшити час вибірки даних, особливо в тих таблицях, які містять десятки або сотні тисяч записів (наприклад, history).

Перевагою PostgreSQL є і підтримка масштабованої архітектури: у разі розширення функціоналу можливо реалізувати реплікацію, створити окремі бази

для архіву, або використовувати партиціювання таблиць для підвищення швидкодії.

На етапі розробки схема бази створювалась вручну, з поступовим уточненням структури згідно логічної моделі. Створення таблиць реалізовано за допомогою SQL-команд CREATE TABLE, які автоматично виконуються під час ініціалізації контейнера, що спрощує перенесення системи на інші сервери або її клонування.

Завдяки використанню PostgreSQL як основної СУБД вдалося забезпечити надійне, стабільне та швидкодійне функціонування інформаційної бази системи, а також створити основу для подальшої аналітики та історичного аналізу даних.

3.2.3 Сервер доступу та маршрутизації

Важливою складовою будь-якої сучасної веб-орієнтованої інформаційної системи є організація взаємодії між користувачем та серверною частиною. У проєктованій системі моніторингу взаємодія користувача з системою відбувається через веб-інтерфейс, який звертається до серверної частини (API, бази даних та інших служб) через захищене HTTP-з’єднання.

Для реалізації надійної маршрутизації запитів, а також для доставки клієнтського інтерфейсу (веб-сторінок, скриптів, стилів) було обрано веб-сервер Nginx. Це сучасне програмне забезпечення з відкритим кодом, що широко використовується у промислових і корпоративних системах. Його популярність зумовлена високою продуктивністю, невеликим споживанням ресурсів, простотою налаштування та широкими можливостями масштабування [3].

Серед основних причин, чому саме Nginx було обрано для цього проєкту:

- Підтримка високої кількості одночасних з’єднань без навантаження на систему;
- Надійне проксування запитів до внутрішніх API або бекенд-сервісів;
- Можливість легко налаштувати HTTPS, редиректи, кешування;
- Простота конфігурації — файли конфігурації Nginx зрозумілі й легко адаптуються під конкретні потреби;
- Активна підтримка і регулярні оновлення безпеки.

Розгортання Nginx також реалізовано у вигляді окремого контейнера Docker. Він працює разом з іншими службами в рамках спільної мережі docker-compose, що дозволяє серверам бачити одне одного за іменами (наприклад, бекенд-додаток може бути доступний за ім’ям api:5000).

Конфігурація Nginx описана у файлі nginx.conf, що копіюється до контейнера під час його запуску.

Оскільки вся система контейнеризована, немає потреби відкривати порти всіх сервісів. Зовнішній доступ надається лише до порту веб-сервера, який далі самостійно пересилає запити до потрібного контейнера. Це не тільки підвищує безпеку, але й спрощує адміністрування – у випадку змін достатньо оновити лише конфігурацію Nginx.

Для зручності в майбутньому до Nginx легко додати:

- SSL-сертифікати (через Let’s Encrypt або вручну);
- базову автентифікацію користувачів на рівні сервера;
- логування доступу з фільтрацією;
- обмеження запитів, що захищає від атак типу brute-force.

Таким чином, Nginx був обраний як універсальний та легкий інструмент, який забезпечує стабільну маршрутизацію, дозволяє обробляти всі запити до інтерфейсу та бекенду, а також виступає єдиною точкою входу у систему моніторингу, що відповідає сучасним принципам розробки масштабованих веб-систем.

3.2.4 Клієнтська частина

Клієнтська частина є невід’ємним елементом програмного забезпечення системи моніторингу та керування мережевими ресурсами. Саме вона забезпечує взаємодію користувача з системою, дозволяє переглядати інформацію про пристрої, слідкувати за станом елементів моніторингу, аналізувати події та отримувати оперативні сповіщення.

У розроблюваній системі клієнтський інтерфейс реалізовано на основі класичного підходу динамічної генерації сторінок. Для цього було обрано мову PHP. Завдяки цьому HTML-розмітка формується на сервері у момент обробки

запиту. Така архітектура дозволяє зручно організовувати відображення контенту в залежності від ролі користувача, параметрів сесії, наявності подій, параметрів фільтрації тощо.

Використання PHP як серверної мови для формування веб-інтерфейсу обумовлено кількома факторами:

- простота інтеграції з HTML та JavaScript;
- можливість безпосередньо працювати з базою даних за допомогою SQL-запитів;
- підтримка шаблонів та повторного використання фрагментів коду;
- розділення логіки та відображення, що полегшує обслуговування;
- повна сумісність з веб-сервером Nginx.

У структурі інтерфейсу PHP-файли відповідають за логіку: перевірку доступу, вибір даних з бази, обробку форм, генерацію HTML. Сама HTML-структура включає типові елементи: заголовки, навігацію, віджети, таблиці, форми, індикатори та інші компоненти, з якими взаємодіє користувач.

Візуальне оформлення інтерфейсу реалізоване за допомогою CSS. Це дозволяє забезпечити цілісність дизайну, змінювати зовнішній вигляд системи незалежно від логіки, реалізовувати теми та підтримувати адаптивність. Сторінки відображаються коректно на більшості сучасних браузерів, з урахуванням особливостей мобільних пристроїв та різної роздільної здатності екранів.

Інтерфейс системи є повноцінним веб-застосунком, який об'єднує серверну логіку PHP, клієнтську взаємодію JavaScript, візуальне оформлення CSS та адаптивну HTML-структуру. Він є зручним, легким у сприйнятті та повністю готовим до розширення або модифікації у майбутньому.

3.2.5 Серверна частина

Серверна частина інформаційної системи виконує роль основного логічного ядра, яке обробляє запити від клієнтського інтерфейсу, виконує взаємодію з базою даних, здійснює перевірку прав доступу, опрацьовує події, формує відповіді та реалізує інші ключові функції системи. Саме тому вибір

мови програмування для реалізації серверного коду є одним із найвідповідальніших рішень у процесі створення прикладного програмного забезпечення, виходячи з цього була обрана мова програмування C.

Мова C – це низькорівнева мова, яка дозволяє писати максимально ефективний і оптимізований код. Вона є однією з найбільш продуктивних мов, широко застосовується в системному програмуванні, зокрема при розробці сервісів, агентів збору даних та серверів обробки.

Серед основних переваг вибору мови C:

- висока швидкодія, що критично важливо для опрацювання великої кількості елементів моніторингу в режимі реального часу;
- контроль над пам’яттю, що дозволяє уникати надлишкових витрат ресурсів;
- стабільна робота за умов цілодобового навантаження;
- можливість прямої взаємодії з мережею, файловою системою, сокетами, бібліотеками операційної системи;
- сумісність із Unix-подібними системами (зокрема Linux), які використовуються у проєкті як серверна платформа.

Серверна частина виконується як окремий процес (демон), що запускається разом із системою. Його функції включають: періодичне опитування пристроїв, фіксацію значень у базі даних, оцінку умов спрацювання тригерів, створення подій, а також логування усіх дій системи. Взаємодія з іншими модулями системи відбувається через стандартні мережеві протоколи та локальні API.

Оскільки мова C не є скриптовою, код компілюється заздалегідь. Це гарантує стабільність та передбачуваність виконання програми, що особливо важливо в умовах безперервного функціонування системи моніторингу.

Таким чином, обрання мови C для реалізації серверного ядра системи є технічно обґрунтованим рішенням, яке базується на потребі досягнення максимальної ефективності, стабільності та низького споживання ресурсів у серверному середовищі.

3.2.6 Серверна частина

У процесі створення програмного забезпечення важливу роль відіграє не лише вибір мов програмування чи технологій, але й правильно підібране середовище розробки. Воно визначає, наскільки зручно і ефективно відбувається написання коду, тестування, налагодження, налаштування конфігурацій, а також робота з системою загалом. У рамках цього проєкту для реалізації різних компонентів було використано низку програмних інструментів та середовищ, які в сукупності утворили повноцінний і зручний стек розробки.

Вся розробка здійснювалась на базі операційної системи Linux, він є стабільною, безкоштовною та відкритою платформою, яка широко використовується в серверних рішеннях і дозволяє гнучко налаштовувати середовище, встановлювати необхідні пакети, створювати скрипти, управляти процесами тощо. Саме Linux є найбільш природним середовищем для роботи з Docker, а також для розгортання серверних систем моніторингу.

Для редагування коду, налаштування конфігураційних файлів, написання SQL-запитів та стилів було використано редактор Visual Studio Code (VS Code). Це кросплатформне середовище розробки, яке підтримує автодоповнення, перевірку синтаксису, розширення для роботи з Docker, PostgreSQL, HTML, PHP, JSON та багатьма іншими технологіями. Також слід відмітити серед переваг даного вибору:

- підтримка роботи з віддаленими контейнерами;
- інтеграція з Git, що дозволяє вести контроль версій;
- гнучке налаштування під себе: тема, шрифт, розкладка, додаткові плагіни;
- вбудований термінал для роботи з Linux-командами прямо в редакторі;
- можливість відкриття декількох проєктів одночасно.

У випадках, коли редагування виконувалося безпосередньо всередині контейнера, використовувалися термінальні редактори, зокрема nano та vim.

Для розгортання, тестування та модифікації програмного середовища

використовувалась система Docker. Всі компоненти такі як: база даних, сервер, інтерфейс, веб-сервер, запускались у вигляді окремих контейнерів, об’єднаних мережею через docker-compose. Це дозволяло локально і швидко тестувати будь-які зміни в коді, перевіряти роботу API, спостерігати за логами в реальному часі та змінювати конфігурацію системи без перезапуску фізичного сервера що значно сповільнювало б розробку системи.

Перевагами такого підходу є:

- можливість "гарячого" оновлення окремих сервісів;
- контроль над версіями компонентів;
- ізоляція середовища, що запобігає конфліктам залежностей;
- спрощення процесу розгортання та масштабування.

Для тестування веб-інтерфейсу системи використовувався браузер Google Chrome. Він забезпечує розширену підтримку інструментів розробника (DevTools), зокрема:

- консоль JavaScript;
- перегляд запитів (Network);
- емуляція мобільних пристроїв;
- перевірка стилів та макетів (CSS Inspector).

Також DevTools дозволяли відслідковувати помилки JavaScript, переглядати заголовки запитів, токени авторизації та аналізувати динамічну поведінку інтерфейсу.

Для роботи з контейнерами та налаштування системи активно використовувались такі утиліти:

- docker, docker-compose – запуск, оновлення, перегляд логів;
- systemctl – керування службами в середовищі Linux;
- curl та wget – для перевірки доступності API та завантаження ресурсів;
- psql – консольна робота з базою даних PostgreSQL;
- netstat – аналіз портів і з’єднань;
- journalctl та tail – перегляд логів у реальному часі.

Середовище розробки, яке використовувалося у цьому проєкті, було побудоване на сучасних, гнучких та відкритих інструментах. Воно забезпечувало максимальну ефективність на всіх етапах розробки – від написання коду до тестування та запуску. Такий підхід дозволив зосередитися безпосередньо на реалізації функціоналу системи, мінімізуючи технічні труднощі, та сформував основу для подальшої підтримки і розвитку програмного забезпечення.

3.3 Алгоритмізація та програмування програмних модулів

Розробка прикладного програмного забезпечення завжди передбачає побудову логічної структури програмних модулів, що забезпечують виконання окремих функціональних задач. На етапі реалізації програмної частини системи моніторингу та керування мережевими ресурсами виникла потреба розділити функціонал на окремі логічні компоненти, кожен з яких виконує певну роль в загальному циклі роботи системи.

Враховуючи організаційну структуру програмного забезпечення, до основних модулів, що потребували алгоритмізації, належать: модуль збору та запису значень елементів моніторингу, модуль перевірки умов (тригерів), модуль створення подій, система формування сповіщень, а також механізм відображення інформації в інтерфейсі користувача.

Одним із ключових процесів у системі є обробка нового значення елемента моніторингу. Цей процес запускається автоматично відповідно до заданого інтервалу опитування. Після отримання нового значення система має зберегти його у базі даних, а також перевірити, чи не викликає це значення спрацювання пов’язаних тригерів. Якщо певна умова справджується, генерується відповідна подія, яка згодом може спричинити формування сповіщення для адміністратора.

Для кращого розуміння логіки роботи системи нижче (рис. 3.1) подано загальну блок-схему алгоритму обробки нового значення параметра моніторингу:

Рис. 3.1 Блок-схема алгоритму обробки нового значення

На основі вищезазначеного алгоритму було реалізовано відповідний програмний модуль, який періодично запускається у фоновому режимі. В залежності від конфігурації, він опрацьовує дані, отримані з SNMP-вузлів, та виконує подальшу логіку [8].

Нижче наведено фрагмент умовного псевдокоду, який демонструє загальну послідовність дій:

```
function process_item_value(item_id, value):
timestamp = get_current_time()
insert_into_history(item_id, value, timestamp)
triggers = get_triggers_for_item(item_id)
for trigger in triggers:
    if evaluate_expression(trigger.expression, value):
        if trigger.status != 'PROBLEM':
            create_event(trigger.id, timestamp, 'PROBLEM')
```

```
            update_trigger_status(trigger.id, 'PROBLEM')
        else:
            if trigger.status == 'PROBLEM':
                create_event(trigger.id, timestamp, 'OK')
                update_trigger_status(trigger.id, 'OK')
```

У реальному виконанні ця функція є частиною основного демона, який працює в середовищі Linux. Він реалізований на мові C, що забезпечує високу швидкодію та стабільність. Код працює з системним часом, опрацьовує SQL-запити, використовує буфери пам’яті для тимчасового збереження значень, та за потреби записує інформацію у журнали логів.

Наступним важливим модулем є перевірка умов спрацювання тригерів. У системі передбачено, що кожен елемент моніторингу може мати пов’язаний один або кілька тригерів, які оцінюють отримані значення згідно з заданими умовами. Це може бути як проста перевірка на перевищення порогу («значення > 90»), так і складні логічні вирази з урахуванням кількох параметрів. У разі істинності виразу система створює подію, яка записується у таблицю events, змінює статус тригера та активує механізм сповіщення.

Таким чином, кожен логічний компонент системи було детально проаналізовано, алгоритмізовано та реалізовано у вигляді програмного модуля. Такий підхід дозволив забезпечити цілісність структури, незалежність компонентів і простоту у супроводі. Надалі реалізація нових модулів або розширення існуючих може здійснюватися без потреби у зміні базової архітектури, що свідчить про масштабованість і правильну побудову структури програмного забезпечення.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМ

4.1 Тестування системи

Завершальним етапом розробки прикладного програмного забезпечення є тестування, яке відіграє ключову роль у забезпеченні якості, надійності та відповідності створеної системи початковим вимогам. У загальному розумінні, тестування – це процес перевірки програмного продукту з метою виявлення помилок, оцінки його стабільності та впевненості в правильності роботи окремих функціональних компонентів. У контексті розробки системи моніторингу та керування мережевими ресурсами тестування відіграє надзвичайно важливу роль, оскільки система працює у реальному часі, фіксує зміни в станах мережі та генерує події, на які користувач повинен реагувати оперативно.

Проведення тестування дозволяє не лише виявити програмні помилки, а й оцінити ефективність взаємодії користувача з інтерфейсом, виявити логічні суперечності у формулюванні тригерів, перевірити коректність взаємодії між окремими компонентами системи та впевнитися в тому, що база даних виконує всі функції зберігання, фільтрації та агрегації інформації.

Загальна методика тестування передбачала поетапну перевірку окремих модулів, що відповідають за: збір даних, збереження інформації, оцінку тригерів, формування подій, відображення інформації в інтерфейсі та надсилання сповіщень. У рамках цієї роботи було використано функціональне тестування — тобто перевірка роботи системи відповідно до визначених функціональних вимог.

Слід зазначити, що функціональне тестування є одним із базових видів тестування програмних систем. Воно полягає у подачі певного набору вхідних даних до системи та спостереженні за тим, чи відповідають отримані результати очікуваним. У цьому випадку тестувались як типові сценарії (нормальні значення параметрів), так і граничні умови (вихід за межі допустимих значень), а також реакція системи у разі виникнення помилки зв’язку або втрати

доступності пристрою.

Для проведення тестування було створене спеціальне тестове середовище, що відображає реальні умови розгортання. Система була запущена на окремому сервері з операційною системою Linux (Ubuntu), у середовищі Docker з використанням інструменту docker-compose. Це дозволило відтворити ізольовану конфігурацію системи, незалежну від середовища розробки.

Насамперед була перевірена авторизація (рис. 4.1)

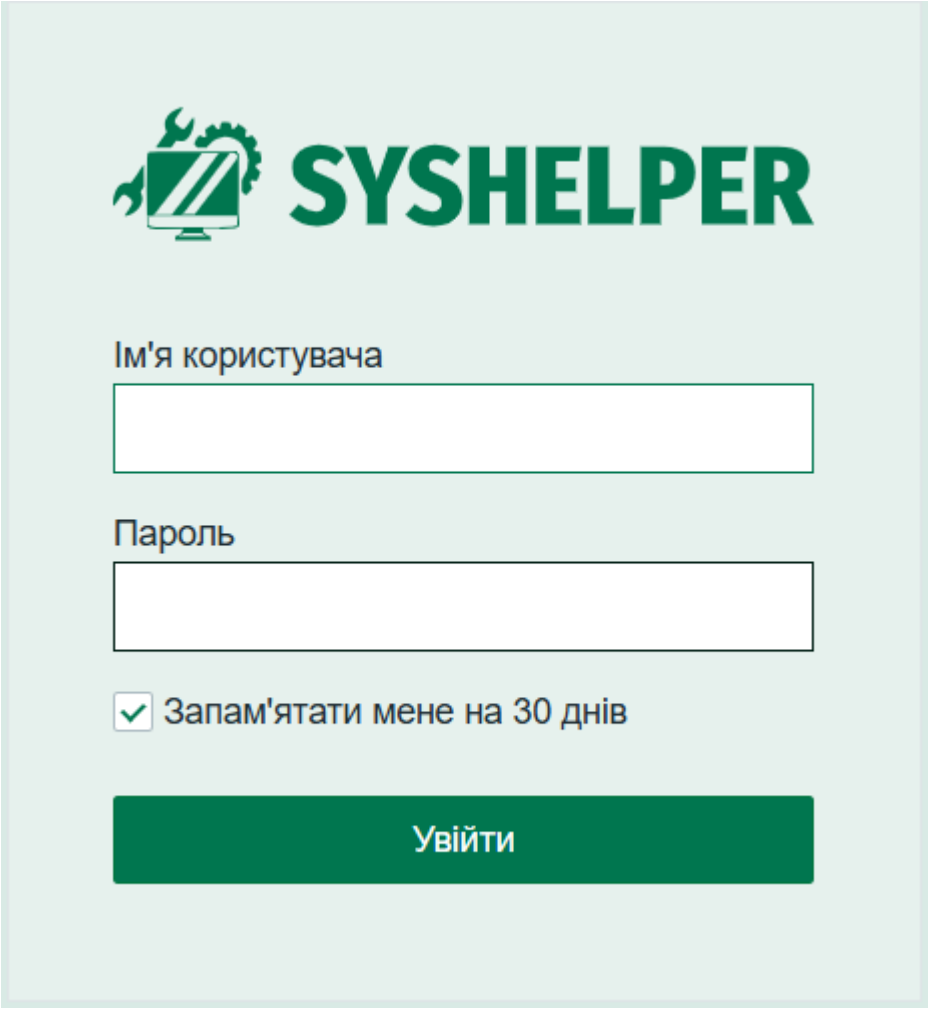

Рис. 4.1 Авторизація

Спершу було протестовано можливість додавання вузлів мережі (рис. 4.2) де також вказано основний протокол для моніторингу.

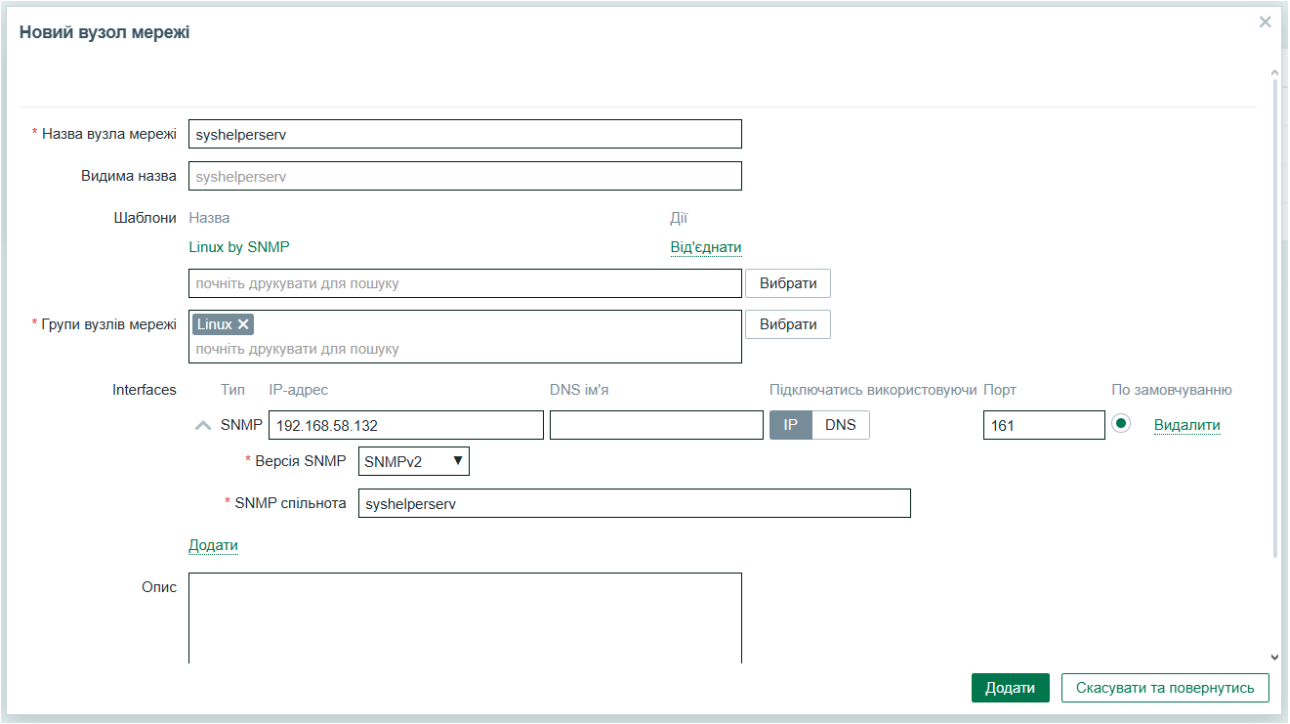

Рис. 4.2 Вікно додавання нового пристрою

На наступному етапі проводилось наповнення бази даних тестовими записами. Зокрема, було додано вузол мережі, та додано до нього набір елементів моніторингу (рис. 4.3). До частини з цих елементів були прив’язані тригери, які повинні реагувати на зміну значення понад певний поріг.

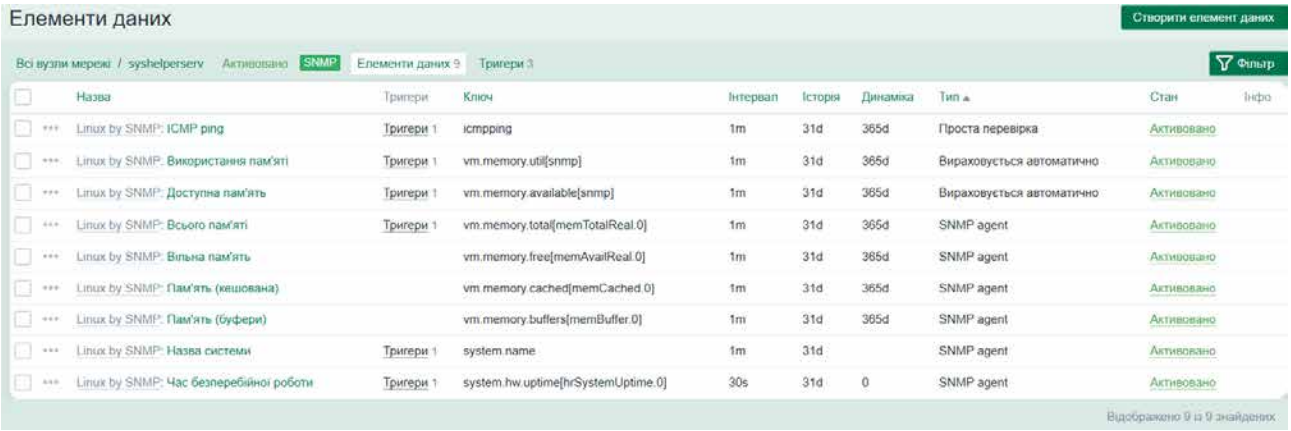

Рис. 4.3 Сторінка елементів даних

Після цього вручну та автоматично імітувались ситуації, коли значення змінюються, тригер спрацьовує, створюється подія, і система повинна відреагувати на неї відповідним чином. Наприклад, якщо значення ICMP ping перевищувало встановлений поріг, система створювала запис у таблиці подій (рис. 4.4), оновлювала стан тригера, та змінювала індикатор у веб-інтерфейсі.

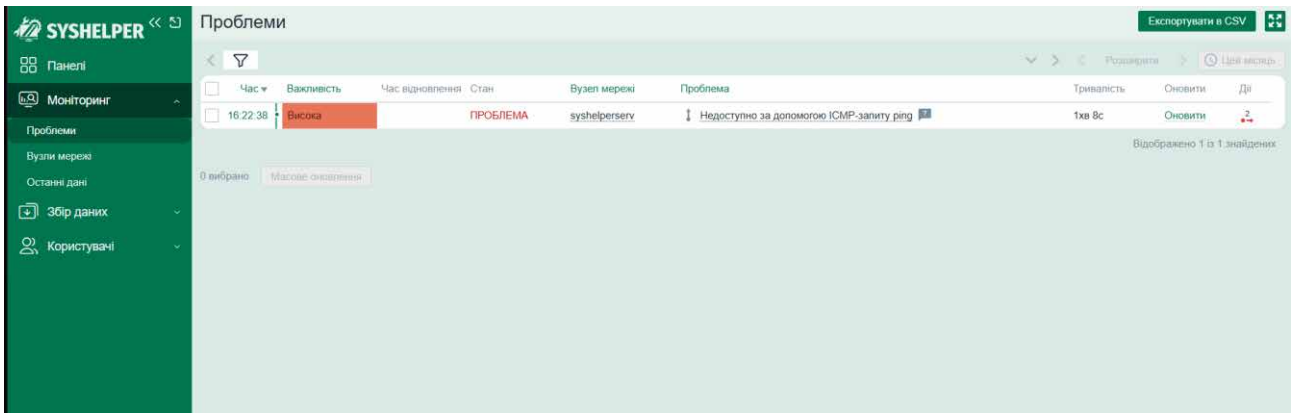

Рис. 4.4 Сторінка «Проблеми»

Окремо перевірялась робота інтерфейсу: доступність усіх вкладок, правильність відображення даних, можливість фільтрації за періодом, пристроєм, тригером. Здійснювалась перевірка сортування, відповідності кольорової індикації станам (нормальний/попередження/проблема), і загального зручності навігації.

Крім перевірки логіки створення подій та сповіщень, окрему увагу було приділено роботі з панелями моніторингу (рис. 4.5). У системі передбачена можливість перегляду агрегованої інформації про стан пристроїв, графічне відображення динаміки параметрів, а також використання віджетів для виведення статистики в режимі реального часу. У процесі тестування було створено декілька тестових панелей з графіками використання пам'яті. Перевірялась швидкість оновлення даних, коректність відображення, відповідність значень поточному стану системи. Інтерфейс панелей виявився зручним та адаптивним: користувач може самостійно обрати потрібні елементи для відображення, змінювати періоди спостереження та сортувати дані за пріоритетом.

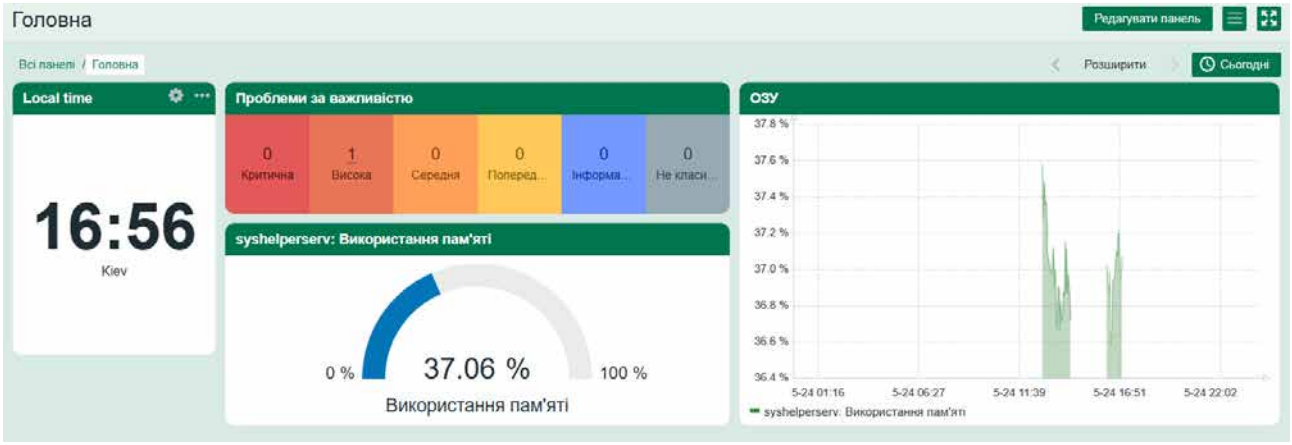

Рис. 4.5 Сторінка панелей моніторингу

У результаті тестування можна зробити висновок, що система працює стабільно, функціонально повноцінно та відповідає вимогам, визначеним на етапі постановки задачі. Проведені перевірки дозволяють стверджувати, що система готова до впровадження в умовах практичного використання на підприємствах чи організаціях, де існує потреба у цілодобовому моніторингу мережевої інфраструктури.

4.2 Вимоги до апаратного та програмного забезпечення

Ефективне функціонування будь-якої програмної системи значною мірою залежить від правильного вибору апаратного та програмного забезпечення, яке забезпечує роботу її основних компонентів. Успішне впровадження системи моніторингу та керування мережевими ресурсами можливе лише за умови чіткого визначення вимог до серверного обладнання, мережевої інфраструктури, клієнтських пристроїв, а також операційних систем, середовищ виконання та додаткових служб.

Апаратні ресурси системи моніторингу не є самоціллю, а розглядаються як інструмент, який повинен гарантувати якісне виконання функцій системи. До прикладу, якщо система призначена для моніторингу 20-30 мережевих пристроїв у межах невеликої організації, то вимоги до серверного обладнання можуть бути мінімальними — вистачить навіть віртуальної машини або офісного комп’ютера. Натомість, якщо система використовується для моніторингу сотень або навіть тисяч пристроїв у промисловому середовищі — необхідні сервери з потужними

процесорами, великим обсягом оперативної пам’яті та високошвидкісними SSD-дисками.

Окремо слід згадати про важливість мережевого середовища. Система моніторингу передбачає постійний обмін даними між вузлами — це можуть бути ICMP-запити, SNMP-збір, HTTP-з’єднання тощо. Навіть у межах однієї локальної мережі важливо враховувати затримки, втрати пакетів, перевантаження комутаторів або некоректну маршрутизацію, які можуть впливати на достовірність отриманих даних. Тому важливо, щоб сервер моніторингу мав стабільне мережеве підключення, бажано гігабітне або вище, з мінімальним джитером.

Не менш важливою є вибір операційної системи та підтримуваного програмного середовища. У сучасних умовах найкращою практикою є використання вільного програмного забезпечення (open-source), оскільки це знижує вартість володіння системою, дозволяє гнучко модифікувати компоненти, а також забезпечує прозорість та керованість процесів. Саме тому як основна операційна система для розгортання була обрана Ubuntu Server — один з найпопулярніших та стабільних дистрибутивів Linux, що має широку підтримку спільноти, добре працює у середовищі Docker і має активну документацію.

Щодо контейнеризації — використання Docker дозволяє абстрагуватись від конкретного обладнання. Після створення правильного образу програма може бути запущена практично на будь-якому сервері або хмарній платформі — без перевстановлення компонентів, залежностей чи повторного налаштування. Це відкриває перспективу масштабування, створення резервних копій, гнучкого оновлення без зупинки роботи всієї системи.

Значна увага приділялась і вибору СУБД. PostgreSQL було обрано не лише за стабільність та функціональність, але й за здатність працювати з великим обсягом структурованої інформації. У випадку систем моніторингу це має особливу вагу, оскільки щосекундне або щохвилинне опитування навіть 10 пристроїв швидко формує десятки тисяч записів. Якщо ці записи не будуть

оптимально оброблені та збережені — система втратить актуальність і перестане бути інструментом контролю.

Загалом, правильно підібране апаратне й програмне забезпечення — це основа для надійної роботи будь-якої системи. Але навіть потужне обладнання не гарантує успіху без грамотної архітектури, розумного підходу до конфігурації та регулярного моніторингу самої системи моніторингу.

У типовому випадку система складається з таких вузлів:

- Сервер моніторингу: основний хост, на якому запущено всі сервіси системи у контейнерах Docker. Він обробляє запити, формує події, зберігає дані та відповідає за сповіщення.
- База даних: контейнер PostgreSQL, який відповідає за зберігання історичних значень, налаштувань, користувачів, тригерів, логів.
- Веб-інтерфейс: реалізований у вигляді PHP-скриптів, доступних через веб-сервер Nginx. Забезпечує взаємодію користувача з системою.
- Клієнтські пристрої: користувачі, які працюють із системою через браузер (на робочих ПК, ноутбуках, планшетах).
- Об’єкти моніторингу: реальні пристрої (сервера, роутери, принтери, точки доступу), які відповідають на запити або надають інформацію через SNMP, ICMP або інші протоколи.

Апаратні вимоги

Сервер моніторингу:

- Процесор: не менше ніж 2 ядра (рекомендовано від 4 ядер, якщо понад 100 пристроїв).
- Оперативна пам’ять: мінімум 4 ГБ ОЗП (рекомендовано 8 ГБ і більше).
- Дисковий простір: не менше ніж 50 ГБ SSD або HDD (у разі зберігання довготривалих історій бажано від 100 ГБ).
- Мережева карта: 100 Мбіт/с або гігабіт (в залежності від трафіку).

База даних (PostgreSQL):

- Може бути встановлена на тому ж сервері або окремо.
- ОЗП: бажано 2 ГБ виділено під PostgreSQL.

- Диск: швидке сховище (SSD рекомендовано) — база часто читає/пише.

Клієнтські пристрої:

- Будь-який сучасний ПК або ноутбук.
- Операційна система: Windows 10+, Linux, macOS.
- Браузер: Google Chrome, Mozilla Firefox, Microsoft Edge (актуальна версія).
- Дозвіл екрана: 1366×768 або вище.

Програмні вимоги:

- Серверне програмне забезпечення:
- Операційна система: Linux (Ubuntu 20.04 LTS або вище).
- Docker: версія 20.x або новіша.
- docker-compose: v1.29 або v2.
- Nginx: як зворотній проксі-сервер та сервер для статики.
- PHP: версія 7.4 або вище (якщо інтерфейс реалізований через PHP).
- PostgreSQL: версія 13 або вище.
- Утиліти: net-tools, curl, systemd, cron.

Додаткові вимоги:

- Порт 80/443 має бути відкритий для доступу до інтерфейсу.
- Порт 5432 (PostgreSQL) має бути доступний всередині docker-мережі.
- Бажано встановити засоби резервного копіювання (наприклад, pg_dump).

Запропонована система не вимагає надмірно високих ресурсів і може бути розгорнута на звичайному сервері з сучасними характеристиками. Водночас архітектура дозволяє масштабування, резервування та використання в більш складних середовищах. Завдяки використанню контейнеризації з Docker, система є портативною та легко переноситься на інші сервери без додаткових налаштувань. Під час експлуатації важливо забезпечити стабільну роботу операційної системи, регулярне резервне копіювання та моніторинг стану дисків і навантаження на базу даних.

Додатково слід зазначити, що під час визначення вимог до апаратного забезпечення необхідно враховувати не лише поточну кількість пристроїв для моніторингу, а й перспективу масштабування системи в майбутньому. Часто в процесі експлуатації виявляється потреба у розширенні функціоналу, підключенні нових вузлів або додаткових компонентів інфраструктури. Тому доцільно ще на етапі планування передбачити певний «запас потужності» - як в обчислювальних ресурсах, так і в обсязі пам’яті та сховища.

Слід також враховувати, що система моніторингу передбачає безперервну роботу у фоновому режимі. Це означає, що серверне програмне забезпечення повинне функціонувати стабільно протягом тривалого часу без перезавантажень або збоїв. У зв’язку з цим, рекомендується використовувати обладнання серверного класу або хоча б індустріальні компоненти з підвищеною надійністю, а також регулярно проводити профілактичні огляди стану дисків, вентиляторів, джерел живлення тощо.

Не варто ігнорувати питання захищеності операційної системи. Рекомендується використовувати стабільні, LTS-версії дистрибутивів Linux, які регулярно оновлюються. Доцільно також обмежити доступ до root-акаунта, налаштувати доступ по SSH лише за ключами, а також встановити базові засоби журналювання та контролю доступу. Такі дії не потребують значних ресурсів, але значно підвищують загальний рівень безпеки й стабільності всієї інфраструктури.

4.3 Склад інсталяційного пакету

Для того, щоб система моніторингу та керування мережевими ресурсами могла бути використана на іншому сервері або в умовах іншої організації, необхідно передбачити повноцінний інсталяційний пакет, який міститиме всі необхідні для її запуску компоненти. Інсталяційний пакет виконує роль універсального архіву, що включає конфігураційні файли, програмний код, опис структури бази даних, а також документацію для адміністратора. Його наявність забезпечує можливість швидкого, зручного та стандартизованого розгортання

системи на будь-якому сумісному сервері.

Згідно з архітектурною топологією системи, описаною у підрозділі 4.2, інсталяційний пакет повинен відповідати її структурі та передбачати окремі компоненти для кожного вузла. Основними вузлами виступають: сервер моніторингу, база даних, інтерфейсна частина (веб-сервер) та конфігураційна зона. Саме з урахуванням цього поділу й формувався склад інсталяційного пакету.

Структура інсталяційного пакету

Інсталяційний пакет є zip- або tar-архівом, який при розпакуванні створює структуровану папку з наступним вмістом:

```
syshelper-package/
├── docker-compose.yml
├── .env
├── /nginx/
│  └── nginx.conf
├── /php/
│  ├── Dockerfile
│  └── /src/
├── /postgres/
│  ├── init.sql (початкова структура БД)
│  └── Dockerfile
├── /backup/
│  └── empty (папка для майбутніх резервних копій)
```

Опис основних файлів та папок:

- docker-compose.yml – основний файл, що описує запуск усіх сервісів (nginx, php, postgres) в окремих контейнерах. Він забезпечує їх зв’язок у спільній мережі, налаштування портів, обсягу пам’яті та монтування томів.
- .env – шаблон файлу середовищних змінних, де адміністратор вказує значення для БД (логін, пароль), порти, назву томів, назву домену тощо.

- /nginx/nginx.conf – конфігурація веб-сервера nginx, яка містить маршрутизацію до інтерфейсу, проксирування запитів, кешування, обробку статичних ресурсів.
- /php/ - підкаталог, що містить Dockerfile для створення контейнера з PHP, а також повний набір файлів інтерфейсу: сторінки у форматах .php, .html, скрипти, стилі тощо.
- /postgres/init.sql – SQL-скрипт для ініціалізації бази даних: створення таблиць, користувачів, початкових параметрів. Виконується автоматично при першому запуску.
- /backup/ — резервна папка для збереження архівів бази даних або конфігурацій (створюється автоматично під час експлуатації).

Процес інсталяції системи:

1. Розпакування архіву на сервері з ОС Linux.
2. Перехід до папки проєкту, редагування файлу .env на основі прикладу.
3. Опціональне редагування конфігурацій nginx (наприклад, для зміни домену або порту).
4. Виконання команди docker-compose up -d для запуску системи.
5. Перевірка доступу до веб-інтерфейсу за IP-адресою або доменом.
6. Створення адміністратора в інтерфейсі або заповнення БД вручну через init.sql.
7. Система готова до роботи – проводиться початкове налаштування, створення пристроїв, елементів даних, тригерів.

Відповідність архітектурі

Оскільки інсталяційний пакет побудований за принципами контейнеризації, він відтворює усю архітектурну модель, описану в топології. Сервер моніторингу, база даних та інтерфейс запускаються як окремі ізольовані служби, що взаємодіють через внутрішню docker-мережу. Це дозволяє легко переносити систему на інші сервери, змінювати лише конфігураційні файли, не переписуючи код.

Оскільки інсталяційний пакет побудований за принципами контейнеризації, він відтворює усю архітектурну модель, описану в топології. Сервер моніторингу, база даних та інтерфейс запускаються як окремі ізольовані служби, що взаємодіють через внутрішню docker-мережу. Це дозволяє легко переносити систему на інші сервери, змінювати лише конфігураційні файли, не переписуючи код.

Переваги такого підходу:

- незалежність від середовища (працює на будь-якому Linux-сервері з Docker);
- автоматичне створення бази даних та ініціалізація конфігурацій;
- легке масштабування та оновлення;
- зручність для резервного копіювання (один docker volume або export dump);

Ретельно сформований інсталяційний пакет виконує роль централізованого засобу підготовки до інсталяції та розгортання всієї системи моніторингу. Його наявність значно спрощує процес розгортання, дозволяючи адміністраторам уникнути ручного встановлення окремих компонентів, залежностей, налаштувань конфігураційних файлів та виправлення типових помилок сумісності. Завдяки чітко структурованому складу інсталяційного пакету, який включає всі необхідні елементи – від вихідного коду до конфігурацій і документації – забезпечується повна автономність та самодостатність процесу встановлення.

Це означає, що для інсталяції не потрібно попередньо встановлювати окремі СУБД, веб-сервери чи середовища виконання мов програмування, оскільки все необхідне розгортається автоматично за допомогою docker-compose. Фактично, інсталяційний пакет діє як повноцінне середовище розробки та виконання одночасно, яке гарантує стабільність, передбачуваність та повторюваність результатів на різних хост-системах.

Важливо підкреслити, що пакет не лише дозволяє виконати первинне встановлення, а й підтримує подальшу експлуатацію, обслуговування, резервне

копіювання та оновлення. Контейнерна структура, яка лежить в основі архітектури системи, забезпечує логічну ізоляцію компонентів, завдяки чому зміни в одному з модулів (наприклад, оновлення версії PostgreSQL чи внесення правок в інтерфейс) не впливають на функціонування інших частин системи. Це значно підвищує гнучкість і знижує ризики порушення працездатності при оновленнях.

Також варто зазначити, що автономність інсталяційного пакету відкриває можливості для його використання у різних середовищах – як у локальній інфраструктурі підприємства (офісна мережа), так і у віртуалізованих або хмарних інфраструктурах (VPS, IaaS). Завдяки мінімальним вимогам до операційної системи та доступності Docker на більшості платформ, система може бути інстальована як на фізичний сервер, так і на віртуальну машину, що є досить значною перевагою.

Серед додаткових переваг інсталяційного пакету варто відзначити:

- Можливість швидкого відновлення: у разі збою основної системи адміністратор може повторно розгорнути систему на іншому сервері з того самого пакету, просто скопіювавши його на інший вузол;
- Готовність до масштабування: завдяки docker-структурі можна запускати окремі сервіси на різних хостах (наприклад, винос бази даних на окремий сервер), не змінюючи логіку роботи системи;
- Прозорість: весь код і конфігурації знаходяться у відкритому вигляді, що дозволяє здійснювати аудит безпеки, оптимізацію та глибоку кастомізацію відповідно до потреб організації.

Враховуючи викладене, можна стверджувати, що інсталяційний пакет не є лише набором файлів, а є складовою частиною архітектурного рішення, яке підвищує адаптивність, надійність і професіоналізм розробленої системи. Саме така форма доставки програмного забезпечення є сучасною практикою у сфері DevOps, підтримує стандарти CI/CD (безперервної інтеграції й розгортання) та відповідає сучасним підходам до життєвого циклу інформаційних систем.

4.4 Інформаційна безпека та захист даних

У сучасному інформаційному суспільстві питання безпеки є одним із найважливіших аспектів при розробці та експлуатації будь-яких програмних систем. Особливо це стосується тих інформаційних рішень, які працюють із серверним обладнанням, базами даних, конфіденційною інформацією та мають відношення до інфраструктури організації. Саме тому під час розробки системи моніторингу та керування мережевими ресурсами були враховані базові принципи інформаційної безпеки, що забезпечують належний рівень захисту на всіх етапах життєвого циклу системи.

Загалом, під терміном «інформаційна безпека» слід розуміти стан захищеності інформації, що обробляється у системі, від випадкового або навмисного впливу, який може призвести до її знищення, викривлення, несанкціонованого використання, копіювання, блокування доступу чи інших небажаних наслідків. Згідно з загальноприйнятими концепціями кібербезпеки, базовими принципами, які слід забезпечити, є конфіденційність (відсутність доступу для сторонніх), цілісність (незмінність даних без авторизованих дій) та доступність (можливість користування інформацією у визначений момент часу).

Система моніторингу, яку було реалізовано у межах цього проєкту, є системою, що безпосередньо взаємодіє з інфраструктурними елементами – такими як маршрутизатори, сервери, мережеві комутатори, точки доступу, системи відеоспостереження тощо. У її базі даних зберігається інформація про топологію мережі, адреси пристроїв, параметри їхнього навантаження, поточний стан, історія подій і навіть дані користувачів. Це створює необхідність формалізованого підходу до питань захисту: як технічного, так і організаційного.

З метою зменшення ризиків несанкціонованого доступу або неумисного втручання в роботу системи було реалізовано кілька послідовних заходів. Усі вони орієнтовані на практичні аспекти впровадження безпеки та відповідають загальновизнаним рекомендаціям із захисту IT-систем у публічних і корпоративних мережах.

Для забезпечення захищеного входу в систему передбачена авторизація

через логін і пароль. У разі введення неправильних даних доступ блокується. Також налаштовано автоматичне завершення сесії після визначеного періоду бездіяльності, що дозволяє уникнути залишення активного сеансу на публічному або незахищеному комп’ютері.

Серверний компонент системи підтримує роботу через протокол HTTPS, який забезпечує шифрування всього трафіку між браузером користувача та сервером. Це унеможливлює перехоплення переданих даних, включаючи облікові записи, через мережу. У разі застосування зовнішнього доменного імені, можна легко налаштувати SSL-сертифікати (наприклад, від Let’s Encrypt).

Ще одним важливим елементом захисту є обробка облікових даних користувачів. У системі паролі не зберігаються у відкритому вигляді — вони проходять процес хешування, тобто перетворення у спеціальну форму, що не дозволяє відновити початковий пароль. При вході користувача обчислюється хеш і порівнюється з тим, що зберігається у базі даних.

Всі дані зберігаються у базі PostgreSQL, яка розгортається в ізольованому середовищі за допомогою контейнерної технології Docker. Доступ до цієї бази можливий лише зсередини docker-мережі, що виключає підключення ззовні безпосередньо. Крім того, встановлюються унікальні паролі доступу для сервісів, і вони не зберігаються у вихідних файлах, які потрапляють у загальнодоступні сховища.

Для забезпечення контролю за діями в системі реалізовано ведення логів – журналу подій, які стосуються ключових операцій, таких як зміна конфігурації, створення або видалення об’єктів, вхід у систему, спрацювання тригерів. Це дозволяє не лише проводити аудит, а й швидко ідентифікувати джерело проблеми або підозрілу активність.

Усі сервіси системи розгортаються через контейнеризацію, що дозволяє легко та безпечно оновлювати компоненти без втручання у базову операційну систему. Регулярне оновлення версій серверного ПЗ, бібліотек та компонентів системи є необхідною умовою забезпечення актуального захисту від відомих вразливостей.

Система передбачає роботу на серверному хості, тому окрім програмного захисту необхідно враховувати й фізичну безпеку. Сервер має розміщуватись у приміщенні з обмеженим доступом, під охороною або контролем доступу. Можливість фізичного втручання в систему повинна бути мінімізована або виключена, з використанням засобів резервного живлення, блокування зовнішніх портів тощо.

Інформаційна безпека не є одноразовим етапом чи налаштуванням, а становить безперервний процес, який супроводжує систему протягом усього життєвого циклу. У ході реалізації цієї системи було враховано не лише базові технічні механізми захисту, але й організаційні принципи, що дозволяють знизити ризики як з боку зовнішнього, так і внутрішнього втручання. Завдяки багаторівневому підходу до безпеки (аутентифікація, шифрування, ізоляція, контроль доступу, аудит) система є надійною платформою для моніторингу та керування мережевими ресурсами.

Фізичні та організаційні заходи безпеки під час роботи із серверним обладнанням.

У разі експлуатації системи моніторингу на реальному фізичному серверному обладнанні виникає необхідність не лише в захисті інформації на програмному рівні, а й в організації безпечної роботи з апаратною частиною. Це особливо актуально у випадках, коли сервери розміщено в спеціалізованих серверних приміщеннях, шафах, кластерах або віртуалізованих середовищах, що потребують фізичного, процедурного й адміністративного контролю.

Загалом, робота із серверними ресурсами вимагає дотримання ряду базових заходів, спрямованих на забезпечення:

- фізичної цілісності обладнання;
- захисту від несанкціонованого доступу;
- попередження аварійних ситуацій;
- дотримання правил експлуатації, вказаних у документації виробника.

Сервери, на яких розміщується система моніторингу, повинні розміщуватись у спеціально обладнаному приміщенні (наприклад, локальній

серверній кімнаті). У такому приміщенні обов’язковою вимогою є обмеження фізичного доступу – шляхом використання замків, систем ідентифікації (магнітні картки, ключі, біометрія), а також обов’язкове ведення журналу відвідувань. Доступ мають отримувати виключно уповноважені особи.

Серверне обладнання, особливо при цілодобовому навантаженні, потребує стабільного кліматичного середовища. Рекомендується обладнати серверну систему моніторингу температури, вентиляції або кондиціонування. Окрім того, для запобігання пошкодженню обладнання внаслідок стрибків напруги або вимкнення електроенергії, рекомендовано використовувати джерела безперебійного живлення (UPS) з можливістю автоматичного вимкнення системи у разі тривалої відсутності живлення.

Усі фізичні пристрої, що беруть участь у реалізації системи, мають бути змонтовані у відповідні серверні шафи. Це не лише впорядковує кабельне середовище, а й дозволяє організувати належне охолодження, захистити порти від випадкового натискання або відключення, мінімізує механічні пошкодження та спрощує обслуговування.

Усі сервери, на яких розміщується система, повинні бути включені до внутрішнього обліку IT-активів. Доцільним є маркування кожного пристрою із зазначенням інвентарного номеру, дати введення в експлуатацію, контактної особи відповідальної за обслуговування та технічних характеристик. Це дозволяє швидко ідентифікувати пристрій у разі виникнення технічної несправності або під час проведення планових перевірок.

Фізичні сервери повинні регулярно перевірятися на предмет наявності пилу, коректної роботи вентиляторів, стану кабелів, з’єднань та світлодіодних індикаторів. У деяких випадках доцільно реалізувати внутрішню систему апаратного моніторингу, яка дозволяє відстежувати температуру, оберти вентиляторів, стан живлення тощо.

Відповідно до політики безпеки організації, доступ до серверного обладнання має бути чітко регламентований. У разі обслуговування сторонніми підрядниками повинно здійснюватися супроводження відповідальною особою.

Усі роботи мають фіксуватись у відповідному журналі або за допомогою системи спостереження.

Оскільки фізичне обладнання схильне до зносу або аварій, обов’язковою вимогою є впровадження системи резервного копіювання (наприклад, щоденне створення бекапів бази даних та конфігурацій). Ці резервні копії мають зберігатись у захищеному сховищі, окремо від основного сервера, з можливістю швидкого відновлення.

Отже, безпека системи не обмежується лише програмним захистом, а охоплює також фізичну інфраструктуру, процедури доступу, інвентаризації, кліматичного та енергетичного контролю. Забезпечення комплексної безпеки при роботі із серверним обладнанням є запорукою стабільності, надійності та довготривалої працездатності не лише розробленої системи, а й усієї IT-інфраструктури організації загалом.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи на тему «Система моніторингу та керування мережевими ресурсами» було проведено комплекс досліджень, розробок і впроваджень, спрямованих на створення програмного рішення, здатного здійснювати автоматизоване спостереження за станом мережевих пристроїв, збереження їх технічних характеристик, виявлення подій, формування тригерів і надання актуальної інформації користувачу в зручному форматі.

Основною метою, що ставилася під час розробки, було створення універсального інструменту для централізованого спостереження за станом мережевих ресурсів в умовах реального середовища. Розроблена система успішно реалізовує зазначене завдання: забезпечує безперервний контроль за роботою пристроїв, здійснює збір і обробку інформації, фіксує події та генерує відповідні повідомлення про зміни у статусі обладнання. Це дозволяє не лише підвищити рівень контролю над інфраструктурою, але й своєчасно виявляти критичні ситуації, сприяючи стабільній роботі IT-систем користувача.

Серед особливостей реалізованого рішення варто відзначити чітку модульну структуру програмного забезпечення, що дає змогу адаптувати систему до потреб різних організацій. Архітектура рішення дозволяє розгортати його як у невеликих мережах, так і в масштабованих середовищах з великою кількістю вузлів. Завдяки використанню сучасного серверного середовища з контейнеризацією було досягнуто високої гнучкості, переносимості та спрощення процесів інсталяції та підтримки.

Важливою перевагою є незалежність системи від конкретного апаратного забезпечення. Рішення може бути впроваджене як на фізичних серверах, так і на віртуальних машинах із типовими параметрами продуктивності, без потреби у спеціалізованому обладнанні. Водночас забезпечено можливість масштабування як по обсягу оброблюваних даних, так і за кількістю пристроїв, що моніторяться, без суттєвого зниження продуктивності.

Проведене тестування підтвердило функціональну відповідність

розробленого ППЗ поставленим вимогам. Було перевірено основні сценарії роботи системи – від створення нових пристроїв і налаштування моніторингових параметрів до відображення спрацювань тригерів та взаємодії через користувацький інтерфейс. Система стабільно функціонує у визначених умовах, демонструє коректну обробку подій і дозволяє користувачеві здійснювати візуальний контроль за станом мережі в реальному часі.

З точки зору техніко-економічної ефективності, створене програмне забезпечення є доцільним до впровадження в умовах середніх і малих підприємств, навчальних закладів або організацій, що не потребують дороговартісних комерційних рішень, але мають реальну потребу у відстеженні стану своєї мережевої інфраструктури. Використання безкоштовних, легальних та відкрито підтримуваних технологій значно знижує витрати на впровадження, технічну підтримку та модернізацію системи.

Оцінка отриманих результатів свідчить про повне виконання поставленого завдання. Усі характеристики проєктованої системи відповідають сформульованим у технічному завданні вимогам, а також демонструють конкурентну здатність розробки в порівнянні з існуючими аналогами. У процесі реалізації були застосовані сучасні методи проєктування логічних структур баз даних, розроблено інтерфейс користувача з урахуванням принципів зручності та доступності, а також реалізовано алгоритми, що забезпечують надійний контроль мережевої активності.

Можливі напрямки подальшого розвитку включають: розширення типів пристроїв, що моніторяться; реалізацію мобільної версії інтерфейсу; інтеграцію із зовнішніми службами сповіщення; побудову гнучкої системи аналітики та звітів на основі історичних даних. Запропоноване рішення може стати основою для побудови комплексного середовища управління IT-інфраструктурою з використанням методів автоматизації та самовідновлення.

Додатково слід зазначити, що розроблена система є прикладом сучасного підходу до створення інформаційних рішень, який поєднує в собі практичність, функціональність та адаптивність. Реалізація системи у вигляді портативного

програмного продукту дозволяє впроваджувати її в різноманітних середовищах – як у локальних офісах із обмеженими ресурсами, так і в розгалужених мережах корпоративного масштабу. Універсальність обраної архітектури сприяє швидкій адаптації рішення під різні конфігурації, без потреби в ґрунтовній технічній підготовці персоналу чи тривалому навчанні користувачів.

Таким чином, розроблена система демонструє повну готовність до практичного використання, відповідає сучасним вимогам до систем даного класу, та має потенціал для подальшого вдосконалення в умовах реального впровадження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/

2. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/

3. Nginx: Official documentation [Електронний ресурс]. – Режим доступу: https://nginx.org/en/docs/

4. Prometheus Monitoring System [Електронний ресурс]. – Режим доступу: https://prometheus.io/

5. Nagios Core Monitoring [Електронний ресурс]. – Режим доступу: https://www.nagios.org/projects/nagios-core/

6. PRTG Network Monitor [Електронний ресурс]. – Режим доступу: https://www.paessler.com/prtg

7. GNU/Linux Ubuntu Server Documentation [Електронний ресурс]. – Режим доступу: https://ubuntu.com/server/docs

8. SNMPv2: Simple Network Management Protocol. RFC 3416. Internet Engineering Task Force (IETF), 2002. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc3416

9. Grafana Labs Documentation: Data Visualization for Monitoring Systems [Електронний ресурс]. – Режим доступу: https://grafana.com/docs/

ДОДАТОК А

Код програми

Сторінок 5

КИЇВ – 2025

А. 1 Виконання SNMP-опитування мережевого пристрою за вказаним OID

```
#include <net-snmp/net-snmp-config.h>
#include <net-snmp/net-snmp-includes.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
if (argc != 4) {
fprintf(stderr, "Usage: %s <ip_address> <oid> <community>\n", argv[0]);
return 1;
}
const char *ip = argv[1];
const char *oid_str = argv[2];
const char *community = argv[3];

struct snmp_session session, *ss;
struct snmp_pdu *pdu;
struct snmp_pdu *response;
struct variable_list *vars;
oid anOID[MAX_OID_LEN];
size_t anOID_len = MAX_OID_LEN;

int status;

// Ініціалізація SNMP
init_snmp("snmp_get_value");
snmp_sess_init(&session);
session.peername = strdup(ip);
session.version = SNMP_VERSION_2c;
```

```
session.community = (u_char *)community;
session.community_len = strlen(community);

ss = snmp_open(&session);
if (!ss) {
    snmp_perror("snmp_open");
    return 1;
}

if (!snmp_parse_oid(oid_str, anOID, &anOID_len)) {
    snmp_perror("snmp_parse_oid");
    snmp_close(ss);
    return 1;
}

pdu = snmp_pdu_create(SNMP_MSG_GET);
snmp_add_null_var(pdu, anOID, anOID_len);

status = snmp_synch_response(ss, pdu, &response);
if (status == STAT_SUCCESS && response->errstat == SNMP_ERR_NOERROR)
{
    vars = response->variables;
    char value[1024] = {0};
    snprint_value(value, sizeof(value), vars->name, vars->name_length, vars);
    printf("%s\n", value);  // Тільки значення (наприклад: "Dell-Router")
} else {
    fprintf(stderr, "SNMP Error\n");
    if (status == STAT_SUCCESS)
        fprintf(stderr, "Error: %s\n", snmp_errstring(response->errstat));
```

```
    else
        snmp_sess_perror("snmp_get_value", ss);
    snmp_close(ss);
    if (response) snmp_free_pdu(response);
    return 1;
}

if (response) snmp_free_pdu(response);
snmp_close(ss);
return 0;
}
```

А. 2 Скрипт для масового опитування пристроїв за SNMP і фіксації результатів у лог-файлі.

```
#!/bin/bash
INPUT="hosts.txt"
COMMUNITY="public"
OID="1.3.6.1.2.1.1.5.0"
LOG="snmp_results.log"

echo "SNMP опитування розпочато: $(date)" > $LOG

while IFS= read -r ip; do
VALUE=$(./snmp_get_value "$ip" "$OID" "$COMMUNITY")
echo "$ip | $VALUE" >> $LOG
done < "$INPUT"

echo "Готово: $(date)" >> $LOG
```

А. 3 Фрагмент коду відображення останніх даних

```
foreach ($data['items'] as $itemid => $item) {
	$host = $data['hosts'][$item['hostid']];

	$data_actions = [];
	$is_graph = ($item['value_type'] == ITEM_VALUE_TYPE_FLOAT ||
$item['value_type'] == ITEM_VALUE_TYPE_UINT64);
	if ($is_graph) {
		$data_actions['graph'] = true;
	}

	if (in_array($item['type'], checkNowAllowedTypes())
			&& $item['status'] == ITEM_STATUS_ACTIVE &&
$host['status'] == HOST_STATUS_MONITORED
			&& array_key_exists($itemid, $data['items_rw'])) {
		$data_actions['execute'] = true;
	}

	$checkbox = new CCheckBox('itemids['.$itemid.']', $itemid);
	if ($data_actions) {
		$checkbox->setAttribute('data-actions', implode(' ',
array_keys($data_actions)));
	}
	$item_name = (new CDiv([
		(new CLinkAction($item['name']))
			->setMenuPopup(
				CMenuPopupHelper::getItem([
					'itemid' => $itemid,
```

```
                                'context' => 'host',
                                'backurl' => (new CUrl('syshelper.php'))
                                        ->setArgument('action', 'latest.view')
                                        ->setArgument('context','host')
                                        ->getUrl()])),
                ($item['description_expanded'] !== '') ?
makeDescriptionIcon($item['description_expanded']) : null ]))
$last_history = array_key_exists($itemid, $data['history'])
                ? ((count($data['history'][$itemid]) > 0) ? $data['history'][$itemid][0] :
null)
                : null;
        if ($last_history) {
                $prev_history = (count($data['history'][$itemid]) > 1) ?
$data['history'][$itemid][1] : null;
                $last_check = (new CSpan(date2age($last_history['clock'])))
                        ->setHint(date2str(DATE_TIME_FORMAT_SECONDS,
$last_history['clock']), '', true, '', 0);
                if ($item['value_type'] == ITEM_VALUE_TYPE_BINARY) {
                        $last_value = italic(_('binary value'))}
                else {
                        $last_value = (new
CSpan(formatHistoryValue($last_history['value'], $item, false)))
                                ->setHint(
                                        (new CDiv(mb_substr($last_history['value'], 0,
HINTBOX_CONTENT_LIMIT)))
                        ->addClass(HINTBOX_RAW_DATA)
                        ->addClass(HINTBOX_WRAP),
                                        '', true, '', 0);}
                $change = '';
```