ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОНІТОРИНГУ ІТ-ІНФРАСТРУКТУРИ ТА КОНТРОЛЮ ПРОДУКТИВНОСТІ ПЕРСОНАЛУ З РОЛЕВИМ ДОСТУПОМ»

на здобуття освітнього ступеня магістр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

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

_____ Богдан ОЛЕКСІЄНКО
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСДМ-61 Богдан ОЛЕКСІЄНКО
(ім'я, ПРІЗВИЩЕ)
Керівник Ольга ПОЛОНЕВИЧ
к.т.н. *(ім'я, ПРІЗВИЩЕ)*
Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

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

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

"____" _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Олексієнку Богдану Олеговичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Інтелектуальна система моніторингу ІТ-інфраструктури та контролю продуктивності персоналу з ролевим доступом
керівник кваліфікаційної роботи: Ольга ПОЛОНЕВИЧ к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від " 30 " жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Системи моніторингу ІТ-інфраструктури та їх архітектурні особливості.
2. Методи збору, обробки та аналізу телеметричних і подієвих даних.
3. Моделі контролю продуктивності персоналу в корпоративних інформаційних системах.
4. Засоби організації ролевого доступу та забезпечення інформаційної безпеки.
5. Науково-технічна та нормативна література з питань моніторингу, аналітики та кібербезпеки.

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

1. Дослідження еволюції підходів та сучасних тенденцій розвитку систем моніторингу IT-інфраструктури.
2. Огляд архітектурних моделей систем моніторингу та засобів організації ролевого доступу.
3. Аналіз методів збору, нормалізації та зберігання телеметричних і поведінкових даних.
4. Дослідження алгоритмічних підходів до аналізу подій та виявлення аномалій.
5. Обґрунтування доцільності інтеграції моніторингу IT-інфраструктури та контролю продуктивності персоналу.
6. Аналіз результатів практичної реалізації інтелектуальної системи моніторингу та контролю продуктивності.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «30» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір та аналіз літератури	30.10.2025 – 02.11.2025	
2.	Дослідження сучасних підходів до моніторингу IT-інфраструктури	03.11.2025 – 07.11.2025	
3.	Дослідження методів аналізу подій та виявлення аномалій	14.11.2025 – 17.11.2025	
4.	Розробка та реалізація інтелектуальної системи моніторингу	18.11.2025 – 24.11.2025	
5.	Висновки по роботі	25.11.2025	
6.	Розробка демонстраційних матеріалів, доповідь.	27.11.2025	
7.	Оформлення магістерської роботи	28.11.2025	

Здобувач вищої освіти _____ Богдан ОЛЕКСІЄНКО
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ольга ПОЛОНЕВИЧ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступеня магістра: 84 стор., 8 рис., 2 табл., 30 джерел.

Мета роботи – створення підходу до побудови інтелектуальної системи моніторингу, здатної збирати різнорідні події в реальному часі, структурувати їх, визначати динаміку роботи мережевих вузлів і перетворювати технічні зміни на управлінські показники.

Об'єкт дослідження – процес моніторингу та аналізу станів IT-інфраструктури у поєднанні з відстеженням активності персоналу.

Предмет дослідження – методи інтеграції процесів моніторингу IT-інфраструктури та контролю продуктивності персоналу в єдину систему з ролевим доступом.

Короткий зміст роботи. У першому розділі магістерської роботи виконано аналіз еволюції підходів до моніторингу IT-інфраструктури та сучасних тенденцій розвитку відповідних систем. Розглянуто архітектурні моделі систем моніторингу, методи збору та обробки телеметричних даних, а також принципи організації ролевого доступу і забезпечення інформаційної безпеки. У другому розділі проведено огляд існуючих інструментів моніторингу IT-інфраструктури та проаналізовано їх функціональні обмеження. Досліджено бізнес-вимоги до контролю продуктивності персоналу, виконано аналіз загроз інформаційної безпеки та обґрунтовано доцільність побудови інтегрованої інтелектуальної системи. У третьому розділі описано практичну реалізацію інтелектуальної системи моніторингу та контролю продуктивності персоналу. Представлено архітектуру програмного комплексу, реалізацію серверної частини, модулі моніторингу IT-інфраструктури та аналізу продуктивності персоналу, а також результати демонстрації основних функціональних можливостей системи.

КЛЮЧОВІ СЛОВА: IT-ІНФРАСТРУКТУРА, СИСТЕМА МОНІТОРИНГУ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА, КОНТРОЛЬ ПРОДУКТИВНОСТІ, РОЛЕВИЙ ДОСТУП, АНАЛІЗ ПОДІЙ, АНОМАЛІЇ, ІНФОРМАЦІЙНА БЕЗПЕКА, ТЕЛЕМЕТРІЯ.

ABSTRACT

The textual part of the master's qualification work comprises 84 pages, 2 tables, 8 fig., 30 references.

The purpose of the work is to develop an approach to building an intelligent monitoring system capable of collecting heterogeneous events in real time, structuring them, determining the operational dynamics of network nodes, and transforming technical changes into managerial performance indicators.

The object of the research is the process of monitoring and analyzing the states of IT infrastructure combined with tracking personnel activity.

The subject of the research is methods for integrating IT infrastructure monitoring and personnel productivity control into a unified role-based access system.

Brief content of the work. The first chapter analyzes the evolution of approaches to IT infrastructure monitoring and current development trends. Architectural models of monitoring systems, methods of collecting and processing telemetry data, as well as principles of role-based access control and information security are considered.

The second chapter reviews existing IT infrastructure monitoring tools and analyzes their functional limitations. Business requirements for personnel productivity control are examined, information security risks are analyzed, and the feasibility of developing an integrated intelligent system is substantiated.

The third chapter describes the practical implementation of an intelligent monitoring and personnel productivity control system. The architecture of the software complex, server-side implementation, IT infrastructure monitoring module, personnel productivity analysis module, and the results of demonstrating the system's key functional capabilities are presented.

KEYWORDS: IT INFRASTRUCTURE, MONITORING SYSTEM, INTELLIGENT SYSTEM, PERSONNEL PRODUCTIVITY CONTROL, ROLE-BASED ACCESS, EVENT ANALYSIS, ANOMALIES, INFORMATION SECURITY, TELEMETRY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACL — Access Control List (список контролю доступу)

API — Application Programming Interface (інтерфейс програмування застосунків)

APM — Application Performance Monitoring (моніторинг продуктивності застосунків)

CPU — Central Processing Unit (центральний процесор)

CSV — Comma-Separated Values (текстовий формат табличних даних, розділених комами)

DevOps — Development and Operations (підхід до спільної роботи розробки та експлуатації)

EFK — Elasticsearch + Fluentd + Kibana (стек збору та аналізу логів)

ELK — Elasticsearch + Logstash + Kibana (стек збору та аналізу логів)

ICMP — Internet Control Message Protocol (службовий мережевий протокол)

I/O — Input / Output (операції введення та виведення даних)

IP — Internet Protocol (мережевий протокол та IP-адреса)

JSON — JavaScript Object Notation (текстовий формат обміну структурованими даними)

RBAC — Role-Based Access Control (рольова модель керування доступом)

SAN — Storage Area Network (мережа зберігання даних)

SLO — Service Level Objective (цільовий показник якості сервісу)

SRE — Site Reliability Engineering (підхід до забезпечення надійності систем)

SNMP — Simple Network Management Protocol (протокол керування мережевими пристроями)

WMI — Windows Management Instrumentation (інтерфейс керування компонентами ОС Windows)

ВСТУП

Потреба у точному й водночас гнучкому контролі IT-інфраструктури вже давно вийшла за межі технічної підтримки. У сучасних організаціях вона перетворилася на окремий пласт управління, де від стану мережі, стабільності робочих місць і дисципліни персоналу залежить не просто швидкість обробки даних, а загальна стійкість бізнес-процесів. Здається, що інструментів моніторингу зараз безліч, але в реальній роботі швидко виявляється, що більшість із них вміє добре робити лише одну частину задачі: одні фіксують технічні події, інші намагаються оцінювати поведінку користувачів, але між цими вимірами майже немає узгодженості. Через це значна частина інформації просто «розчиняється» між окремими системами, і керівнику доводиться вручну зводити стан інфраструктури з активністю персоналу, що природно створює прогалини. На цьому тлі тема побудови інтегрованої інтелектуальної системи, яка одночасно відслідковує стан мережі й аналізує продуктивність співробітників, стала надзвичайно актуальною.

У певний момент стає очевидно, що традиційні засоби не справляються з інтенсивністю змін. Робочі місця зникають із мережі, потім знову з’являються; одні співробітники працюють стабільно, інші – фрагментарно; при цьому обсяг телеметрії постійно зростає, і без автоматизації ця інформація не перетворюється на практичну користь. Саме з цієї логіки виникла ідея розробити систему, здатну не просто реагувати на технічні події, а й формувати на їх основі осмислену оцінку діяльності персоналу. Так сформувався концепт інтелектуальної системи моніторингу IT-інфраструктури та контролю продуктивності з ролевим доступом, де обидві складові працюють як частини однієї аналітичної моделі.

Мета роботи – створення підходу до побудови інтелектуальної системи моніторингу, здатної збирати різнорідні події в реальному часі, структурувати їх, визначати динаміку роботи мережевих вузлів і перетворювати технічні зміни на управлінські показники.

Об’єкт дослідження – процес моніторингу та аналізу станів IT-інфраструктури у поєднанні з відстеженням активності персоналу.

Предмет дослідження – методи інтеграції процесів моніторингу IT-інфраструктури та контролю продуктивності персоналу в єдину систему з ролевим доступом.

Методи дослідження поєднували огляд теоретичних підходів, аналіз архітектурних моделей моніторингу, роботи з телеметричними даними, перевірку алгоритмів виявлення змін у поведінці вузлів, моделювання потоків подій і розробку прикладного програмного забезпечення. Поступово стало зрозуміло, що традиційні способи обчислення показників продуктивності не можуть працювати без прямої прив'язки до технічних подій, тому в системі використовувався подієво-орієнтований підхід, де кожна зміна стану створює часовий інтервал, який у подальшому стає елементом аналізу.

Наукова новизна полягає у формуванні моделі, що дозволяє об'єднати моніторинг інфраструктури і контроль персоналу у спільне аналітичне ядро. Запропонована система використовує мінімалістичні, але точні механізми подієвої фіксації, що забезпечують коректне визначення тривалості онлайн та офлайн періодів, не перевантажуючи інфраструктуру надмірною кількістю даних. Такий підхід створює основу для інтелектуальних надбудов, яким не потрібні складні інструменти машинного навчання, щоб отримувати достовірні управлінські метрики.

Практична значущість результатів полягає в тому, що розроблена система може застосовуватися у реальних організаційних умовах: вона забезпечує моніторинг мережі, сигналізує про інциденти, зберігає їх хронологію та автоматично формує аналітичну звітність для керівництва. Ролевий доступ дозволяє гнучко розмежувати функції між адміністраторами та управлінцями. Завдяки модульності система може масштабуватися й адаптуватися під інші середовища без повної перебудови архітектури.

Апробація результатів магістерської роботи. Основні положення і результати магістерської роботи публікувались на науково практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМ МОНІТОРИНГУ ІТ-ІНФРАСТРУКТУРИ

Моніторинг ІТ-інфраструктури рідко починають сприймати серйозно з першого дня роботи з системами. Спершу здається, ніби все досить передбачувано: сервери крутяться, мережа тримає з'єднання, сервіс відповідає якось рівно, і взагалі нічого особливого. Але минає трохи часу, і вже відчувається певне «підспудне» напруження, бо якась дрібниця починає вибиватися: то запит іде на долю секунди довше, то вночі логи ніби поводяться дивніше, ніж мали б. А іноді взагалі виникає відчуття, що вся інфраструктура живе своїм життям, хоча ти її, наче, контролюєш. Це незручно, бо і наче нічого не ламається прямо зараз, і водночас зрозуміло, що відсутність явної проблеми-це не гарантія, що все гаразд.

І з досвідом приходить дивне розуміння: дрібні відхилення не просто сигнали. Вони як сліди від чогось більшого, що легко загубити, якщо дивитися лише на окремі значення. Тому стає потрібно збирати не тільки метрики, а все підряд-те, що колись вважалося другорядним. Логи, не зовсім зрозумілі події, короткі сплески навантаження. Іноді навіть дії користувачів, які взагалі ніхто не відносив до категорії критичних. Усе це поступово починає сплітатися у щось більш складне, ніж просто «моніторинг».

З часом стає помітно, що вже недостатньо знати, що сервер живий. Бажано розуміти, як він «дихає». Чи нормально реагує на пікові періоди, чому два однакові запити поводяться по-різному, чому вчора певний модуль працював впевнено, а сьогодні вже не так. Світ, здається, не став гіршим, просто системи стали більш розгалуженими, і їх поведінка іноді нагадує живий організм, у якому багато процесів неможливо розкласти на прямі причинно-наслідкові зв’язки. Або можна, але тільки після тривалого розбору.

До цього додається інший аспект, який раніше часто ігнорували: люди. Дивно, але багато аномалій прямо пов’язані не зі збоями техніки, а з тим, як персонал виконує свої дії. Хтось запускає важкий процес не в той час, хтось щось оновлює вручну, хтось робить десятки зайвих запитів через звичку або поспіх.

Воно ніби незначне, але технологічно дуже відчутне. І коли це помічаєш, починаєш розуміти, чому на окремому рівні моніторинг виглядає неповним: частина картини лишається за межами технічного аналізу.

Паралельно розмивається межа між моніторингом та кібербезпекою. Якщо подія виглядає підозрілою, часто складно відразу сказати, це помилка системи чи нетипова активність користувача, чи взагалі спроба втручання. Поява хмар, контейнерів, мікросервісів додала хаотичності: кожен компонент живе окремо, змінюється окремо і «ламається» теж по-своєму. І виглядає так, ніби немає більше однієї стабільної точки контролю. Через це моніторинг перетворюється не на список перевірок, а на спосіб розуміння того, що взагалі відбувається з інфраструктурою.

Поступово складається враження, що моніторинг сьогодні це не просто технічний нагляд, а певна комбінована практика, яка стосується і технічних станів, і звичок людей, і логіки взаємодії компонентів між собою. І стає доволі логічно розбиратися, як саме змінилися підходи, чому, наприклад, чистих метрик уже недостатньо, які архітектури безпечніше й простіше підтримувати, і як поєднати контроль системи з аналізом поведінки персоналу так, щоб він не став інструментом тиску, а допомагав уникати проблем.

Усе це створює базу для розгляду теоретичних аспектів моніторингу, але не в сенсі сухої теорії, а радше як спроби пояснити той хаос, з яким стикається інфраструктура, коли вона зростає і ускладнюється. Тому питання розвитку підходів, архітектур, методів збору і аналізу даних поступово вибудовуються в цілісну картину, без якої вже важко зрозуміти логіку сучасних систем моніторингу.

1.1 Еволюція підходів до моніторингу IT-інфраструктури та сучасні тренди

Перші підходи до моніторингу корпоративних IT-інфраструктур мали переважно ручний характер. Контроль стану систем здійснювався без

централізованих інструментів і формалізованих процедур, ґрунтуючись на періодичних перевірках, аналізі журналів та інтуїтивному досвіді адміністратора. Такий підхід був прийнятним лише за умов невеликої кількості серверів і низької динаміки змін, але не забезпечував масштабованості та передбачуваності.

Подальший розвиток мереж призвів до появи стандартних протоколів збору стану обладнання, зокрема SNMP, що дозволило централізовано отримувати базові метрики без прямого доступу до кожного вузла. На цій основі сформувалися перші системи моніторингу, які автоматизували опитування пристроїв, порівняння показників із пороговими значеннями та генерацію сповіщень у разі відхилень.

Наступним етапом стало становлення класичних платформ моніторингу, таких як Nagios, Zabbix, Cacti та подібні рішення. Вони запровадили структуру конфігурацій, шаблони перевірок, ієрархію об’єктів та базові механізми пріоритизації подій. Водночас концептуально моніторинг залишався орієнтованим на контроль окремих метрик і відповідь на питання, чи перебувають показники в допустимих межах.

Обмеження такого підходу проявилися з часом. Порогові механізми погано виявляли поступові деградації продуктивності, не враховували взаємозв’язки між компонентами та створювали значний обсяг малозначущих сповіщень. Крім того, класичний моніторинг був зосереджений переважно на фізичній інфраструктурі, залишаючи прикладний рівень і реальний користувацький досвід поза фокусом уваги.

Ситуація суттєво ускладнилася з поширенням віртуалізації, хмарних платформ і мікросервісної архітектури. Інфраструктура стала динамічною, ресурси — тимчасовими, а залежності між компонентами — складними та неочевидними.

У таких умовах класичні метрики окремих вузлів перестали відображати реальний стан сервісів, а проблеми часто виникали без явного перевищення порогових значень.

Додатковим чинником стала зміна вимог до якості сервісів. Орієнтація на показники доступності була доповнена метриками продуктивності та стабільності з точки зору користувача, що знайшло відображення в концепціях SLO та SLA. Це змістило фокус моніторингу з ресурсів на поведінку системи в реальних сценаріях використання.

У відповідь на ці виклики почала формуватися концепція спостережуваності (observability), яка передбачає комплексний аналіз стану системи на основі поєднання метрик, логів і трасування запитів. Такий підхід дозволяє не лише фіксувати відхилення, а й аналізувати їх причини, взаємозв'язки між компонентами та динаміку розвитку інцидентів. Важливу роль у цьому відіграють уніфіковані підходи до збору телеметрії, зокрема ініціативи на кшталт OpenTelemetry.

Сучасні системи моніторингу також активно використовують елементи автоматизованого аналізу та алгоритми виявлення аномалій, що дає змогу зменшити шум сповіщень і виявляти проблеми на ранніх етапах. У результаті моніторинг еволюціонував від простого контролю параметрів до інструмента глибокого розуміння поведінки складних IT-систем та підтримки їхньої стабільності в умовах постійних змін.

1.2. Архітектурні моделі побудови систем моніторингу (агентні, безагентні, гібридні)

Агентні системи моніторингу

Коли говорять про архітектуру системи моніторингу, мають на увазі не тільки те, який стоїть сервер з дашбордами, а й те, як саме дані народжуються на контрольованих вузлах і як вони потрапляють у "центр". Десь логіка спостереження знаходиться майже повністю в центральному компоненті, а десь навпаки-значна частина роботи виконується прямо на машинах, за якими стежать. Оця різниця в тому, де "живе" інтелект моніторингу, і формує основні архітектурні моделі.

Агентний підхід у цьому сенсі виглядає доволі очевидно: на кожен

потрібний сервер, робочу станцію або інший вузол ставиться невелика програма, яка живе там постійно. Вона збирає показники, слухає події, іноді навіть аналізує їх локально, а потім відправляє далі те, що має піти в центральну систему. На перший погляд це звучить як черговий “маленький сервіс”, яких і так повно, але на практиці саме від цього агента залежить, наскільки глибоко компанія реально “бачить” те, що відбувається в її інфраструктурі.

Як це виглядає в живому середовищі? На сервері з’являється ще один процес, який більшість користувачів взагалі не помічають. Він тихо читає системні лічильники, викликає внутрішні API операційної системи, дивиться, що робиться з дисками, мережевими інтерфейсами, процесами. Подекуди він ще й підчіплюється до застосунків, щоб зчитувати додаткову телеметрію: наприклад, час виконання запитів до бази чи внутрішні метрики сервісу. З боку це може виглядати як невеликий демон, але обсяги інформації, які він здатен зібрати, вражають.

Сильна сторона агентного підходу якраз у тому, що він знаходиться максимально близько до реальних подій. Не потрібно гадати, що там насправді робиться на хості: програма сидить всередині й бачить усе власними очима. Можна збирати не тільки загальні показники, а й досить низькорівневі речі: системні виклики, параметри вводу-виводу, помилки ядра, події безпеки. Якщо потрібно, агент може навіть кешувати дані локально, коли зв'язок із сервером моніторингу обривається, а потім “дозалити” їх, коли канал відновиться. Це те, чого часто не вистачає безагентним моделям.

Інша цікава властивість агента-він може частину логіки аналізу виконувати прямо на вузлі. Не завжди є сенс ганяти всі сирі дані в центр. Іноді вигідно, щоб саме агент вирішував, що варте відправлення, а що можна відфільтрувати на місці. Наприклад, якщо процес стабільно працює в межах нормальних значень, то немає потреби кожну секунду повідомляти про це центральний сервер. Навпаки, якщо він помічає локальну аномалію, то може підсилити частоту відправки або навіть сформувати попередній висновок. Виходить така собі “мала аналітика на краю”.

Звісно, у цього підходу є й своя тіньова сторона. Щоб усе це запрацювало,

потрібно цей агент встановити, оновити, підтримувати в актуальному стані на кожному вузлі. У невеликій інфраструктурі це ще можна робити вручну, але в середовищі зі сотнями чи тисячами серверів така історія швидко перетворюється на головний біль. І доводиться будувати додаткову логіку розгортання-через системи конфігурацій, групові політики, власні скрипти. Не всі організації до цього готові, особливо якщо інфраструктура історично хаотична.

Ще одна проблема, яка часто спливає на практиці,-вплив агента на продуктивність. Він, звісно, не має права “з’їдати” помітну частину ресурсів, але повністю непомітним він теж не буває. Кожного разу, коли йому потрібно зібрати метрики, він робить певні виклики, читає файли, опитує системні структури. Якщо конфігурацію переборщити, агент може створювати додаткове навантаження, яке в критичних сценаріях буде дуже відчутним. Тому параметри його роботи треба дуже обережно налаштовувати, і це знову ж таки додає задач адміністраторам.

Плюс, з точки зору безпеки, кожен агент-це ще один шматок коду в системі. Йому довіряють досить багато: він має доступ до системної інформації, іноді працює з підвищеними правами, може підключатися до внутрішніх API. Якщо в ньому є вразливості або якщо його конфігурація скомпрометована, це створює додатковий канал ризику. Через це безпекові команди часто ставляться до агентів з підозрою, особливо в середовищах із жорсткими політиками. Вони хочуть знати, що саме агент робить, які порти відкриває, які дані передає, як шифрує трафік і хто має доступ до його конфігурації.

Разом з тим є такі задачі, де без агента просто ніяк. Наприклад, коли потрібно моніторити роботу кінцевих пристроїв користувачів, а не тільки серверів. Робочі станції, ноутбуки, навіть мобільні пристрої-усе це не завжди видно з боку мережі. Там потрібен хтось, хто сидітиме “на місці” й фіксуватиме, як поводяться локальні процеси, які додатки споживають ресурси, чи не відбувається щось підозріле. Без агента в такому контексті моніторинг зводиться до здогадок за непрямими ознаками.

Окрема історія-агенти, що інтегруються безпосередньо в застосунки. Тут мова вже не стільки про операційну систему, скільки про внутрішню логіку

сервісу. Такі агенти, по суті, "вшиваються" в код або під'єднуються через спеціальні бібліотеки й перехоплюють інформацію про те, як виконуються запити, які транзакції проходять, де виникають затримки. Це дозволяє побачити картину наскрізь: від фронтенду, який отримує запит, до бази даних, яка його обробляє. І знову ж таки, без внутрішнього агента до такої глибини дістатися складно.

Однак і тут є зворотний бік-залежність від технологічного стеку. Для різних мов, фреймворків, платформ доводиться використовувати різні модулі, що збільшує фрагментацію. Якщо компанія використовує кілька стеків одночасно, підтримувати всі ці агенти в актуальному стані буває непросто. Іноді доводиться обирати: або мати глибоку видимість для частини сервісів, або широку, але поверхневу-для більшості.

Усе це приводить до простого висновку: агентні системи моніторингу дають дуже глибокий погляд на вузол і застосунок, але вимагають дисципліни в управлінні агентами, акуратних налаштувань і чітких правил безпеки. Якщо ці речі в організації кульгають, така архітектура легко перетворюється на хаос із різними версіями агента, невідомими конфігураціями й несподіваними навантаженнями на продакшн.

Попри всі недоліки, агентний підхід залишається базовим для багатьох зрілих систем моніторингу. Він дає те, чого важко досягти інакше: деталізовану картину того, що робиться всередині кожної машини. І коли мова заходить про інтелектуальні системи, які мають аналізувати не тільки поверхневі симптоми, а й глибинні патерни, наявність якісно розгорнутої мережі агентів перетворюється на одну з ключових передумов.

Безагентні моделі

Після того як агентні моделі стали буденністю у багатьох компаніях, паралельно існувала й інша логіка-спробувати зібрати інформацію так, щоб узагалі нічого не встановлювати на самі сервери. На перший погляд це звучить зручно: не треба бігати з агентами, стежити за їхніми версіями, турбуватися, що вони створять додаткові ризики чи навантаження. Система моніторингу просто "визирає" назовні й питає в сервера, що з ним відбувається. І хоча інколи це

виглядає майже магічно просто, насправді цей підхід має свої глибокі нюанси.

Безагентні інструменти опираються на можливість отримати дані дистанційно. У найпростішій формі це може бути звичайний пінг чи перевірка порту-щоб зрозуміти, чи сервер хоч відповідає. Трохи складніший рівень-опитування за певним протоколом, наприклад SNMP або WMI. У цих випадках сам сервер нічого не встановлює додатково: у нього вже є вбудовані механізми, які можуть дати певну інформацію, якщо їх правильно запитати. Це виглядає як розмови через щілину: ти не заходиш у приміщення, але можеш через двері почути, що там хтось рухається.

Тут перевага очевидна-нічого не втручається в систему. Адміністратори часто відчувають полегшення, бо не треба мати справу з програмами, які сидять у глибині операційної системи. Особливо це актуально для середовищ із суворими політиками безпеки, де ставити сторонній код на сервери заборонено або потрібно проходити довгі внутрішні перевірки. У таких умовах безагентний моніторинг іноді виявляється єдиним способом хоч якось спостерігати за сервісами.

Але тут починаються цікаві обмеження. Зовнішнє опитування дає рівно стільки, скільки сервер готовий “показати”. Якщо протокол підтримує 50 параметрів-ти отримаєш 50. Якщо 5-то 5. І з цього моменту видимість стає не результатом архітектури моніторингу, а, скоріше, наслідком того, наскільки виробник обладнання заклав потрібні можливості. Ти не можеш просто взяти та додати ще один параметр у моніторинг, як у випадку з агентами. Ти обмежений тим, що доступно “із заводу”.

Особливо жорстко це відчувається на прикладному рівні. Сервер може сказати, що в нього 30% CPU і певне навантаження на мережу. Але він не розповість, що якась бізнес-логіка уповільнюється, або що база даних пішла в блокування. Такі речі видно лише зсередини. Тому безагентні системи зазвичай дають хороше уявлення про “здоров’я” інфраструктури в цілому, але мало що говорять про внутрішнє життя застосунків.

Ще одна особливість полягає в тому, що безагентний моніторинг живе в мережевій логіці. Йому потрібний доступ: порти, протоколи, можливість

підключення. І якщо інфраструктура сегментована, якщо між серверами стоять брандмауери або складні маршрути, безагентна система раптом починає спотикатися. Декілька закритих портів-і частина моніторингу провалюється. А інколи буває, що система працює прекрасно, але мережевий шлях до неї затінений, і тоді моніторинг показує “помилку”, яка насправді не стосується самого сервера.

У великих організаціях безагентні схеми інколи нагадують розгалужену павутину. Один центральний вузол опитує десятки сегментів, у кожному з яких свої правила доступу, свої винятки, свої обмеження. І все це потрібно підтримувати в робочому стані, хоча логіка моніторингу, здавалося б, дуже проста. У деяких компаніях навіть створюють окремі “проксі-сервери моніторингу”-такі собі проміжні пункти, через які проходять запити. І ще одна цікава деталь: будь-які зміни в мережевій політиці можуть раптом завалити моніторинг, навіть якщо на самих серверах нічого не змінювалося.

Водночас є сфери, де безагентний підхід важко замінити чимось іншим. Наприклад, коли треба моніторити мережеве обладнання. Комутатори, маршрутизатори, балансувальники-усе це зазвичай і так відповідає на запити SNMP, і ставити на них “агент” просто немає де. Так само з хмарними сервісами: багато з них не дозволяють ставити свій код “всередину”, а от власні API або метрики вони дають. Тобто моніторинг працює, але працює на умовах, які ставить сама платформа.

Цікавий момент проявляється там, де компанія намагається моніторити не сервер, а конкретну функцію-наприклад, API, який обробляє замовлення. Тут безагентні перевірки часто використовують “зовнішні” сценарії, коли тестовий запит надсилається на сервіс, і система дивиться, чи пройшов він успішно. Це вже навіть не моніторинг заліза-це перевірка реального досвіду користувача. Такі речі дають цінну інформацію, бо вони не залежать від внутрішньої архітектури: якщо тестовий запит почав відповідати повільніше, це може бути раннім сигналом.

Але знову ж: зовнішній тест не пояснює чому. Він показує симптом, але причину приховує. І в цьому-головна відмінність безагентного підходу від

агентного. Один дивиться на систему зсередини, інший-зовні. Один бачить деталі, інший-загальну поведінку.

Ще одна проблема, яка інколи проявляється несподівано: безагентні системи надто чутливі до самої мережі. Якщо мережа нестабільна або перевантажена, моніторинг починає показувати “помилки”, яких насправді немає. А адміни потім витрачають години, щоб розібратися, що сталося: сервер чи мережа? Іноді виходить так, що сервіс працює чудово, а моніторинг “панікує”. Це створює специфічний шум і знижує довіру до самої системи.

Проте є одна риса, за яку безагентний моніторинг все ж цінують. Він м’яко входить у середовище. Не вимагає ні встановлення агентів, ні оновлень, ні інтеграції з операційною системою. Якщо компанія хоче почати швидко, просто ввімкнути кілька перевірок і одразу отримати картину-це той випадок, де безагентний підхід нагадує включення світла в кімнаті. Він не дає глибину, але дає базове розуміння стану.

Тому безагентний моніторинг часто живе у двох ролях: або як швидкий спосіб охопити широку мережу без вторгнення в її вузли, або як інструмент зовнішнього контролю, що імітує точки зору користувача. Але там, де потрібно зрозуміти справжню причину проблеми, він починає розводити руками.

І, можливо, саме через ці недоліки і з’явилась думка, що тільки один підхід-агентний чи безагентний-майже ніколи не покриває всіх потреб. З часом більшість компаній приходять до того, що потрібна комбінація двох світів.

Гібридні моделі

Через деякий час після появи обох підходів-і агентного, і безагентного-багато команд уже інтуїтивно розуміли, що жоден із них сам по собі не дає повної картини. Агент показує глибину, але забирає багато сил на розгортання й обслуговування; безагентний швидко охоплює площу, але бачить поверхню. І виходило щось подібне до ситуації, коли маєш два ліхтарики: один світить далеко, але вузько, інший-широко, але недалеко. І коли тебе цікавить не один конкретний об’єкт, а вся місцевість, доводиться брати обидва.

Саме так поступово з’явилися гібридні архітектури моніторингу. Вони не

були якесь “нове слово”-скоріше вони стали відповіддю на те, що інфраструктуру нереально покрити одним інструментом. У великих компаніях це виглядало майже природно: один сегмент мережі неможливо навантажувати агентами-там використовували безагентні перевірки; інші частини, де потрібні деталі й аналіз поведінки, отримували агентів. У результаті це складалося в таку собі мозаїку спостереження, де важливо не те, який тип моніторингу використовується, а те, чи бачить команда реальну причину подій.

Гібридна схема зазвичай виростає з того, що інфраструктура дуже строката. Наприклад, у компанії можуть бути старі фізичні сервери, сучасні контейнерні платформи, віртуальні машини в хмарі, мережеве обладнання, а поруч-звичні робочі станції працівників. Усе це різні світи. Ідея, що “агент поставить все на свої місця”, виглядає занадто оптимістично. Десь агент не працює через відсутність підтримки; десь його не дозволяє ставити служба безпеки; десь це просто недоцільно. Так само й безагентні перевірки не скрізь доречні: вони не дістануться глибин сервісу, вони губляться в сегментації, а інколи чіпляються за мережеві “шуми” й створюють ілюзію проблеми.

Тому архітектори моніторингу почали будувати системи так, щоб вони могли обіймати обидва підходи одночасно. Наприклад, вузли, де важлива продуктивність застосунку, отримують агентів. Вони збирають детальні метрики, дивляться, як поводяться потоки, ловлять аномалії на рівні процесів. А більш периферійні частини, де потрібно лише зрозуміти доступність чи загальний стан, йдуть через безагентні механізми. Таке поєднання дає змогу не витрачати ресурси там, де це не потрібно, але водночас не втрачати глибину там, де без неї не обійтись.

Це поєднання має ще одну цікаву властивість: воно дозволяє вибудовувати “багатошарову” картину. Зовнішній моніторинг показує симптоми: сервіс повільний, API відповідає нерівномірно, користувачі відчувають затримки. Це як бачити хвилі на воді. А внутрішній моніторинг через агентів показує, що саме створює ці хвилі-яка транзакція гальмує, який процес упав у високий IO, який мікросервіс затримує вхідний трафік. Звідси й народжується той «ефект просвітлення», про який говорять SRE-команди: коли два шари моніторингу

накладаються один на одного, з хаосу починає проступати логіка.

У гібридних систем є ще одна природна перевага-вони краще переживають зміну середовища. Наприклад, якщо компанія переносить частину сервісів у хмару, безагентна частина може легко підхопити нові вузли через API провайдера. А якщо якісь сервіси залишаються локально й потребують більш детального нагляду, агентний шар просто продовжує працювати. Жодної революції-просто гнучкість, яка знімає біль під час міграції.

У компаніях, де інфраструктура “живе” своїм життям, гібридні моделі інколи стають єдиним способом зберегти цілісність моніторингу. Бо завжди знайдеться підсистема, де агент-зайвий, і завжди знайдеться інша, де без агента узагалі неможливо обійтись. Спроба нав’язати одну модель усім частинам інфраструктури створює більше проблем, ніж рішень: одні вузли перезавантажені, інші недодивлені, а десь доводиться придумувати складні винятки.

Ще один цікавий аспект гібридних систем-це їх зв’язок із безпекою. Коли компанія має справу з великою кількістю ролей доступу, різними рівнями сегментації, окремими регламентами для персоналу, моніторинг повинен уміти працювати в цих умовах. Агент може збирати детальну інформацію там, де це дозволено політиками. Безагентний моніторинг може дивитися на сегменти, доступ до яких обмежений. У результаті виходить система, яка не порушує правила, але все одно отримує достатньо даних, щоб аналізувати поведінку.

Гібридність добре поєднується й з аналітичними компонентами. Внутрішні дані агентів можна використовувати для побудови моделей продуктивності: як поводяться процеси, які патерни повторюються. А зовнішні безагентні сигнали дають ширший контекст: у який момент користувачі почали відчувати зміни, коли сервіс уповільнився, як це вплинуло на загальні бізнес-процеси. Модель, яка об'єднує ці два світи, уже не просто моніторить, а починає робити висновки. Десь навіть передбачати проблеми.

Про таку архітектуру іноді кажуть, що вона “вміє дивитися одночасно і всередину, і назовні”. Це влучне порівняння. Бо справді, внутрішні агенти

говорять дуже багато, але інколи надто технічно. А зовнішній моніторинг говорить занадто мало, але часто дуже влучно, бо показує те, що бачить користувач. Поєднання цих точок зору і формує ту повноту картини, якої не може дати жоден із підходів поодинці.

І, зрештою, гібридні моделі стали фактичним стандартом у більшості сучасних IT-ландшафтів. Навіть коли компанія починає з агентів, рано чи пізно з'являється потреба “подивитися з боку”. А коли починають із безагентних методів, швидко стає очевидно, що потрібна деталізація, яку дає тільки агент. І вся ця еволюція веде до того, що моніторинг перестає бути однотипним інструментом. Він стає набором різнорівневих джерел даних, які постійно взаємодіють і доповнюють одне одного.

Такі архітектури найближчі до того, що сьогодні називають інтелектуальними системами моніторингу. Бо без гібридності неможливо аналізувати складну поведінку, відстежувати зміни у роботі персоналу, дивитися на взаємозалежності між застосунками та інфраструктурою. Вони дають той фундамент, на якому вже можна будувати алгоритми виявлення аномалій, моделі прогнозування й системи ролей доступу.

1.3. Організація ролевого доступу та засоби корпоративної кібербезпеки.

На ранніх етапах розвитку IT-інфраструктур управління доступом здійснювалося переважно вручну шляхом індивідуального призначення дозволів користувачам. Такий підхід базувався на списках контролю доступу (ACL) і був прийнятним лише за невеликої кількості користувачів та низької динаміки змін. Зі зростанням організацій і ускладненням інфраструктур ці механізми втратили керованість, що призводило до накопичення надлишкових прав, помилок у налаштуваннях і зниження рівня безпеки.

Вирішенням цієї проблеми стала модель ролевого доступу (RBAC), у межах якої дозволи призначаються не окремим користувачам, а ролям, що відповідають їхнім функціям у організації. Такий підхід дозволив впорядкувати

систему доступів, спростити адміністрування та реалізувати принцип мінімальних привілеїв, відповідно до якого користувач отримує лише ті права, що необхідні для виконання поточних завдань.

З розвитком складних багатокомпонентних систем стало очевидно, що сам по собі RBAC не є достатнім. Контроль доступу потребує доповнення механізмами аудиту та журналювання дій користувачів, які дозволяють відстежувати, хто, коли і за яких умов виконував певні операції. Журнали активності стали ключовим елементом корпоративної кібербезпеки, оскільки інциденти дедалі частіше пов’язані не з технічними збоями, а з людським фактором.

Подальше ускладнення інфраструктур, поява хмарних сервісів і віддаленої роботи призвели до розмивання традиційного мережевого периметра. Це зумовило перехід до концепції Zero Trust, відповідно до якої жодна взаємодія не вважається довіреною за замовчуванням. Кожен запит на доступ перевіряється з урахуванням ідентичності користувача, його ролі, стану пристрою, місця та часу підключення, а також поточного контексту системи.

У межах цієї парадигми роль користувача розглядається не як статичний набір прав, а як основа для формування динамічних дозволів. Все більшого поширення набувають тимчасові доступи, контекстно-залежні політики та багатофакторна автентифікація, що суттєво знижує ризики, пов’язані з надлишковими або «забутими» привілеями.

Важливим напрямом розвитку корпоративної кібербезпеки став аналіз поведінки користувачів. Навіть дозволені дії можуть створювати ризики, якщо вони є нетиповими для певної ролі або виконуються в незвичному контексті.

Тому сучасні системи безпеки дедалі частіше застосовують поведінкові моделі для виявлення аномалій у діях персоналу, доповнюючи статичні політики динамічним аналізом.

У результаті контроль доступу, аудит, поведінковий аналіз і моніторинг інфраструктури перестали існувати як окремі компоненти. Вони формують єдину систему, у межах якої технічні метрики, події безпеки та дії користувачів розглядаються в спільному контексті. Така інтеграція є необхідною умовою для

ефективної роботи інтелектуальних систем моніторингу, оскільки дозволяє не лише фіксувати відхилення, а й коректно інтерпретувати їх причини.

Тому, сучасна організація ролевого доступу та засобів корпоративної кібербезпеки базується на принципах мінімальних привілеїв, контекстної перевірки, безперервного аудиту та аналізу поведінки. Без узгодженої та дисциплінованої системи доступів інтелектуальний моніторинг втрачає значну частину своєї ефективності, оскільки не здатний повноцінно враховувати людський фактор у складних IT-середовищах.

Загалом, агентні, безагентні та гібридні архітектури відрізняються за рівнем глибини моніторингу, складністю впровадження та придатністю до інтелектуального аналізу. Узагальнене порівняння основних характеристик наведених підходів подано в таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика архітектурних моделей систем моніторингу

Критерій порівняння	Агентні моделі	Безагентні моделі	Гібридні моделі
Спосіб збору даних	Через локально встановлений агент на вузлі	Дистанційне опитування через мережеві протоколи або API	Комбінація агентного та безагентного підходів
Глибина видимості	Висока: доступ до внутрішніх процесів, системних і прикладних метрик	Обмежена: доступ лише до доступних зовнішніх параметрів	Варіативна: глибока для критичних вузлів, базова для периферійних
Вплив на інфраструктуру	Помірний, залежить від конфігурації агента	Мінімальний, відсутнє втручання у вузол	Оптимізований за рахунок вибіркового застосування
Складність впровадження	Висока у великих середовищах	Низька, швидкий старт	Середня, потребує архітектурного планування
Масштабованість	Обмежена керуванням агентами	Висока за умови доступності мережі	Висока, адаптована до різнорідних середовищ

Залежність від мережі	Низька (можливе локальне кешування)	Висока (чутливість до затримок і сегментації)	Сбалансована
Рівень безпекових ризиків	Підвищений через встановлення додаткового ПЗ	Нижчий, але залежний від відкритих протоколів	Контрольований через політики доступу
Придатність для аналізу аномалій	Висока	Обмежена	Найвища
Типові сценарії застосування	Сервери, робочі станції, застосунки	Мережеве обладнання, хмарні сервіси	Корпоративні та гібридні IT-інфраструктури

1.4. Методи збору, нормалізації та зберігання телеметричних та поведінкових даних

Телеметричні дані в IT-інфраструктурах охоплюють сукупність показників і подій, які генеруються компонентами системи в процесі її функціонування. До них належать метрики використання ресурсів, журнали подій, трасування запитів, а також записи, що відображають поведінку користувачів і адміністраторів. Ці дані відрізняються за структурою, частотою надходження та семантичним навантаженням, що ускладнює їх об’єднання в єдину аналітичну модель.

Метрики зазвичай представлені числовими часовими рядами, що характеризують стан системи в конкретні моменти часу. Логи фіксують події та помилки у вигляді текстових або напівструктурованих записів. Трасування відображає послідовність виконання операцій між компонентами розподілених систем. Поведінкові дані формуються переважно як побічний продукт аудиту доступів і дій користувачів та мають значний контекстний компонент, пов’язаний із людським фактором.

Збір телеметрії здійснюється різними методами залежно від типу джерела. Для серверів і операційних систем широко застосовуються агентні підходи, за яких спеціальні процеси збирають метрики та події безпосередньо на вузлах. Для мережевого та спеціалізованого обладнання використовуються протоколи

опитування, зокрема SNMP або WMI, які дозволяють отримувати стандартизовані показники без встановлення додаткового програмного забезпечення.

Окрему категорію становлять лог-форвардери, які передають журнали подій із локальних систем у централізовані сховища. Для отримання більш глибокого контексту на рівні застосунків застосовується інструментація коду, що дає змогу фіксувати ключові етапи виконання логіки сервісів. У сучасних системах ці підходи часто стандартизуються за допомогою єдиних інтерфейсів збору телеметрії. Додатково використовуються низькорівневі методи, такі як збір подій ядра операційної системи або аналіз мережевого трафіку через механізми віддзеркалювання.

У хмарних середовищах значна частина телеметрії доступна через API провайдерів, що реалізує модель підписки на події та показники. На практиці жоден із методів не є універсальним, тому ефективний збір даних базується на комбінуванні різних каналів із урахуванням навантаження, вимог безпеки та необхідного рівня деталізації.

Після надходження в центральну систему телеметричні та поведінкові дані потребують нормалізації. Основними завданнями цього етапу є уніфікація форматів часу, приведення назв і типів полів до спільної схеми, вирівнювання частоти надходження метрик та усунення некоректних або шумових значень. Без цього кореляція подій між різними джерелами стає неточною або неможливою.

Важливим аспектом нормалізації є об’єднання пов’язаних подій у спільні логічні ланцюги. Для цього використовуються кореляційні ідентифікатори, які дозволяють пов’язати записи логів, метрики та трасування, що належать до однієї транзакції або сценарію. За відсутності таких ідентифікаторів відновлення зв’язків ускладнюється і знижує точність аналізу.

Зберігання телеметрії реалізується з урахуванням специфіки даних. Метрики зазвичай розміщуються у сховищах часових рядів, оптимізованих для швидкого запису та агрегації. Логи зберігаються у системах, орієнтованих на повнотекстовий пошук і індексацію великої кількості подій. Дані трасування потребують спеціалізованих структур для збереження ієрархії викликів. На

практиці застосовується багаторівнева модель зберігання з поділом на «гарячі», «теплі» та архівні шари.

Окремо визначаються політики зберігання даних, які регулюють глибину історії та рівень деталізації для різних типів телеметрії. Такий підхід дозволяє збалансувати вартість зберігання й аналітичну цінність інформації.

Таким чином, методи збору, нормалізації та зберігання телеметричних і поведінкових даних формують фундамент сучасних систем моніторингу. Якість цих процесів безпосередньо впливає на достовірність аналізу та ефективність інтелектуальних алгоритмів виявлення аномалій, оскільки навіть найскладніші методи обробки не компенсують хаотичні або неконсистентні вхідні дані.

1.5. Алгоритмічні підходи до аналізу подій та виявлення аномалій

Перші підходи до виявлення аномалій в IT-системах базувалися на простих порогових правилах. Якщо значення метрики перевищувало заздалегідь визначений рівень, система генерувала сповіщення. Така логіка була інтуїтивною й певний час працювала ефективно, оскільки інфраструктури були відносно простими, а відхилення — очевидними.

Зі зростанням складності систем порогові механізми втратили ефективність. Одна й та сама метрика могла бути критичною в одному контексті та нормальною в іншому, а більшість проблем почали проявлятися не в окремих показниках, а в їх поєднаннях. Це зумовило перехід до евристичних правил, які враховували кілька взаємопов’язаних сигналів і частково імітували досвід інженера. Однак такі правила вимагали ручного налаштування і не масштабувалися разом з інфраструктурою.

Наступним етапом стала кореляція подій, яка дозволила аналізувати часові зв’язки між різними компонентами системи. Завдяки цьому аномалії почали розглядатися як ланцюги взаємопов’язаних подій, а не як ізольовані спрацювання. Водночас зростання обсягів і різнорідності даних призвело до появи надмірної кількості корельованих сигналів, що ускладнювало інтерпретацію результатів.

Подальший розвиток алгоритмічних підходів пов’язаний із використанням статистичних моделей. Замість фіксованих порогів системи почали вивчати нормальну поведінку метрик і формувати адаптивні межі відхилень. Такі підходи дозволили зменшити кількість хибних сповіщень і виявляти повільні деградації, які не проявляються у вигляді різких стрибків. Водночас статистичні методи чутливі до шуму і не завжди ефективні в складних багатовимірних середовищах.

Подальший крок у розвитку виявлення аномалій пов’язаний із використанням алгоритмів, здатних аналізувати форму сигналів і багатовимірні стани системи. Кластеризаційні методи дозволяють виділяти типові режими роботи та ідентифікувати рідкісні або нетипові стани. Алгоритми на кшталт DBSCAN корисні тим, що не потребують заздалегідь визначеної кількості кластерів і можуть безпосередньо виявляти шумові точки.

Для роботи з багатовимірними даними застосовуються методи, що оцінюють відхилення у просторі кількох метрик одночасно. Вони дозволяють фіксувати ситуації, коли окремі показники виглядають нормальними, але їх комбінація є нетиповою. Такий підхід значно перевершує можливості ручного аналізу в складних інфраструктурах.

Окремий клас алгоритмів орієнтований на аналіз часових рядів. Вони дозволяють враховувати сезонність, тренди та зміну ритму роботи системи. Це дає змогу виявляти аномалії, що проявляються у вигляді поступових змін або відхилень від очікуваної динаміки, а не як одиничні піки.

Для текстових даних, зокрема логів, застосовуються методи виявлення структурних шаблонів. Система навчається типовим форматам повідомлень і фіксує появу рядків, що не вписуються у відомі структури. Часто саме такі поодинокі записи стають першими індикаторами нетипових ситуацій.

Зі зростанням ролі людського фактора в інфраструктурних інцидентах почали використовуватися підходи до аналізу поведінкових даних. Вони орієнтовані не на окремі дії, а на послідовності та контекст поведінки користувачів або сервісних облікових записів, що дозволяє виявляти нетипові сценарії взаємодії з системою.

Для великих і динамічних середовищ особливе значення мають методи навчання без учителя, які не потребують попередньої розмітки аномалій. Такі алгоритми формують модель нормальної поведінки і розглядають відхилення від неї як потенційні аномалії. Вони добре підходять для інфраструктур, де ручна класифікація подій є практично неможливою.

У розподілених мікросервісних системах важливим напрямом є аналіз трасувань і графів залежностей. Аномалії тут часто виникають не в окремому сервісі, а у взаємодії між ними. Аналіз структури ланцюгів викликів і змін у графі взаємодій дозволяє виявляти приховані проблеми, які не видно на рівні окремих метрик.

Сучасні платформи моніторингу зазвичай поєднують кілька алгоритмічних підходів. Прості методи використовуються для швидкої фільтрації очевидних відхилень, тоді як складніші моделі залучаються для аналізу нетипових і нових сценаріїв. Така багаторівнева архітектура дозволяє зменшити кількість хибних спрацювань і водночас підвищити чутливість до складних аномалій.

Таким чином, алгоритмічний аналіз подій еволюціонував від простих порогових правил до комплексних моделей, що враховують часову динаміку, багатовимірні залежності та поведінковий контекст. Це створює передумови для переходу від реактивного виявлення інцидентів до їх раннього прогнозування та проактивного управління стабільністю IT-інфраструктури.

1.6 Теоретичні передумови для інтеграції моніторингу інфраструктури та контролю персоналу

Тривалий час системи моніторингу IT-інфраструктури зосереджувалися виключно на технічних аспектах: метриках ресурсів, логах, подіях сервісів і часі відповіді. Дії людей при цьому залишалися поза фокусом — моніторинг фіксував лише технічні наслідки, але не враховував поведінкові причини, що до них призвели. Такий підхід був прийнятним у відносно статичних середовищах, але втратив ефективність із ростом складності та динамічності інфраструктур.

У сучасних системах дедалі більше інцидентів виникає не внаслідок суто технічних збоїв, а через зміну сценаріїв роботи персоналу, некоректні або нетипові дії користувачів, помилки під час оновлень і конфігурацій. У цих умовах розмежування між “технічними” та “людськими” причинами стає умовним. Моніторинг інфраструктури фіксує наслідки, тоді як контроль дій персоналу відображає першопричини, і розгляд цих аспектів окремо призводить до фрагментарного розуміння інцидентів.

Додатковим чинником інтеграції стало ускладнення архітектур: поширення мікросервісів, контейнеризації, хмарних середовищ і автоматизованих CI/CD-процесів. У таких системах навіть одна дія співробітника може викликати ланцюгові ефекти, які проявляються із запізненням і в різних компонентах. Без поєднання технічної телеметрії з інформацією про дії людей ці зв’язки часто залишаються непомітними.

Унаслідок цього поведінкові дані персоналу почали розглядатися як окремий клас телеметрії. Вони так само, як і технічні події, мають часову прив’язку, контекст і можуть бути корельовані з іншими сигналами. Цей підхід підтримується і сучасними вимогами корпоративної кібербезпеки, де журнали дій користувачів розглядаються не лише як засіб захисту, а і як джерело аналітичної інформації, що доповнює моніторинг інфраструктури.

Теоретично інтеграція базується на ідеї спільного контексту подій. Найпростішим рівнем такого зв’язку є часовий збіг, але в складних системах цього недостатньо. Для встановлення причинно-наслідкових залежностей використовуються контекстні ланцюги подій, ідентифікатори сесій і запитів, а також узагальнення множинних дій у логічні операції. Це дозволяє розглядати технічні збої не як ізольовані точки, а як результат послідовностей людських і системних дій.

Важливу роль відіграє аналіз поведінкових патернів. Дії, які формально не порушують політик доступу, можуть бути нетиповими для конкретної ролі або сценарію роботи. У поєднанні з технічними реакціями системи такі відхилення дозволяють виявляти приховані ризики ще до появи явних збоїв.

У результаті інтеграція моніторингу інфраструктури та контролю персоналу формує єдиний інцидентний контекст, у межах якого поєднуються відповіді на питання “що сталося”, “чому це сталося” і “які дії цьому передували”. Це дозволяє перейти від фрагментарного аналізу до цілісного бачення IT-системи як складного соціотехнічного середовища.

Таким чином, теоретичні передумови інтеграції ґрунтуються на розумінні того, що технічні та поведінкові події є частинами одного безперервного процесу. Моніторинг інфраструктури надає форму спостереження, тоді як контроль дій персоналу наповнює її змістом. Лише їх поєднання створює основу для глибокого аналізу інцидентів, раннього виявлення нестабільності та побудови інтелектуальних систем моніторингу.

РОЗДІЛ 2. ПІДХОДИ ДО ПОБУДОВИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

Практичний аналіз роботи корпоративної IT-інфраструктури показує, що її поведінка суттєво відрізняється від формалізованих моделей, описаних у методичних рекомендаціях. У реальних умовах сервіси працюють нерівномірно, затримки можуть виникати й зникати без очевидних причин, а окремі компоненти реагують на події із запізненням. Ці технічні процеси тісно переплітаються з активністю користувачів, утворюючи складну динамічну систему, у якій дії персоналу та інфраструктурні сигнали взаємно впливають один на одного.

Попри велику кількість зібраних даних, класичні підходи до моніторингу часто не дозволяють отримати цілісне уявлення про ситуацію. Логи, метрики, мережеві показники та події безпеки зазвичай існують у відокремлених системах і аналізуються незалежно. Кожен інструмент фіксує власний фрагмент реальності, не враховуючи контексту інших джерел. У результаті навіть повні набори даних залишаються розрізненими і не пояснюють реальних причин інцидентів.

Під час аналізу практичних сценаріїв стає очевидно, що проблема полягає не стільки у відсутності моніторингу, скільки в обмеженості способу інтерпретації подій. Складні взаємозв’язки між технічними параметрами та поведінкою користувачів намагаються пояснити через статичні графіки, які не відображають динаміку процесів. У таких умовах деградація сервісу може виглядати як технічна несправність, хоча її джерелом є поведінкові чинники, або навпаки.

Це формує потребу у підході, який не розділяє інфраструктуру та персонал на незалежні рівні, а розглядає їх як частини єдиного середовища. Поєднання технічних і поведінкових сигналів дозволяє виявляти закономірності, які залишаються прихованими при ізольованому аналізі. Особливого значення набуває різна ритміка даних: одні події виникають постійно, інші — лише у моменти активності користувачів. Без узгодження цих часових масштабів контекст подій втрачається.

У сучасних організаціях IT-інфраструктура безпосередньо впливає на бізнес-процеси та продуктивність персоналу. Короткочасні збої, які не фіксуються середніми значеннями метрик, можуть призводити до зривів робочих сесій, обривів з'єднань або втрати доступу до сервісів. Це створює ситуацію, коли формально інфраструктура виглядає стабільною, але фактичний користувацький досвід свідчить про протилежне.

Особливо гостро ця проблема проявляється у малих і середніх організаціях, де інфраструктура часто базується на доступному мережевому обладнанні, а класичні системи моніторингу орієнтовані переважно на статистичні показники. Такі рішення ефективно візуалізують метрики, але слабо відображають подієву та поведінкову складову, яка фактично формує інциденти. У цих умовах постає питання побудови моніторингу, що відповідає динамічній природі реального середовища.

Додатковим чинником є обмеженість людських ресурсів. Адміністратори часто поєднують кілька ролей і не мають можливості постійно аналізувати складні дашборди. Це зумовлює потребу в системі, яка формує стислі, подієві сигнали та оперативно повідомляє про значущі зміни. У цьому контексті подієво-орієнтований підхід виглядає більш адекватним, оскільки подія фіксує реакцію системи, а не лише її усереднений стан.

Таким чином, побудова інтелектуальної системи моніторингу розглядається не як створення чергового інструмента збору статистики, а як пошук механізмів виявлення взаємозв'язків між технічними подіями та поведінкою користувачів. Саме поєднання цих сигналів дозволяє перейти від фрагментарного спостереження до цілісного розуміння процесів, що відбуваються в корпоративній IT-інфраструктурі.

2.1. Огляд існуючих інструментів моніторингу IT-інфраструктури та їх функціональних обмежень

Аналіз традиційних інструментів моніторингу IT-інфраструктури показує, що попри зовнішню еволюцію — нові плагіни, дашборди, API — їх внутрішня

філософія практично не змінилася. Більшість таких систем побудовані навколо моделі періодичного опитування та фіксованих тригерів, що добре працювало в умовах статичних і відносно простих інфраструктур, але погано відповідає сучасним динамічним середовищам.

Класичні рішення, такі як Zabbix, Nagios, Icinga та PRTG, орієнтовані на контроль стану окремих компонентів. Вони ефективно відповідають на питання «працює чи ні», але майже не намагаються пояснити, *чому* відбулося відхилення. Події розглядаються ізольовано, без спроби побудови причинно-наслідкових зв’язків між ними. У результаті система може фіксувати десятки алертів, які насправді є наслідком одного процесу, але не здатна виділити його як першопричину.

Суттєвим обмеженням таких інструментів є відсутність роботи з контекстом. Вони добре вимірюють стабільність, але не розуміють логіку функціонування середовища. Більшість проблем у реальних системах виникають поступово — через деградацію, накопичення затримок, зміну ритму роботи. Традиційний моніторинг реагує лише на різкі порушення, коли інцидент уже відбувся, що робить його суто реактивним.

Окремою проблемою є повна відсутність у таких системах моделі людської поведінки. Зміни навантаження, спричинені робочою активністю персоналу, трактуються як технічні аномалії, хоча в реальності можуть бути нормальними для певного часу або процесу. Система не співставляє технічні сигнали з діями користувачів, через що аналітика втрачає зміст і перетворюється на фіксацію симптомів.

З переходом до більш динамічних архітектур — контейнерів, мікросервісів, CI/CD — у багатьох середовищах почали використовувати метрик-орієнтовані платформи на кшталт Prometheus та VictoriaMetrics. Вони значно покращують збір і зберігання часових рядів та дозволяють аналізувати динаміку показників у часі. Проте їх базова модель зводить інфраструктуру до набору чисел, позбавлених подієвого та поведінкового контексту.

Метрики добре описують стан, але майже не пояснюють поведінку. Зміна значення фіксується, але причина залишається поза межами системи. Людська

активність не вписується у формат часових рядів без суттєвих спотворень, тому поведінкові сигнали фактично виключені з аналізу. Алертинг у таких системах здебільшого базується на ручних порогах або простих статистичних правилах, які не враховують контексту роботи середовища.

Логові платформи — ELK/EFK, OpenSearch, Splunk, Graylog — забезпечують детальну фіксацію подій і є цінним джерелом технічної інформації. Проте вони переважно виступають інструментами пошуку, а не пояснення. Логи реєструють події, але не формують між ними зв’язків. Побудова причинних ланцюгів у таких системах покладається на людину, що стає практично неможливим у великих середовищах з мільйонами подій.

APM та observability-платформи (Datadog, New Relic, Dynatrace) роблять крок уперед, поєднуючи метрики, логи та трасування і будуючи мапи залежностей між сервісами. Вони ефективно аналізують поведінку додатків, але залишаються обмеженими технічною площиною. Поведінка персоналу, робочі ритми, зміни сценаріїв використання системи перебувають поза їхньою моделлю. Навіть складні алгоритми аномалій у таких рішеннях часто вказують на відхилення, не пояснюючи їх контексту.

Загальною проблемою для всіх розглянутих класів інструментів є фрагментованість даних: різні часові шкали, формати, рівні деталізації та відсутність єдиного простору інтерпретації. Жодна з систем не виступає центральною точкою істини, яка здатна об’єднати технічні події, роботу додатків і поведінку людей в єдину аналітичну модель.

У підсумку існуючі інструменти моніторингу залишаються важливими компонентами IT-екосистеми, але не можуть бути основою для інтелектуального аналізу. Вони ефективно показують *що* сталося, але майже не відповідають на питання *чому*. Саме це обмеження створює передумови для побудови інтегрованої інтелектуальної системи, яка не замінює наявні рішення, а надбудовується над ними, поєднуючи технічну телеметрію з поведінковими даними та дозволяючи переходити від реактивного реагування до осмисленого аналізу причин.

Узагальнену характеристику основних класів інструментів моніторингу та їхніх функціональних обмежень наведено в таблиці 2.1.

Таблиця 2.1

Порівняльна характеристика існуючих інструментів моніторингу IT інфраструктури та їх функціональних обмежень

Клас інструментів (приклади)	Основний тип даних	Функціональні можливості	Ключові обмеження
Класичні системи моніторингу (Zabbix, Nagios, Icinga)	Статуси, базові метрики	Контроль доступності хостів і сервісів, тригери, сповіщення	Реактивна логіка, відсутність контексту та причинно-наслідкових зв’язків
Метрик-орієнтовані платформи (Prometheus, VictoriaMetrics)	Часові ряди метрик	Збір і зберігання великої кількості числових показників	Аналіз лише числових змін без пояснення причин
Логові системи	Події, текстові логи	Детальна фіксація подій, гнучкий пошук	Високий рівень шуму, відсутність автоматичної кореляції
APM / Observability платформи (Datadog, New Relic, Dynatrace)	Метрики, трейси, логи	Аналіз продуктивності додатків, побудова технічних залежностей	Фокус на технічних процесах, ігнорування поведінки персоналу
Комбіноване використання інструментів	Різнорідні дані	Широке покриття інфраструктури	Відсутність єдиної аналітичної моделі та централізованої інтерпретації

2.2. Бізнес-вимоги до контролю продуктивності персоналу та операційної ефективності

Коли фокус зміщується з технічного рівня на бізнесовий, виникає відчуття переходу з лабораторних умов у реальне життя. Для технічних команд

інфраструктура — це сервери, сервіси, мережі, логи та навантаження. Для бізнесу ж це лише середовище, яке або підтримує роботу людей, або створює їй зайві перешкоди. Саме тому питання продуктивності персоналу в корпоративному контексті виникає не як бажання контролювати співробітників, а як спроба зрозуміти, чому результати інколи не відповідають витраченим ресурсам.

У більшості компаній фонд оплати праці є одним з найбільших витратних блоків, і логічно, що керівництво прагне розуміти ефективність цих інвестицій. Проте оцінити її інтуїтивно практично неможливо. Зовні все виглядає стабільно: люди працюють, задачі виконуються, процеси рухаються. Але при детальнішому аналізі проявляються численні дрібні втрати часу й уваги, які не фіксуються жодними формальними показниками.

Робочий день співробітника часто складається з десятків мікрозатримок: повільні відповіді систем, зайві дії в інтерфейсі, часті перемикання між задачами, очікування результатів внутрішніх процесів. Кожна з таких затримок незначна, але в сумі вони формують години втраченої продуктивності. Ці втрати залишаються майже непомітними — ні для керівництва, ні для IT, ні навіть для самого працівника.

У цей момент бізнес стикається з проблемою суб’єктивності. Керівники бачать падіння ефективності, команди пояснюють це технічними складнощами, IT наполягає на стабільності інфраструктури, а HR не фіксує відхилень. Усі мають рацію частково, але відсутність спільного набору даних не дозволяє поєднати ці погляди в єдину картину. Інформація існує фрагментовано, без центральної логіки.

Додаткову складність створює непередбачуваність робочих процесів. Бізнес очікує стабільного темпу, прогнозованих строків і рівномірного навантаження. У реальності ж продуктивність може коливатися без очевидних причин. Часто ці коливання пояснюють людськими факторами, хоча на практиці причиною можуть бути незначні технічні деградації, які змушують людей витрачати додатковий час на кожну дію. Без аналітики такі ситуації виглядають як «погана робота», хоча насправді це наслідок умов, у яких працює персонал.

Окремий бізнесовий інтерес пов’язаний із процесними втратами. Навіть у сильних командах значна частина часу може витрачатися на повторювані операції, ручні виправлення, дублювання даних або неузгоджені внутрішні правила. Ці втрати складно оцінити без вимірювань, але саме вони часто знижують загальну ефективність значно більше, ніж індивідуальні помилки.

Стратегічно бізнесу важлива передбачуваність. Коли продуктивність коливається без зрозумілих причин, порушується планування, з’являються затримки й дисбаланс навантаження. У цьому контексті контроль продуктивності — це не про мікроменеджмент, а про керованість і стабільність операційних процесів.

Ще один важливий аспект — ризики. Вузькі місця, залежність від окремих співробітників, нерівномірний розподіл навантаження або неформальні практики роботи створюють приховані загрози для компанії. Ці ризики часто не пов’язані з «поганою роботою», а є наслідком відсутності системного бачення.
Таким чином, бізнес потребує контролю продуктивності не як інструменту спостереження за людьми, а як засобу розуміння того, як функціонує організація загалом. Де виникають затримки, які процеси блокують результат, як інфраструктура впливає на роботу персоналу і навпаки. Без поєднання технічних метрик і поведінкових даних цей аналіз неможливий: кожен із цих підходів дає лише частину реальності.

Тому системи вимірювання продуктивності в бізнесовому сенсі є насамперед діагностичними. Вони покликані виявляти закономірності та проблемні зони, а не здійснювати контроль за кожною дією. Їх ефективність залежить від балансу між збором даних, прозорістю, довірою та етичними межами. Надмірний контроль руйнує саму ідею аналітики.

При цьому продуктивність не зводиться до одного показника. Вона формується з індивідуального темпу, командної динаміки та якості процесів. Бізнесу важливіше бачити, де саме губиться час, ніж оцінювати окремих людей. Часто саме процесні втрати, а не індивідуальні особливості, мають вирішальний вплив на результат.

Окремої уваги потребує межа між продуктивністю та втомою. Вигорання проявляється не різко, а поступово — через збільшення кількості помилок, часті перемикання між задачами, зростання часу виконання простих дій. Такі сигнали складно помітити традиційними засобами, але вони добре проявляються в поведінкових патернах.

Водночас цифри без контексту не мають сенсу. Зміна темпу роботи може мати десятки причин — від технічних проблем до складності задач або фізичного стану людини. Тому бізнес очікує від системи не лише фіксації дій, а й можливості інтерпретації, без перетворення персоналу на набір абстрактних показників.

Критичною є й прозорість. Якщо співробітники не розуміють, які дані збираються і для чого, система не працює. У здорових організаціях контроль продуктивності розглядається як інструмент діагностики, а не тиску. Це потребує чіткої ролева́ї моделі доступу: керівництво бачить тенденції, HR — загальну динаміку, IT — технічні сигнали, без надмірної деталізації індивідуальної поведінки.

У підсумку бізнес формулює запит не на вимірювання заради вимірювання, а на отримання індикаторів, які мають практичний сенс: час очікування, повторювані цикли, частоту переривань робочого процесу. Це ті параметри, які безпосередньо впливають на результат, але рідко усвідомлюються без аналітики.

Найважливіше ж полягає в тому, що система має допомагати, а не тиснути. Довіра до інструменту дозволяє бізнесу отримати об’єктивність, а співробітникам — відчуття справедливості та видимості їхньої роботи. У цьому сенсі вимірювання продуктивності стає способом зменшення хаосу й підвищення передбачуваності, а не інструментом контролю.

З точки зору операційної ефективності, ключовим стає розуміння взаємодії між людьми та інфраструктурою. Більшість проблем лежить у проміжній зоні між технічним і людським. Саме тому бізнес формує вимогу до інтегрованої системи, яка здатна бачити не лише окремі сигнали, а й їх взаємний вплив,

дозволяючи перейти від фрагментарних спостережень до цілісного розуміння того, як компанія працює щодня.

2.3. Моделювання потоків даних та інтеграція різнорідних джерел інформації

Під час аналізу формування потоків даних у середніх і великих організаціях швидко стає зрозуміло, що дані існують незалежно від того, чи хтось їх системно збирає. Вони виникають у кожній дії користувача, кожному запиті до системи, кожній мікрооперації автоматизованого сервісу. Технічні компоненти, мережа та люди працюють у різних ритмах, але всі ці процеси зрештою мають складатися в єдину аналітичну картину.

Інфраструктурні дані, як правило, формуються більш впорядковано: метрики серверів, технічні події, журнали запитів. Вони мають структурований вигляд і передбачувану форму, але генеруються з різною частотою. Лог може з’явитися в будь-який момент, тоді як метрики надходять за фіксованим інтервалом, а поведінкова дія користувача взагалі не прив’язана до технічних циклів. У результаті загальна картина складається з фрагментів, зафіксованих у різний час і з різних точок зору.

Поведінкові дані суттєво відрізняються за природою. Якщо технічні метрики описують кількісні показники, то поведінкові сигнали відображають логіку дій: початок, зупинку, очікування, повернення до завдання. Такі дані складно звести до числових рядів без втрати змісту, оскільки вони потребують контексту для інтерпретації.

Додаткову складність створює різний рівень деталізації. Сервер може згенерувати сотні подій за секунду, тоді як одна дія користувача може стати причиною каскаду цих подій. У сирому вигляді вони виглядають незалежними, хоча насправді є частинами одного процесу. Саме тому моделювання потоків даних не зводиться до механічного об’єднання записів, а потребує узгодження різних “історій” в єдину логічну послідовність.

Окремим викликом є часовий аспект. Час фактичного виконання дії і час її фіксації в системі не завжди збігаються. Затримки мережі, черги запису, асинхронна обробка призводять до зсувів, які можуть суттєво спотворювати аналітику. Ігнорування цієї різниці створює хибні причинно-наслідкові зв’язки.

Потоки даних часто є суперечливими. Інтерфейс може сигналізувати про успішну операцію, тоді як лог сервера фіксує помилку. Такі розбіжності не завжди свідчать про збій — вони відображають різні рівні системи, кожен з яких має власну перспективу. Без інтеграції ці сигнали формують парадоксальну картину, хоча всі вони є коректними у своїх контекстах.

Ще одним фактором є “сліпі зони” — ситуації, коли технічні показники не демонструють проблем, але користувачі відчувають деградацію роботи. Причини можуть полягати у нестандартних сценаріях використання, короткочасних мережевих затримках або особливостях інтерфейсу. Без поєднання технічних і поведінкових даних такі ситуації залишаються незрозумілими.

Інтеграція ускладнюється також наявністю дублікатів, втрат подій і різною надійністю джерел. Надлишок даних створює шум, нестача — спотворює висновки. До цього додаються конфліктні записи, які виникають через різні точки фіксації однієї події. Людський фактор лише підсилює цю нестабільність: поведінкові дані за своєю природою нерівномірні та непередбачувані.

У підсумку інтеграція різнорідних джерел інформації є не технічною операцією, а задачею узгодження смислів. Просте об’єднання таблиць або логів не створює цінності без додаткового шару логіки, який дозволяє інтерпретувати події в контексті.

Інтегрована модель даних у такій системі повинна виступати своєрідним “шаром перекладу” між технічним і поведінковим світом. Вона має бути гнучкою, щоб адаптуватися до змін ритму роботи людей і інфраструктури, але водночас достатньо структурованою, щоб запобігати хаосу.

Базовим елементом такої моделі стає спільний таймлайн, який об’єднує події різних типів у логічні історії. Дії користувача, технічні запити та відповіді

системи повинні розглядатися не як незалежні записи, а як елементи одного процесу. Без цього аналітика залишається поверхневою.

Наступним етапом є нормалізація, яка передбачає приведення різних форм подій до спільної семантики. Різні позначення помилок, статусів або результатів повинні інтерпретуватися системою як пов'язані явища, без втрати контексту.

Ключовим є контекстуальний шар, який дозволяє співставляти технічні сигнали з поведінковими причинами. Зміна метрики може бути наслідком дій користувачів, технічного оновлення або поєднання кількох факторів. Інтегрована модель повинна зберігати ці зв'язки, а не спрощувати інтерпретацію.

У результаті модель набуває оповідної логіки: вона не просто зберігає події, а формує послідовності, які пояснюють, що саме відбулося. Побудова зв'язків між поведінкою, інфраструктурою та результатами дозволяє бачити взаємодії, а не окремі сигнали.

Загальною складовою є робота із затримками. Оскільки повної синхронності досягти неможливо, система повинна враховувати допустимі зсуви та орієнтуватися не лише на точний час, а й на логічну послідовність подій.

Та у підсумку інтегрована модель даних стає шаром інтерпретації, який дозволяє перетворити розрізнені сигнали на цілісну картину. Саме вона є основою інтелектуальної системи, здатної не лише фіксувати події, а й пояснювати їх у технічному, поведінковому та бізнесовому контекстах одночасно.

2.4. Аналіз загроз і ризиків інформаційної безпеки у контексті розроблюваної системи

Коли починаєш дивитися на систему, яка одночасно стежить за інфраструктурою і збирає поведінкові сигнали з робочих місць, то доволі швидко стає помітно, що ця конструкція живе за власними правилами. Вона не схожа ні на класичний технічний моніторинг, ні на типові корпоративні аналітичні платформи. Усе надто змішане. І саме тому загрози тут теж утворюються якось інакше, інколи навіть несподівано.

Звичайні моніторингові рішення ризикують у передбачуваний спосіб: хтось намагається залізти на сервер, підмінити агент, підчистити логи, і всі розуміють, що від них очікувати. А коли в екосистемі з'являється поведінковий шар, структура ризиків починає хитатись у бік чутливих речей. Бо технічні дані самі по собі — це цифри, а поведінкові події дуже швидко стають майже персональними. І будь-яка спроба втрутитися в них не просто шкодить системі, а фактично впливає на сприйняття того, що відбувається в компанії.

Проблема в тому, що така система зібрала в одному місці занадто багато того, що в інших рішеннях розкидано по різних сервісах. І саме концентрація робить її привабливою мішенню. Якась дрібниця у логі поведінки, один неточний запис, невеликий зсув у часі — і вже аналітика виглядає так, ніби хтось навмисно занижує чи підвищує продуктивність. Не потрібно навіть складної атаки; достатньо втрутитися в один канал між агентом і збором даних, і система втрачає здатність показувати правдивий контекст.

Ще одна цікава, хоч і неприємна річ — кількість каналів, через які ці дані передаються. Десь агент знімає технічні метрики, десь окремий модуль фіксує дії користувача, а паралельно ще існує внутрішній API для обміну подіями. І кожен із цих потоків може бути слабшим за інший. Наприклад, один розробник колись залишив тимчасовий ключ, хтось не встиг увімкнути перевірку цілісності пакетів, або обмеження доступів з часом розмилися — усе це відкриває щілини, через які можна непомітно просунутись углиб.

Найбільш підступні загрози часто не зовні, а зсередини. Не тому, що хтось хоче нашкодити, а тому що велика кількість доступів завжди створює хаос. Один адміністратор має права на зміну конфігурацій, інший — на аналітичні дані, третій — на поведінкові сигнали, і система раптом опиняється в ситуації, де надто багато людей бачать те, що їм не потрібно. І якщо серед них знайдеться хтось недоброчесний або просто необережний, наслідки можуть виявитися значно гіршими, ніж у випадку стандартної IT-атаки.

Ще один момент — підміна поведінкових даних. Це навіть звучить дивно, але такі атаки складно помітити. У технічному шарі все тримається на контрольних сумах, відстеженні послідовності, перевірці системних журналів. А

поведінкова активність — це патерни. І якщо його хтось вручну сфальсифікує, він може виглядати "нормально". Людина зробила десять дій — так показує система. А чи це вона? Чи це не вона? Без додаткових маркерів визначити складно. Підміна навіть одного шматка даних може створити хибне враження про те, що відбувається насправді.

Ще складніше стає, коли до цього додається тема приватності. Технічні дані — це "холодні" цифри. А сигнали поведінки — це вже не зовсім техніка. Там з’являється межа, за яку переходити небезпечно і в юридичному, і в етичному сенсі. І будь-який витік з такої системи має зовсім інший масштаб шкоди. Це не просто "хтось викрав логи". Це компрометація внутрішньої роботи людей. Тому система повинна мати не просто технічний захист, а й захист від надмірного збору, від небезпечних ролей, від ситуації, коли одна людина бачить більше, ніж має право.

І, певно, найбільш неприємний сценарій — коли атака або збій стосується не одного елементу, а ламає логіку всієї аналітики. У звичайних системах пошкодження метрики — це дрібниця: відновили і пішли далі. Тут же кожна подія вплетена в інші. Якщо зламати тільки одну, весь ланцюжок починає виглядати недостовірно. Система більше не розуміє, що відбулося раніше, що пізніше, а що було наслідком, а не причиною. Це вже не технічна, а концептуальна втрата.

Тому ці системи потребують зовсім іншого підходу до аналізу загроз: не захищати "сервер", не захищати "канал", а захищати саму історію подій. А це значно складніше, ніж традиційні методи, бо історія — це вже інтерпретація, а не просто набір битів.

Коли починаєш розкладати потенційні загрози по поличках, раптом виявляється, що чим складніша система, тим простіше її заплутати. І дивно те, що не завжди найбільш небезпечні речі приходять ззовні. Іноді зовнішній зловмисник виглядає майже примітивно порівняно з тим, що може зробити невелика група людей всередині компанії або навіть одна людина з неправильним рівнем доступу.

Почати, мабуть, варто з внутрішніх загроз. Вони завжди незручні, бо про них у корпоративному середовищі не надто люблять говорити. Проте в системах, які збирають поведінкові події, такі ризики відчуваються набагато гостріше. Адміністратор, який має широкі повноваження, теоретично може переглядати або редагувати дані так, що це не одразу виявиться. Не тому, що він «поганий», а тому що сама структура системи інколи дозволяє це зробити — десь API занадто довірливий, десь логування дій адміністратора неповне, десь немає відокремленого шару аудиту. І для того, хто розбирається у внутрішній архітектурі, маніпуляція виглядає не як злочин, а як технічний хак.

Є й більш прямолінійні сценарії: співробітник, який намагається "виправити" власні поведінкові події. Це трапляється нечасто, але трапляється. Наприклад, якщо хтось боїться, що низькі показники продуктивності вплинуть на оцінку його роботи, він може спробувати обійти систему. Перезапустити агент. Навмисно створити патерн, який виглядає як активність. Або, навпаки, приховати реальні дії, якщо вони не виглядають переконливо. І що найгірше — поведінкові дані не мають чіткої структури, тому інколи ці маніпуляції можуть виглядати як нормальна робоча поведінка.

Зовнішні загрози виглядають менш психологічно, але більш системно. Для атаки на платформу моніторингу не обов’язково пробивати DMZ чи намагатися проникнути в інфраструктуру компанії. Достатньо атакувати агент, який збирає дані. Маленький компонент, часто встановлений на кожному робочому місці, стає дуже привабливою точкою входу. Якщо його зламати, можна не лише заважати передачі подій, а й підміняти їх. Тобто система покаже абсолютно іншу картину, і керівництво навіть не здогадається, що щось змінилося.

Ще один варіант — атаки на транспортний канал. Навіть якщо він зашифрований, завжди є ризик, що на проміжних вузлах можна створити фрагментацію пакетів або викликати затримки, які спотворять часову послідовність подій. Ніби дрібниця, але коли система працює на кореляції часу, такі «шумові атаки» можуть ламати аналітичні висновки. І найгірше — це не виглядає як злому. Це виглядає як повільна мережа або нестабільність каналу.

Окремо є сценарій з переповненням системи. Якщо завалити її надмірною кількістю технічних подій або фальшивих поведінкових сигналів, аналіз стає неробочим. Це не DDoS у класичному сенсі, але щось близьке: система втрачає можливість правильно групувати події, пропускає ключові моменти, починає плутати логи. І замість того, щоб фіксувати продуктивність, вона переймається тим, щоб просто вижити під навантаженням.

Є ще одна, трохи парадоксальна загроза: конфліктні джерела. Якщо зловмисник підміняє події лише в одному вузлі, то інші джерела даних можуть почати суперечити йому. Це добре, так? На жаль — ні. Бо система може не бути готовою до таких суперечностей і почне будувати неправильні висновки, комбінуючи фрагменти, які не повинні існувати одночасно. І тут вже не просто проблема безпеки, а проблема логіки. Як відновити правду, якщо два сигнали виглядають правдоподібними, але вказують у різні боки?

Ще буває, що атакують не систему і не агента, а користувача. Через соціальну інженерію, наприклад. Перехоплення доступу, маніпуляція довірливістю, банальні фішингові листи. І тоді зловмисник може отримати доступ до аналітики або панелі керування навіть без технічного злому. У таких системах панель — це фактично центр впливу, тому її компрометація може мати значно більший ефект, ніж компрометація одного сервера.

Якщо хоча б частково скласти всі ці ризики в одну картину, стає зрозуміло, що атакуючі сценарії тут не ізольовані. Вони перетинаються, посилюють один одного. Маніпуляція даними у поведінковому шарі може нічим не відрізнятися від дрібного технічного збою, а зовнішня атака може зовні виглядати як звичайна внутрішня помилка. І саме поєднання цих деталей робить систему вразливою не через одну конкретну слабкість, а через те, що вона занадто складна, щоб утримувати стабільність без спеціальної захисної логіки.

Коли доходиш до питання, як саме будувати захист для системи, де змішано технічні події й поведінку людей, раптом стає зрозуміло, що класичні рекомендації тут працюють тільки частково. Вони потрібні, але вони не покривають найважливішого — хаосу, який виникає на стику різних типів даних. І саме тому архітектура має бути не просто "захищеною", а ще й стійкою до

неправильних інтерпретацій, до внутрішніх помилок, до людського фактору. Тобто, по суті, вона повинна тримати удар не лише технічний, а й структурний.

Перше, що зазвичай доводиться переосмислювати, — це ролі. І не в теоретичному сенсі, а в тому дуже приземленому, де кожна роль бачить тільки те, що їй потрібно. Система з поведінковими даними не може дозволити, щоб одна людина мала доступ до всього. Це не просто небезпечно, це руйнує довіру. Тому модель доступів вибудовується навколо мінімальних привілеїв. Керівник бачить агрегати. Технічні фахівці — технічний шар. Аналітики — знеособлену статистику. І лише окремий спеціальний контур може працювати з деталізацією, але й там доступ має бути тимчасовим і під аудиторським контролем.

Наступний принцип — сегментація даних. Це звучить ніби очевидно, але в подібних системах є одна особливість: просте розділення на «таблиці» не працює. Потрібно логічно розшаровувати інформацію: технічні події окремо, поведінкові — окремо, кореляції — в іншому шарі. Бо якщо зламати кореляційний шар, можна вкрасти сам сенс даних. Якщо зламати поведінковий — можна підмінити активність користувачів. Якщо технічний — створити ілюзію збою там, де його не було. Тому розділяти доводиться не для зручності, а для того, щоб атака не поширювалась між шарами.

Ще один фундаментальний елемент — цілісність. У системах цього типу важливо не лише зберігати дані, а й доводити, що вони не були змінені. Причому доводити автоматично, без додаткових ручних перевірок. Це може бути контроль хешів, цифрові підписи, верифікація часових міток, навіть окремий журнал, який фіксує появу та структуру кожної події. Якщо такої перевірки немає, будь-яка точкова маніпуляція перетворюється на «правду», і система вже не може сама себе захистити.

Транспортний рівень також стає критичним. Не через страх перехоплення (хоч і це теж важливо), а через можливість спотворення. Якщо пакети доходять із затримкою, система починає будувати хибний таймлайн. І тут захист — це не тільки шифрування, а й перевірка консистентності: чи можна довіряти порядку подій, який система отримала? А якщо ні, що тоді робити — відкидати,

перераховувати, зберігати як «підозріле»? Це рішення, які мають бути вшиті в архітектуру, а не додані пізніше.

Окрема історія — аудит. І не той, де раз на півроку перевіряють журнали, а постійний, автоматичний, який фіксує все, до чого торкався адміністратор чи інший привілейований користувач. У такої системи є одна вразливість: внутрішні дії часто важко відрізнити від легітимних. Тому потрібен механізм, який дозволяє бачити, що змінилося, ким, коли і чи виглядає це логічно. Якщо аудиту немає — архітектура стає сліпою до власних внутрішніх ризиків.

Далі — захист від надмірного збору. І це не суто технічне питання, а радше етичне. Якщо система збирає занадто багато, з'являються зайві ризики, а користувачі починають сприймати її як інструмент контролю над собою. Проблема в тому, що в такому середовищі змінюється поведінка — люди намагаються обходити систему, боятися робити окремі дії, штучно створювати активність. І виходить парадокс: замість того, щоб давати точні дані, система провокує спотворення. Захист від надмірного збору — це спосіб не лише знизити ризик витоку, а й втримати організацію нормального робочого середовища.

І нарешті, баланс. Не всі системи безпеки повинні бути надто суворими. Якщо перестаратись, система перестане працювати через власну бюрократію. Захист має бути розумним, а не «для галочки». І тут, мабуть, ключовий момент у побудові такої архітектури — розуміння, що безпека не може йти проти продуктивності. Вона повинна бути непомітною, прямолінійною, природною. Бо якщо безпекові механізми заважають аналізу або ускладнюють робочі процеси, система буде безпечною, але марною.

У кінцевому підсумку архітектура такої системи стає схожою на конструкцію, де безпека і логіка аналізу взаємно підсилюють одна одну. Одні механізми ловлять технічні атаки, інші — поведінкові маніпуляції, треті — внутрішні ризики. І якщо все це зібрано правильно, система не просто витримує загрози, а й зберігає довіру до своїх висновків. А це, чесно кажучи, важливіше за будь-які окремі метрики чи журнали.

2.5. Методичні принципи формування єдиного аналітичного ядра системи

Як тільки намагаєшся з’єднати технічний моніторинг із поведінковими патернами, виявляється, що без якогось центрального «мозку» система починає поводитись як купа окремих сенсорів, які не знають одне про одного. Вони збирають дані, але ці дані висять у повітрі, не даючи жодного нормального висновку. І саме тому поняття аналітичного ядра виникає не як красива архітектурна ідея, а як вимушена відповідь на хаос, який з’являється, коли в системі забагато джерел і занадто різні типи подій.

Сама по собі технічна інформація — це просто ряди цифр. Поведінкова — набір дій. Процесна — опис життєвого циклу задачі. І кожна з цих реальностей існує наче окремо, доки хтось не з’єднає їх у спільну картину. Проблема в тому, що цей "хтось" у складних системах уже не може бути людиною. Надто багато подій, надто різні формати. Виходить, що треба мати щось, що постійно дивиться на всі ці потоки й намагається вловити не лише факт, а й сенс — що призвело до чого, яка дія стала причиною, яка стала лише наслідком.

І тут з’являється ключова логіка: аналітичне ядро не просто об’єднує дані, воно створює спосіб мислити про них. Це не діаграма, не процесор і не статистичний модуль. Це механізм, який намагається зібрати реальність так, щоб вона була зрозумілою. У цій точці ядро працює як інтерпретаційний центр: воно збирає події, розкладає їх на фрагменти, дивиться, чи співпадають часові мітки, чи є логічні зв’язки, і чи можна пояснити закономірності.

Іноді саме ядро доводиться робити досить хитким: не тому, що це красиво, а тому що поведінкові дані рідко бувають точними. Люди не працюють по циклам, і навряд будь-яка система здатна зіпхнути це у сувору модель. Отже, аналітичне ядро має не тільки збирати “чисте”, а й розуміти “нечисте”: затримки, дивні відхилення, незавершені цикли, повторні операції. Якщо воно не адаптивне, будь-яка складність ламає картину.

Фактично, аналітичне ядро — це точка, де система перестає бути просто моніторингом. Воно намагається відтворити логіку роботи організації. Не

ідеалізовану, не книжкову, а справжню — з хаосом, перервами, технічними стрибками, людськими затримками. І якщо цей механізм працює правильно, він утримує цілісність даних так, як тримає форму каркас будівлі: його майже не видно, але без нього все розсиплеться.

Як тільки намагаєшся пояснити, за якими правилами має працювати аналітичне ядро, виявляється, що це не алгоритм і навіть не набір методів у прямому сенсі. Це більше схоже на своєрідну методичну угоду між різними частинами системи: технічними, поведінковими, процесними. Вони говорять різними мовами, і ядро мусить навчитися перекладати ці мови одне в одну так, щоб нічого не втратити і не перекрутити.

Починається все з моделі подій. Звучить просто, але насправді це один із найскладніших шарів. З технічними подіями ще більш-менш зрозуміло: вони структуровані, короткі, однозначні. А поведінкові? Вони завжди “недосконалі”: користувач може натиснути одну кнопку, потім передумати, повернутись, знову щось натиснути — і для ядра це виглядає як шум. Модель подій має не лише фіксувати факт, а й розуміти, що з цього — робоча дія, а що — просто рух миші чи випадковий жест. Без цього ядро захлинеться у дрібницях.

Нормалізація — наступний принцип, і він набагато глибший, ніж здається. Тут не про те, щоб «привести формати до спільного вигляду». Тут про сенс. Наприклад, одна система каже “операція завершена”, інша — “успіх”, третя — “код 200”. Формально все це одне й те саме. Але якщо поведінкові дані кажуть, що користувач тричі перезапускав дію, поки з’явився “успіх”, ядро має бачити не один успішний запит, а складну серію спроб. Бо саме вона відображає реальність. Нормалізація тут — це інтерпретація, а не форматування.

Окрема історія — зіставлення сигналів. Часто буває так, що подія на сервері відбулась у момент t, а дія користувача — у момент t-2 секунди. Деякі системи відразу втрачають контекст: «час не співпадає». Але у реальному житті ці події пов’язані. І саме ядро має зшити їх разом, навіть якщо часові мітки плавають. Тут працює принцип «розумної кореляції»: не буквально за секунду, а за змістом. Приблизно, але точно настільки, щоб не втратити причинно-наслідковість.

Потім з’являється ідея сенсових шарів. Це та сама концепція, яка дозволяє не змішувати все в одну кашу. Технічний шар — про факти. Поведінковий — про наміри та дії. Причинно-наслідковий — про логіку, яка між ними. І ядро має тримати ці шари окремо, але водночас будувати між ними зв’язки. Якщо їх змішати, отримаємо хибні висновки: або техніка винна там, де проблема у користувачі, або навпаки.

Найделікатніший принцип — робота з невизначеністю. У поведінкових даних завжди будуть розриви. У технічних — затримки. У логах — суперечності. І ядро не має права реагувати на це як на помилку. Його задача — «утримувати» складну реальність, навіть якщо вона не піддається суворій структурі. Іноді рішенням стає позначення події як "підозрілої". Інколи — побудова альтернативних інтерпретацій. А інколи потрібно просто прийняти те, що дані неповні, але на рівні системи це краще ніж вигадане узгодження.

Тут проявляється ще одна методична річ: аналітичне ядро має бути не тільки формальним, а й обережним. Воно не може автоматично зшивати події, якщо вони виглядають схожими. І водночас не може їх розривати, якщо вони не зовсім збігаються. Це тонка межа, де вирішує контекст, а не правила. І саме ця межа робить ядро не просто технічним компонентом, а логічною "опору" всієї системи.

Усе це разом створює зовсім іншу філософію. Ядро не рахує, не оцінює, не контролює. Воно збирає історію — таку, якою вона є. Усі події, усі відхилення, усі дрібні речі, які на перший погляд не мають значення, але в сумі формують цілісну картину. І ця методична послідовність дозволяє системі не просто читати дані, а розуміти їх.

Коли доходиш до практики, стає зрозуміло, що аналітичне ядро не може існувати як окрема "красива" концепція. Воно або працює разом з усіма модулями системи, або взагалі не має цінності. І тут починається зовсім інший рівень викликів — не теоретичних, а тих, що проявляються тільки в реальній роботі: коли дані приходять не рівно, події зсуваються, користувачі змінюють свої патерни, а інфраструктура поводиться не так, як у документації.

Першим завданням стає організація взаємодії між ядром і зовнішніми

компонентами. Звучить банально: API, черги, буфери. Але насправді тут ключове — не механіка, а стійкість. Якщо технічний модуль раптом дає на вихід потік подій у десять разів більший за норму (наприклад, під час аварії), ядро не може просто "захлинутися". Йому потрібно відреагувати так, щоб не втратити логіку: зупинити частину подій, тимчасово агрегувати їх, відмітити як «аномальні». Інакше аналітика розсиплеться на шматки.

Інша річ — швидкість. У нормальному моніторингу затримка на кілька секунд нічого не вирішує. Але коли ядро працює з поведінковими подіями, секунди стають важливими. Якщо система почне оновлювати моделі надто повільно, користувач отримає картину, яка на крок позаду реальності. І це викликає дивний ефект: дані ніби є, але відчуття правдивості зникає. Тому ядро має працювати досить швидко, щоб підтримувати «теплий контекст» подій — той, який ще не став історією.

Але одночасно має бути й протилежна якість — терпимість до затримок. Парадокс, але без неї система не виживе. Якщо ядро відразу викидає події, які не вписуються в таймінги, воно втратить найважливіше: здатність зберігати цілісність. Реальні дані завжди приходять хвилями: частина — миттєво, частина — із запізненням, частина — пакетами. І ядро повинно приймати все це як нормальність, а не як відхилення.

Ще один елемент — модуль логічного узгодження. Це не традиційний алгоритм, а швидше «бюро перекладів» між подіями. Коли ядро отримує сигнал, який суперечить іншому, йому доводиться вирішувати, чи ці події дійсно суперечать одна одній, чи вони просто описують різні аспекти одного процесу. Наприклад: сервер пише, що операція завершилась, а поведінкова подія каже, що користувач натиснув кнопку повторно. Це конфлікт? Ні. Це просто дві точки однієї історії. Але без модуля узгодження система робить хибні висновки.

Є ще питання адаптивності. Аналітичне ядро не може бути жорстким. Воно повинно потроху вчитись — не в сенсі «штучного інтелекту», а в сенсі розуміння того, як реально працюють процеси в конкретній компанії. Якщо одне й те саме відхилення повторюється багато разів, ядро має змінити свою модель: або розширити часові рамки, або додати новий тип зв’язку між подіями. Інакше

доведеться постійно вручну підлаштовувати правила.

Пояснюваність — ще одна вимога. У складних системах дуже легко отримати правильний результат неправильним шляхом. І якщо ядро не вміє пояснити, чому воно зшило події саме так, а не інакше, довіру до системи втратять першими не користувачі, а IT-фахівці. Тому потрібні механізми, які дозволяють побачити, як система "думала": які події вона вважала ключовими, які ігнорувала, які підозрювала в аномальності.

Окремо стоїть питання стійкості до пошкоджених даних. У реальності частина подій буде ламаною: обірвані пакети, неповні записи, дублікати, затерті фрагменти. Ядро не може просто викидати такі речі — іноді саме пошкоджені події несуть сенс (наприклад, маркер збою). Тому потрібен цілий механізм, який визначає: «ця подія неповна, але важлива», «ця — потенційно шкідлива», «ця — може бути шумом». Це майже як фільтрація у музиці: потрібно прибрати зайве, але не загубити основну мелодію.

У підсумку аналітичне ядро стає майже живим компонентом. Не в сенсі штучного інтелекту, а в сенсі того, що воно працює як система, яка не просто виконує правила, а зберігає зв’язність реальності. Воно не дозволяє технічним і поведінковим подіям розбігатися у різні боки. Воно згладжує хаос, але не приховує його. І саме завдяки цьому система перестає бути просто інструментом збору даних і починає бути інструментом розуміння.

2.6. Аналітичне обґрунтування необхідності побудови інтегрованої системи

Якщо чесно подивитися на те, як у більшості компаній збираються дані, складається враження, що кожен елемент живе у власній реальності. Технічні системи щось собі раху́ють, фіксують навантаження, помилки, плюсують метрики, і виглядає це дуже впорядковано. Люди при цьому працюють у своєму ритмі: зупинився, передумав, відклав, повернувся, ще раз натиснув… Ніби нічого складного, але коли порівнюєш ці два світи, вони майже не стикуються між собою. А ще окремо стоять бізнес-процеси, які проходять крізь техніку та

поведінку, але дуже часто не співпадають ні з одними, ні з іншими.

Через це виникає дивне явище: дані є, але вони ніби про різні історії. Наприклад, сервери показують "усе зелено", а команда на практиці працює повільніше, ніж зазвичай. І ніхто не може пояснити чому. Або протилежна ситуація: користувачі повторюють одну й ту саму дію кілька разів поспіль, а технічні логи наполягають, що "все добре". Часом здається, що ти дивишся два різні фільми, які випадково перемішались у одному плеєрі.

Інколи такі розриви виглядають навіть трохи абсурдно. Наприклад, поведінкові події сигналізують: людина намагається завершити операцію вже втретє. А сервер у той самий момент демонструє стабільність — мовляв, запит оброблено, мережа чиста, затримок немає. І виходить, що техніка стверджує одне, користувачі відчувають інше, а процеси показують зовсім третій результат. У цьому всьому наче немає конфлікту, просто кожен шар говорить своїми словами, і жоден із них нічого не знає про інші.

Через таку відірваність аналітика стає дуже хиткою. Люди малюють графіки, рахують середні значення, шукають пік активності або зависання, але висновки з цього — половинчасті. Наче читаєш роман, але тобі дали лише кожну п’яту сторінку. Можна щось зрозуміти, але сенс постійно вислизає.

Бувають ще смішніші ситуації. Наприклад, система фіксує, що користувач нічого не робить хвилину чи дві. І технічний шар показує тишу — усе стабільно, жодних подій. А насправді людина просто чекає, поки інтерфейс підтягне дані з іншого сервісу. І виходить, що проста затримка у віддаленій системі перетворюється на "падіння продуктивності персоналу", хоча це абсурд. Або інша крайність: сервер ледь дихає від навантаження, а поведінкові дані виглядають ідеально рівними — працівники припинили будь-які дії, бо система не відповідає. І якщо дивитись на поведінку без техніки, здається, що всі просто перестали працювати.

Ці розриви накопичуються і створюють таку собі "аналітичну зону туману". Formально все зрозуміло, але коли потрібно з’єднати подію з її причиною — ланцюга немає. Техніка оцінює себе. Люди оцінюють себе. Процеси оцінюються ще якось. І в підсумку керівник бачить фрагменти, але не

історію.

Ось чому аналітика інколи дає помилкові висновки. Наприклад:

– технічний моніторинг підказує, що система працює чудово;

– поведінкові дані — що люди перевантажені;

– процесні показники — що час виконання задачі росте.

І кожен шар, якщо дивитись на нього ізольовано, правий. А якщо разом — картинка розвалюється, бо між шарами немає зв’язків.

Це і створює ту саму проблему: система бачить події, але не розуміє їхню історію. А без історії будь-яка подія виглядає як випадковість. І поки технічні, поведінкові та бізнесові дані не з’єднані в один контекст, неможливо зрозуміти, чому процес повільний, чому співробітник щось повторює, чому сервер вважає, що все гаразд, хоча користувач нервує.

У такий момент інтегрована система перестає бути «опцією» і стає просто необхідністю. Не як модний тренд, а як спосіб перестати дивитися на роботу компанії через маленькі вікна.

Коли намагаєшся розібратися, чому взагалі потрібна інтегрована система, спочатку здається, що відповідь очевидна: дані розрізнені, треба зібрати їх разом. Але це надто поверхнево. Насправді інтеграція потрібна не через те, що дані лежать у різних місцях, а через те, що без інтеграції неможливо зрозуміти, що насправді сталося. Технічний шар бачить одне, люди — інше, процеси — третє. І коли між ними немає перехідних містків, будь-який аналітичний висновок виходить майже інстинктивним, а не логічним.

З логічного боку все навіть простіше, ніж здається. Система не може описувати себе частинами. Процес складається з технічних дій, поведінкових реакцій і результату. Це один ланцюг, але класичні інструменти розглядають його ніби окремими шматками. І якщо аналізувати лише один шар, ти отримуєш тільки історію «зсередини» цього шару. Тобто аналіз виходить частковим за визначенням. Звідси — маса помилкових висновків, коли проблема не там, де її намагаються шукати.

Технічний шар теж не витримує цих суперечностей. Системи стали настільки розподіленими, що події з’являються в різні моменти часу, інколи

взагалі поза синхронізацією. Іноді лог пише про подію через секунду після того, як вона реально відбулась. А поведінковий модуль фіксує дію миттєво. Виходить парадокс: події, які пов’язані між собою, у даних опиняються далеко одна від одної. І тут інтеграція вже не про "з’єднати таблиці", а про "зрозуміти реальний порядок подій", що без спеціальної логіки просто неможливо.

Ще одна технічна причина — різні "ритми" даних. Метрики течуть рівно, логи — ривками, поведінкові сигнали — як доведеться. Усе це неможливо покласти в одну модель без спеціального механізму, який би вмів «згладжувати» хаос. Старі підходи намагаються робити це через правила: якщо А сталося після В, значить вони пов’язані. Але в реальних системах це майже ніколи не працює. Події пов’язані змістом, а не часом. І саме це змушує створювати нову модель — не схему, а логічний рівень, який підтримує взаємозалежність.

Організаційні аргументи не менш вагомі. Керівництво, як правило, живе в реальності звітів: «ефективність упала», «навантаження зросло», «процеси сповільнились». Але якщо немає нормальної інтеграції, джерело проблеми визначається буквально "на око". Хтось каже: «це техніка», інший — «це користувачі», третій — «це процеси». І кожен начебто має аргументи, бо базується на своєму шарі реальності. Але істина лежить посередині, у тому місці, де технічна подія вплинула на поведінку, а поведінка — на процес.

І без інтеграції цього місця не видно взагалі., тому через це виникає організаційний перекіс: компанія шукає рішення там, де проблема тільки відображається, але не виникає. Інколи це призводить до абсолютно неправильних рішень. Наприклад, замість того щоб виправити затримку в одному мікросервісі, бізнес починає "оптимізувати" персонал, хоча люди просто стояли в очікуванні відповіді системи. Або навпаки — змінюють процес розв’язання задач, хоча реальна причина була у поведінковому факторі (неправильна послідовність дій).

Інтеграція тут потрібна не як красива теорія, а як спосіб уникнути цих неправильних рішень. Бо будь-яка система, яка дивиться тільки на один шар, неминуче викривлює картину. Висновки стають "кусочними", але бізнесу потрібна цілість.

Ще один аргумент — довіра. Якщо дані суперечать одне одному, користувачі перестають довіряти системі. ІТ — технічним метрикам, керівництво — поведінковим, аналітики — процесним. Виходить не система, а набір сигналів, які всі читають по-своєму. Інтегрована модель, навпаки, створює точку узгодження. Можна не погоджуватись із конкретним висновком, але сама логіка стає зрозумілою: можна прослідкувати, як подія вплинула на людину, а та — на процес.

У результаті логічні, технічні та організаційні причини сходяться в одній точці: без інтеграції система майже завжди бреше, навіть якщо всі дані в ній правдиві.

Коли нарешті збираєш усі ці дані докупи і система починає працювати як єдине ціле, виникає трохи дивне відчуття: ніби нічого особливого не змінилося, але раптом стає видно те, що раніше вислизало. Події починають складатися в історії, а історії — в закономірності. І саме тут виявляється, навіщо вся ця інтеграція була потрібна. Не для ефектності, і точно не заради модного слова "аналітика". Вона потрібна, щоб нарешті з'явився контекст, якого завжди не вистачає.

Найперше, що помічається, — це те, що система перестає давати суперечливі сигнали. Якщо раніше технічний шар міг говорити одне, а поведінковий — інше, то інтегроване ядро робить те, чого не робили інші частини: воно пояснює, чому все так. І виявляється, що більшість «дивних» інцидентів зовсім не дивні, просто вони раніше були розірвані на два несумісні шматки. Коли їх нарешті зводять разом, ситуація стає логічною, навіть якщо вона неприємна.

Другий ефект — різке зниження шуму. Поведінкові дані самі по собі — суцільний рух. Технічні — хаотичні сплески. І лише коли вони потрапляють у спільний контур, стає зрозуміло, які події важливі, а які — фоновий шум. Система починає бачити не просто "робив дію" або "метрика стрибнула", а ланцюг: користувач зробив це, бо техніка відповіла так, і це вплинуло на процес ось у такий спосіб. Цей ефект зшивання подій настільки змінює аналітику, що інколи здається, ніби система «порозумнішала», хоча насправді вона просто

перестала втрачати напівправду.

Третя річ — точність. У традиційних рішеннях аналітика часто працює "навмання": підрахунок середніх значень, графіки за днями, великі категорії. В інтегрованій системі це змінюється. Через те, що всі шари даних нарешті сполучені, система бачить причинно-наслідкові зв'язки. Не тільки "що сталося", а "чому саме так". І це не просто зручність. Це повністю змінює спосіб прийняття рішень. Можна не здогадуватися, а знати.

А потім з’являється можливість, якої без інтеграції просто не існувало — прогнозування. Якщо система бачить реальні ланцюги подій, а не окремі елементи, вона починає помічати, які патерни повторюються. Наприклад, що конкретна затримка на сервері через десять хвилин призводить до зменшення активності користувачів. Або що певна поведінкова серія завжди закінчується технічним збоєм. Це виглядає майже магічно, але це просто нормальний наслідок того, що дані більше не розірвані.

Ще одна перевага — прозорість. Не в сенсі «всі всіх контролюють», а в сенсі, що кожна подія бачить свою причину. IT перестає звинувачувати персонал, персонал — інфраструктуру, керівництво — процеси. Коли все в одному просторі, всі бачать однакову картину, і дискусія нарешті переходить з рівня припущень у рівень фактів.

Ну і стратегічна складова теж з’являється. Інтегрована система вже не просто вимірює, вона дозволяє зрозуміти, як компанія працює насправді: де вузькі місця, де люди витрачають час, де техніка тягне вниз, а де процеси збивають темп. Це не контроль у сенсі "хтось за кимось слідкує". Це спосіб побачити реальну структуру роботи — таку, яку не можна побудувати вручну або вигадати за допомогою окремих метрик.

І найважливіше: інтегрована система перестає бути просто інструментом збору даних. Вона стає інструментом розуміння. Можна знати, що відбувається. Але значно цінніше — розуміти, чому це відбувається. І тільки система, яка тримає в одній точці технічні, поведінкові та процесні події, здатна давати такі відповіді.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ ТА КОНТРОЛЮ ПРОДУКТИВНОСТІ

Система, яку довелося розробляти для цієї роботи, виросла з дуже практичної потреби: мати під руками інструмент, який реально показує, що відбувається в мережі протягом дня, хто працює за робочою станцією, а хто зникає з мережі так само раптово, як і з’являється. У багатьох організацій подібні речі залишаються на рівні здогадів або коротких коментарів "щось у нього інтернет падає". Тут же хотілося отримати картину без припущень, у якомусь більш предметному вигляді. Так і сформувалась ідея невеликого комплексу, який не претендує на масштабну платформу, але вирішує дуже конкретні задачі.

Основою всієї логіки став регулярний збір подій про стан робочих машин. Не потрібно нічого встановлювати на стороні користувача, ніяких агентів чи прихованих трекерів – достатньо зрозуміти, коли пристрій відповідає, а коли ні. Звучить просто, але на практиці це дає багато інформації: видно тенденції, видно провали, видно "мертві години", коли комп’ютер стоїть офлайн, хоча мав би працювати. І, що цікаво, часом справді можна простежити робочі звички окремих співробітників: у кого все стабільно, а у кого постійні відключення чи дивні "порожні проміжки".

Щоб ці події не тонули в інформаційному шумі, було вирішено зберігати їх у вигляді звичайного журналу. Простий формат, який легко переглянути і вручну, і через pandas, і згодом перетворити в акуратний звіт. Саме звіти, до речі, дозволили побачити, що навіть базові дані про "онлайн/офлайн" можна обробити так, що виходить цілком зрозуміла модель робочого часу за різні періоди. Хтось проводить у мережі шість годин підряд, хтось працює ривками, а хтось має підозріло багато відключень – і це вже не припущення, а цифри.

Інтерфейс вирішили зробити максимально простим – Telegram дав тут більше, ніж очікувалось. Бот сповіщає про зміни статусів, приймає команди, дозволяє згенерувати звіт і нічого не вимагає від користувача, окрім доступу до месенджера. У підсумку виходить така собі контрольна панель, тільки без панелі: усі важливі сигнали прилітають у звичайний чат. Керівнику навіть комп’ютер не

завжди потрібен – звіт приходить у телефон, і його можна відкрити буквально на ходу.

Загалом весь механізм працює як одна зв’язка: опитування вузлів, журнал подій, обробка накопичених даних і рольовий доступ, щоб розділити можливості адміністратора й керівника. Від цього система виглядає не складною, а швидше компактною, але з чітким набором можливостей. Вона дозволяє бачити технічний стан мережі, оцінювати активність співробітників і формувати наочну звітність, яка дійсно допомагає зрозуміти, що відбувається у внутрішній інфраструктурі протягом дня, тижня чи довшого періоду.

3.1. Постановка задачі дослідження та формалізація функціональних вимог

Завдання, яке стояло перед системою, поступово сформувалось із доволі приземленої потреби: потрібен був інструмент, який дає можливість швидко зрозуміти, що відбувається з робочими станціями в мережі та наскільки стабільно вони працюють протягом дня. Звучить майже буденно, але саме ця буденність і створює найбільше проблем. Якщо робоча машина "падає" кілька разів на день, це зазвичай ніхто не фіксує. Якщо співробітник регулярно зникає з мережі, поки його комп’ютер раптово перестає відповідати, це бачать лише постфактум, коли щось уже не виконано.

У такій ситуації задачою стає не просто моніторинг доступності, а створення механізму, який поєднує дані про інфраструктуру й дані про активність персоналу в одній картині. Виходило, що потрібно одночасно відстежувати технічний стан вузлів, розуміти часові проміжки між подіями, зберігати їх у вигляді хронології та, головне, робити це так, щоб підсумкова інформація була придатною для аналізу. Тобто не лише відповідь "онлайн/офлайн", а щось значно повніше: коли саме відбулася зміна, як довго тривав попередній стан, на якому робочому місці це сталося і до якого співробітника це належить.

```
connection = "📶 Wi-Fi" if mac in wifi_macs else "🖧 Кабель"
```

Потрібно було визначитись із тим, якими мають бути функціональні можливості системи. По-перше, регулярний збір подій. Безперервне опитування вузлів мало давати потік статусів, але не хаотичний, а впорядкований за логікою зміни станів. Саме тому в основу проєкту закладено фіксацію переходів, а не кожної окремої відповіді. Такий підхід дозволяє уникнути шуму та зосередитись на суттєвому – на моменті, коли коливання статусу технічно відображає зміну поведінки користувача або проблеми з обладнанням.

```
if prev_status is None or prev_status != online:
    log_to_csv(host, online, connection)
```

По-друге, збереження всієї історії подій у зручній формі. Не в тимчасовій змінній, яка зникне після перезавантаження, а в журналі, що накопичує дані день за днем. Тут зручним виявився CSV – простий формат, але достатній, щоб з нього витягувати потрібні патерни та будувати динаміку.

Фрагмент реалізації базового журналу подій у коді виглядає приблизно так:

```
def log_to_csv(host, online, connection):
    with open(CSV_PATH, "a", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow([
            datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            host["ip"], host["mac"], host["name"], host["room"],
            "ONLINE" if online else "OFFLINE",
            connection
        ])
```

Кожен запис тут – це маленький "зріз" події: час, IP, MAC, конкретний працівник, його локація і новий стан. Хронологія таких зрізів поступово перетворюється у щось значно вагоміше за суму частин.

Третя вимога виросла з того, що доступ до аналітики не може бути однаковим для всіх. Адміністратору важливо бачити інциденти – падіння й відновлення. Керівнику – узагальнення: скільки часу співробітники провели

онлайн, чи були у них довгі перерви, як виглядала ситуація за тиждень, а не лише за останні три хвилини. Тому необхідно було закласти ролевий доступ, розмежовуючи функції.

У коді це вирішено без зайвих ускладнень, але достатньо ефективно:

```
PASSWORDS = {
   "boss": "1811",
   "admin": "1234"
}
ROLES = {
   "Керівник": "boss",
   "Адміністратор": "admin"
}
```

Це дозволило надалі розділити інтерфейс: адміністратор запускає моніторинг, дивиться події, але не може формувати звіти; керівник, навпаки, отримує доступ до генерації узагальненої статистики.

Четверта функціональна вимога стосувалась формування зрозумілої аналітики. Ідея полягала в тому, щоб на основі накопиченої історії можна було розрахувати фактичний час перебування у мережі. Не теоретичний, а той, що об’єктивно випливає з логів: коли ONLINE змінився на OFFLINE, тоді й закінчився попередній період активності. Ця модель проста, але вона точніше відображає робочий ритм, ніж будь-які ручні записи.

```
if status == "ONLINE":
   start_time = timestamp
else:
   online_duration += timestamp - start_time
```

Важливо було ще одне: надати цю аналітику у такій формі, яка зрозуміла людині без технічного досвіду. Excel виявився найпростішим рішенням. Вивантаження трьох аркушів – повний журнал, тільки події, та підсумкові дані – закривало повністю вимогу щодо звітності.

І нарешті, потрібно було передбачити механізм взаємодії, який не ускладнює роботу користувача. Telegram підійшов ідеально: мінімум дій, швидке сповіщення, природний інтерфейс. Команди накладаються на роль користувача, а сама система працює фактично як канал комунікації між мережею та керівництвом.

У сумі всі ці вимоги й сформували ту модель, яка реалізована у проєкті. Вона не прагнула до надмірної складності, а навпаки – до практичності. Зібрати процеси моніторингу, журналювання, аналітики й керування доступом так, щоб вони працювали разом, давали зрозумілий результат і не вимагали від користувача нічого зайвого.

3.2. Архітектура програмного комплексу та логічна структура компонентів

Архітектура всієї системи формувалась не як класична багаторівнева схема, що малюється наперед, а як набір практичних рішень, які поступово складалися в цілісний механізм. У центрі всього – список вузлів, або, якщо дивитися ширше, перелік робочих станцій, що прив’язані до конкретних людей. Саме з цього списку починається вся робота комплексу. Кожен запис містить не лише IP чи MAC, а й інформацію, яка робить події зрозумілими і "людськими" для керівника. Наприклад, структура одного вузла виглядає так:

```
{
  "ip": "192.168.88.253",
  "mac": "E8:6A:64:B6:3E:75",
  "name": "Петренко Петро Васильович",
  "room": "каб. 407, поверх 4"
}
```

Така форма зберігання даних дозволяє системі працювати не з абстрактними адресами, а з конкретними профілями співробітників, і це сильно

змінює характер усієї взаємодії: коли надходить сповіщення про зміну статусу, користувач отримує не технічне повідомлення, а подію, що стосується конкретної людини, конкретного місця роботи. Це не просто для зручності – так формується контекст, у якому керівник може швидко інтерпретувати технічну інформацію.

Другий важливий компонент архітектури – Telegram-бот, який виступає єдиним інтерфейсом користувача із системою. Бот не тільки показує результати, а й керує доступом. Для цього реалізована проста FSM-схема, де користувач проходить через кілька станів: вибір ролі, введення пароля, доступ до меню команд. Код, що керує цим циклом, побудований через `ConversationHandler`:

```
conv_handler = ConversationHandler(
    entry_points=[CommandHandler('start', start)],
    states={
        STATE_ROLE: [MessageHandler(Filters.text & ~Filters.command, role_selected)],
        STATE_PASSWORD: [MessageHandler(Filters.text & ~Filters.command, password_entered)],
        STATE_MENU: [MessageHandler(Filters.text & ~Filters.command, menu_handler)],
    },
    fallbacks=[CommandHandler('start', start)]
)
```

Жодних баз даних для користувачів тут немає – достатньо словника `user_sessions`, де для кожного `chat_id` зберігається роль та індикатор активного моніторингу. У такий спосіб система не ускладнюється зайвими шарами, але при цьому забезпечує чітке розмежування можливостей. Ролі також визначені у вигляді словників:

```
PASSWORDS = {"boss": "1811", "admin": "1234"}
```

```
ROLES = {"Керівник": "boss", "Адміністратор": "admin"}
```

Це дає змогу обмежити доступ до звітів, залишивши їх лише тим, хто має керівні повноваження.

Основна логіка моніторингу винесена в окремий фоновий потік, який запускається щоразу, коли користувач натискає "Старт моніторингу". Потік працює паралельно з ботом і не блокує його взаємодію з користувачем. Старт виглядає досить компактно:

```
thread = threading.Thread(
    target=monitoring_thread,
    args=(context.bot, chat_id, hosts),
    daemon=True
)
thread.start()
```

Усередині цього потоку відбувається опитування вузлів методом ICMP. Для кожного IP система викликає:

```
resp = ping(ip, count=2, timeout=1)
online = resp.success()
```

Тут же підтримується кілька внутрішніх структур, які дозволяють системі розуміти не лише поточний стан, а й те, коли він змінився та як довго тривав попередній період. Зберігаються значення `prev_status`, час останньої зміни `last_change`, а також момент початку онлайн- чи офлайн-інтервалів (`online_start_time`, `offline_start_time`). Завдяки цим даним система може формувати інформативні повідомлення, у яких присутній не лише статус, а й тривалість попереднього періоду роботи. Механізм виглядає приблизно так: якщо пристрій був ONLINE, але став OFFLINE, обчислюється різниця часу, і в Telegram відправляється повідомлення з прикріпленою тривалістю. Запис у журнал створюється одразу в момент зміни:

```
log_to_csv(h, online, connection)
```

Окрема частина архітектури – файл журналу. Уся історія подій зберігається в одному CSV-файлі, який автоматично створюється на робочому столі користувача, якщо його раніше не існувало:

```
DESKTOP = os.path.join(os.path.expanduser("~"), 'Desktop')
CSV_PATH = os.path.join(DESKTOP, "monitor_log.csv")
```

Структура CSV визначена функцією `init_csv`, де задаються заголовки колонок. Кожен запис має однакову форму:

```
writer.writerow([
    datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    h["ip"], h["mac"], h["name"], h["room"],
    "ONLINE" if online else "OFFLINE",
    connection
])
```

Таким чином формується історія подій, яка потім служить матеріалом для аналітичної обробки. З архітектурної точки зору це рішення дуже прямолінійне, але саме в його простоті полягає сила: жодного зайвого шару, жодних залежностей, можливість читати файл у будь-якому середовищі й у будь-який момент.

Коли керівник запитує звіт, запускається аналітичний модуль, який читає CSV через pandas:

```
df = pd.read_csv(CSV_PATH)
df["datetime"] = pd.to_datetime(df["datetime"])
df = df.sort_values(by="datetime")
```

Подальша логіка ґрунтується на послідовному проходженні подій того самого IP і обчисленні різниці між ними. Якщо попередній статус був ONLINE,

а наступна подія припадає на момент OFFLINE, система додає тривалість, яка минула між цими двома точками. На цій основі створюється підсумкова таблиця з активністю за кожен період – день, тиждень, місяць, увесь час роботи. Усі результати збираються в Excel-файл, який містить три аркуші: повний журнал, події зі змінами станів та зведені показники продуктивності.

Фінальний етап – форматування Excel-документа. Тут використовується openpyxl, що дозволяє зробити документ набагато читабельнішим: колонки автоматично підлаштовуються, шапки таблиць виділяються кольором, рядки зі змінами статусу підсвічуються. Логіка застосування стилів виглядає приблизно так:

```
if cell.value == "ONLINE":
    cell.fill = green_fill
    cell.font = Font(bold=True, color="006100")
elif cell.value == "OFFLINE":
    cell.fill = red_fill
    cell.font = Font(bold=True, color="9C0006")
```

Це надає звіту завершеності та робить його зручним у роботі. Керівник отримує документ, у якому "видно все": і коли пристрій працював, і коли падав, і які періоди займала кожна подія.

Якщо спробувати поглянути на всю архітектуру цілісно, то вона складається з кількох взаємопов’язаних шарів: модель вузлів, интерфейс Telegram-бота, фоновий моніторинговий потік, підсистема журналювання та аналітичний блок зі звітністю. Кожен шар працює автономно, але результат з’являється лише тоді, коли вони взаємодіють разом. Власне ця взаємодія й визначає характер системи – вона не перевантажена складними технологіями, але дає чітку, структуровану та керовану модель активності користувачів і роботи інфраструктури.

3.3. Реалізація серверної частини: збір подій, база даних, API, модуль ролевого доступу

Серверна частина системи стала тим місцем, де вся логіка моніторингу, обробки подій і взаємодії з користувачем фактично зводиться докупи. Вона працює як такий собі "двигун", який не видно ззовні, але від нього залежить усе – і коректність фіксації статусів, і точність побудови часових інтервалів, і те, що користувач отримає у вигляді звіту. Щоб досягти цього, довелося поєднати кілька різних підходів у межах одного процесу, причому так, щоб вони не заважали один одному. Логіка Telegram, моніторинг ICMP, журналювання, аналітика і модуль доступу — все це взаємодіє паралельно і потребує тонкого узгодження.

Першим кроком при запуску програми йде ініціалізація середовища та створення журналу, якщо його ще немає:

```
init_csv()
```

Ця функція перевіряє існування CSV-файлу, і якщо файл відсутній, створює шапку. Архітектурно це маленька деталь, але вона знімає необхідність у складних перевірках у момент запису: будь-який потік або виклик може просто звернутися до `log_to_csv`, не турбуючись про структуру файлу. Так створюється відчуття, що база даних завжди готова до запису.

Далі система піднімає Telegram API:

```
updater = Updater(BOT_TOKEN, use_context=True)
dp = updater.dispatcher
```

Класична для Python Telegram Bot API конструкція, але тут вона має особливе значення: `Dispatcher` фактично бере на себе функцію API-контролера для зовнішнього доступу. Саме він приймає всі повідомлення користувачів і передає їх у відповідні функції, які відтворюють поведінку

сервера. Взаємодія тут одностороння: ініціатива завжди йде від Telegram, і саме це дозволяє уникнути зайвих ресурсних циклів у головному потоці.

Особливе місце займає FSM — механізм станів, через який проходить кожен користувач. Це своєрідна людино-машинна угода: поки користувач не визначився з роллю і не підтвердив пароль, сервер не запускає для нього критичних процесів. Такий підхід знімає потребу в окремій системі аутентифікації чи збереженні паролів у базі даних — вони просто порівнюються з локальним словником:

```
PASSWORDS = {"boss": "1811", "admin": "1234"}
```

І хоча така схема виглядає дуже простою, вона відмінно працює в контексті Telegram, де один користувач — це однозначний `chat_id`. Завдяки цьому вся логіка контролю доступу зводиться до того, чи є в `user_sessions` запис із правильними атрибутами. Цей словник, по суті, виконує роль оперативної бази даних сесій:

```
user_sessions[chat_id] = {"role": role, "monitor": False}
```

Збереження лише матеріально необхідних даних дозволило уникнути надмірності та зайвих перевірок.

Там, де починається моніторинг, починається і справжня робота сервера. Елементарний виклик:

```
"Старт моніторингу"
```

перетворюється на запуск фонової задачі, яка живе власним життям і не залежить від щільності взаємодії користувача з ботом. Це реалізовано через стандартну модель потоків:

```
thread = threading.Thread(
    target=monitoring_thread,
    args=(context.bot, chat_id, hosts),
    daemon=True
```

```
)
thread.start()
```

Потік працює в режимі daemon — тобто він завершується автоматично, коли закінчує роботу основний процес. Такий режим обрано для того, щоб не створювати ситуацій, де код висить у пам’яті після закриття програми.

Сама функція `monitoring_thread` працює циклічно: доки для поточного `chat_id` встановлено `"monitor": True`, вона постійно опитує всі вузли. Це дає можливість одночасно запускати кілька потоків для різних користувачів, якщо є така потреба, — система не обмежує кількість активних сесій.

Перевірка доступності виконується через бібліотеку `pythonping`:

```
resp = ping(ip, count=2, timeout=1)
online = resp.success()
```

Тут є важливий нюанс: система записує тільки зміну станів. Якщо вузол відповідає нормально, і так було раніше — нічого не відбувається. Такий підхід дозволяє уникати “засмічення” журналу однотипними рядками, які не несуть нової інформації. Замість цього фіксуються тільки ключові події: перехід в offline та повернення в online.

Для цього весь час підтримуються чотири словники:

```
prev_status = {}
online_start_time = {}
```

Сервер фактично веде мікро-базу поточних станів для кожного IP. Завдяки цьому можливо зрозуміти, що саме змінилося, і сформувати повідомлення з конкретними даними. Система може сказати не лише “вузол зник”, а й “він був у мережі 3 год 18 хв 22 сек перед тим, як зник із мережі”.

Повідомлення формується через спеціальну функцію:

```
msg = format_node_message(h, online, duration)
```

А сама логіка форматування виглядає так:

```
f"👤 Користувач: {h['name']}\n"
f"🔔 Статус: {'ONLINE' if online else 'OFFLINE'}\n"
f"⏱️ Тривалість попереднього стану: *{duration}*\n"
```

Це робить події не просто технічними, а зрозумілими навіть для людей без IT-фону.

Після надсилання повідомлення в Telegram подія записується в CSV:

```
log_to_csv(h, online, connection)
```

Сам CSV-файл у цій системі — це більше, ніж просто таблиця. Він слугує повноцінною базою даних, яка зберігає всю історію змін і стане основою для аналітики. Його структура була обрана не випадково: дані мають бути легкі для обробки, переносимі, незалежні від середовища виконання і прості у відтворенні вручну. CSV відповідає всім цим критеріям.

Інша частина серверної логіки — це аналітичний модуль, який працює виключно тоді, коли керівник запитує звіт. Так система не витрачає ресурси без потреби. Модуль читає CSV у pandas, розраховує тривалості, визначає точки переходів і формує Excel-документ із кількома аркушами. Усе це відбувається всередині одного виклику:

```
generate_and_send_excel(update, context)
```

Тут же формується й API-відповідь: документ надсилається на робочу станцію або смартфон керівника. Немає окремого сервера API — Telegram виконує роль транспортного шару.

Завершуючи опис серверної частини, можна сказати, що вся вона працює за принципом тісної інтеграції простих елементів: локальних словників, CSV, потоків, ping-запитів і FSM в Telegram. І хоча кожен елемент окремо здається не надто складним, у сумі вони формують логічно завершену систему, яка

безперервно приймає рішення, накопичує події, аналізує дані й надає інформацію в максимально зручній формі.

3.4. Модуль моніторингу IT-інфраструктури: механізми фіксації інцидентів та подій

Модуль моніторингу в цій системі став тим елементом, який задає ритм усій роботі: саме він створює безперервний потік даних про стан мережевих вузлів, із якого потім формується історія інцидентів та подій. Логіка його роботи побудована доволі прагматично: якщо робоча станція відповідає — це один стан, якщо перестає відповідати — це інший стан, і саме момент переходу між ними має найбільшу інформативну вагу. Такий підхід дозволяє не зберігати зайву інформацію, а фіксувати лише суттєві зміни, які впливають на доступність робочого місця та продуктивність працівника.

Коли користувач запускає моніторинг, сервер створює фоновий потік, у якому й відбувається основна робота. Цей потік постійно проходить по масиву `hosts`, де описано всі робочі станції, і виконує для кожної з них два послідовних запити. У коді це виглядає як нескінченний цикл, який працює до тих пір, поки в об’єкті сесії встановлений прапорець `"monitor": True`:

```
while monitoring_enabled:
    for h in hosts:
        ip = h["ip"]
        resp = ping(ip, count=2, timeout=1)
        online = resp.success()
```

Зовні ця конструкція здається дуже простою, але всередині вона створює безперервний потік сигналів, із яких модуль намагається зрозуміти не просто “стан зараз”, а поведінку вузла в динаміці. Для цього сервер веде у пам’яті кілька словників: `prev_status`, `last_change`, `online_start_time`, `offline_start_time`. У цих структурах зберігається попередній стан, час

останньої зміни і момент початку відповідного інтервалу. Завдяки цьому система знає, скільки часу пристрій перебував у мережі перед тим, як зник, або навпаки — як довго був недоступним перед поверненням.

Сам механізм переходу починається з порівняння поточного стану з тим, який був зафіксований раніше. Якщо різниці немає, модуль рухається далі. Але коли стан відрізняється, це означає інцидент, причому незалежно від того, чи це падіння, чи повернення. У момент переходу система визначає тривалість попереднього стану. Наприклад, коли вузол був ONLINE, а тепер став OFFLINE, обчислюється час, протягом якого він перебував у мережі:

```
online_time = (now - online_start_time.get(ip)).total_seconds()
```

Як тільки тривалість обчислена, формується повідомлення у форматі, який містить не лише сухий технічний статус, а й контекст, достатній для управлінського рішення. У цьому повідомленні є IP, MAC, прізвище співробітника, посада, локація, новий статус і тривалість попереднього періоду. Це робиться через функцію `format_node_message`, яка будує структурований текст:

```
msg = format_node_message(h, online, online_time)
```

Після формування повідомлення система одразу надсилає його в Telegram:

```
bot.send_message(chat_id=chat_id, text=msg, parse_mode="Markdown")
```

Це важливий момент, оскільки керівник або адміністратор бачить зміни практично миттєво, і рішення можна приймати в реальному часі. Такі сповіщення працюють як система оперативного реагування на збій, без потреби заходити в спеціальні панелі керування чи відслідковувати графіки.

Одразу після надсилання сповіщення відбувається фіксація події у CSV-файлі — саме цей файл виступає хронологічним сховищем усіх інцидентів. Запис здійснюється функцією `log_to_csv`, яка формує повноцінний рядок із датою, IP, MAC, прізвищем, посадою, локацією й статусом:

```
writer.writerow([
    datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    h["ip"], h["mac"], h["name"], h["room"],
    "ONLINE" if online else "OFFLINE",
    connection
])
```

У результаті кожен перехід стає окремою подією з повноцінним контекстом, що дозволяє згодом аналізувати поведінку вузлів у будь-який період. Сам CSV, попри простоту формату, перетворюється на чітко структурований журнал інцидентів.

У зворотному сценарії — коли вузол був OFFLINE, а потім став ONLINE — механізм працює абсолютно дзеркально. Система визначає тривалість періоду недоступності:

```
offline_time = (now - online_start_time.get(ip)).total_seconds()
```

і формує нове повідомлення з деталізацією "Час у офлайні". Так само як у випадку падіння, подія відправляється користувачу і записується в лог. Завдяки цьому повна історія доступності робочої станції фіксується у вигляді пари чергувань ONLINE/OFFLINE з точними часовими проміжками між ними.

Окремо варто сказати про причину, чому система фіксує тільки переходи, а не кожний ping. У типовому офісному середовищі ping може виконуватися сотні або тисячі разів за день на кожен вузол. Запис кожної відповіді створив би інформаційний шум і зробив би аналітику надмірно ускладненою. Натомість фіксація лише переходів дозволяє зберегти ідеальну чистоту журналу: кожний рядок — це подія, а не “пульсація” мережевого трафіку. Аналітика у такому випадку стає значно точнішою і не потребує додаткових фільтрів чи агрегування.

Тривалості періодів у системі мають ключову роль, тому для їх коректного подання використовується спеціальна функція форматування, яка перетворює секунди в години, хвилини або дні у зручному форматі. Наприклад:

```
def format_duration(sec):
```

```
if sec < 60:
    return f"{sec} сек"
elif sec < 3600:
    m = sec // 60
    s = sec % 60
    return f"{m} хв {s} сек"
```

Це робить повідомлення інтуїтивними, а звіт — зручним для читання.

Уся логіка модуля моніторингу таким чином створює плавну картину життя мережевих вузлів: кожна зміна фіксується, кожен перехід має часову прив’язку, кожний період доступності чи недоступності має чіткий початок і кінець. І саме ця повнота інформації дозволяє системі формувати достовірну статистику для управлінських рішень, а також будувати детальні звіти, у яких час онлайн і офлайн кожного співробітника поданий математично коректно й візуально зрозуміло.

3.5. Модуль контролю продуктивності персоналу: логіка обчислення показників і формування звітності

У модулі контролю продуктивності зібрані алгоритми, які фактично дозволяють перетворити окремі події ONLINE/OFFLINE у структурований опис того, як співробітники поводяться в мережі протягом дня, тижня чи довшого періоду. Хоча система не вимірює активність на рівні натискань клавіатури чи робочих файлів, вона оперує тим, що найчастіше є найбільш достовірним індикатором присутності — стабільністю підключення робочого місця. Якщо комп’ютер працівника доступний у мережі, система трактує це як робочий період, і на цьому побудована вся подальша аналітика.

Коли користувач із роллю керівника запускає формування звіту, модуль спершу завантажує CSV-файл, у якому міститься повна хронологія змін статусів. На цьому етапі важливо не просто прочитати файл, а привести його до формату,

у якому з ним можна працювати без втрати контексту. Саме тому перше, що робиться, — це очищення назв колонок, перейменування їх у більш зручний варіант і переведення часу у формат `datetime`. Це виглядає так:

```
df = pd.read_csv(CSV_PATH)
df.columns = [c.strip() for c in df.columns]
df = df.rename(columns={
    "datetime": "Дата та час",
    "ip": "IP",
    "mac": "MAC",
    "name": "ПІБ",
    "room": "Локація",
    "status": "Статус"
})
df["Дата та час"] = pd.to_datetime(df["Дата та час"])
df = df.sort_values(by="Дата та час")
```

На цьому етапі CSV перестає бути просто текстом і починає виглядати як акуратно розмічений журнал подій. Далі модуль формує "подієвий" набір даних — таблицю, у якій відображено суттєві переходи. Щоб це зробити, потрібно переглянути журнал у зворотному порядку (від найновішого до найстарішого), поступово обчислюючи, скільки саме тривав попередній стан. Тут логіка досить цікава: модуль дивиться на пару сусідніх записів тієї самої машини і визначає, чи було це завершення онлайн-періоду або завершення офлайн-періоду. Саме так з’являються текстові деталі, які пізніше потраплять у звіт.

У коді цей процес реалізовано через два словники — `prev_status` і `prev_time`, у яких зберігається інформація про останній відомий стан IP та час, коли він почався. Коли аналіз доходить до наступного запису, модуль визначає різницю між часовими мітками і формує змістовний опис:

```
delta = int((curr_time - prev_time[ip]).total_seconds())
if prev_status[ip] == "ONLINE" and curr_status == "OFFLINE":
```

```
    det = f"Був ONLINE {delta} сек"
elif prev_status[ip] == "OFFLINE" and curr_status == "ONLINE":
    det = f"Був OFFLINE {delta} сек"
```

Це дозволяє побачити, наприклад, що робоче місце перебувало онлайн 5 годин, а потім впало; або що працівник був відсутній 23 хвилини і повернувся. Для реального процесу управління персоналом це має значення, бо дає можливість помітити нерівномірності чи аномалії в поведінці конкретного співробітника.

Окрема увага приділяється обробці часових періодів. Просте порівняння подій одна з одною не дає відповіді на питання “скільки часу працівник працював сьогодні?”. Тому система вводить чотири часові межі: за день, тиждень, місяць і весь доступний період. Це дозволяє оцінювати активність на різних рівнях часової деталізації. Межі задаються через об’єкти `datetime`:

```
now = datetime.now()
period_labels = [
    ("За день", now - timedelta(days=1)),
    ("За тиждень", now - timedelta(weeks=1)),
    ("За місяць", now - timedelta(days=30)),
    ("Весь час", df["Дата та час"].min())
]
```

Далі починається найцікавіша частина — обчислення активних інтервалів. CSV дає інформацію лише про точки зміни статусу, але тривалість цих інтервалів модуль повинен обчислити самостійно. Алгоритм працює так: для кожного IP формується підтаблиця подій у хронологічному порядку, після чого для кожного періоду система проходить рядки один за одним, визначаючи, чи попередній статус був ONLINE, і чи належить відповідний інтервал до обраного часового вікна.

```
if prev_status2 == "ONLINE":
    delta = int((curr_time2 - prev_time2).total_seconds())
    total_online += delta
```

Таким чином формується фактичний робочий час. Тут варто відмітити, що модуль ніде не робить припущень на кшталт “якщо інтервал не закрився, значить працівник продовжує працювати”. Якщо день не завершився нормальною подією OFFLINE, система вважає, що відлік тривав до останнього зафіксованого часу — це дозволяє уникнути штучних перекручувань даних.

Після того як обчислення завершено, всі результати для кожного співробітника збираються у фінальний DataFrame. Саме цей набір даних потім переноситься у Excel як компактна таблиця з чотирма числовими показниками. Для зручності читання система перетворює секунди у формат на кшталт “3 год 15 хв 22 сек”:

```
df_sum["ПІБ"] = df_sum["ПІБ"].str.replace(r"\s+", " ", regex=True).str.strip()
```

Це також очищає ПІБ від зайвих пробілів, бо в CSV при ручних змінах або некоректних вводах іноді можуть виникати дрібні викривлення.

Оформлення Excel — окрема складова, яка впливає на сприйняття даних. Модуль не просто будує таблиці, а й автоматично приводить документ до вигляду, який можна без вагань передати керівництву. Регулюється ширина колонок, застосовуються кольорові стилі, а статуси у вигляді текстових позначок виділяються зеленим або червоним залежно від значення. Наприклад:

```
if cell.value == "ONLINE":
    cell.fill = green_fill
    cell.font = Font(bold=True)
```

Такі візуальні елементи роблять документ швидко читабельним, навіть якщо у звіті сотні рядків.

У підсумку модуль продуктивності перетворює набір елементарних технічних сигналів на структурований часовий профіль діяльності

співробітника. Він дозволяє побачити закономірності, пікові періоди, місця, де активність переривалася, і навіть оцінити загальну стабільність підключення. Це важливо, тому що зазвичай такі речі залишаються непомітними: графік може падати, працівник може зникати з мережі, але без фіксації це губиться у потоці щоденної роботи. Тут же кожна подія набуває прямого змісту, а вся діяльність — зрозумілої часової структури.

3.6. Опис сценаріїв використання системи та демонстрація її ключових функціональних блоків

У реальному використанні система поводиться набагато простіше, ніж виглядає її внутрішній код. З погляду користувача все зводиться до кількох звичних дій у Telegram, хоча під капотом у цей момент запускається цілий ланцюг взаємопов'язаних процесів. Найбільш показовим є базовий сценарій, з якого взагалі починається робота - співробітник відкриває чат з ботом і надсилає команду `/start`. Далі вступає в дію логіка модулю ролевого доступу: функція `start` не просто вітається, а відразу підказує, в якому статусі користувач хоче працювати, і пропонує вибір між ролями "Керівник" та "Адміністратор". Це робиться через клавіатуру, щоб зняти зайві питання з форматами вводу:

```
def start(update: Update, context: CallbackContext):
    reply_markup = ReplyKeyboardMarkup(
        [["Керівник", "Адміністратор"]],
        one_time_keyboard=True,
        resize_keyboard=True
    )
    update.message.reply_text("Оберіть свою роль:", reply_markup=reply_markup)
    return STATE_ROLE
```

Демонстрація старту бота, та вибір ролі наведено в рисунку 3.1

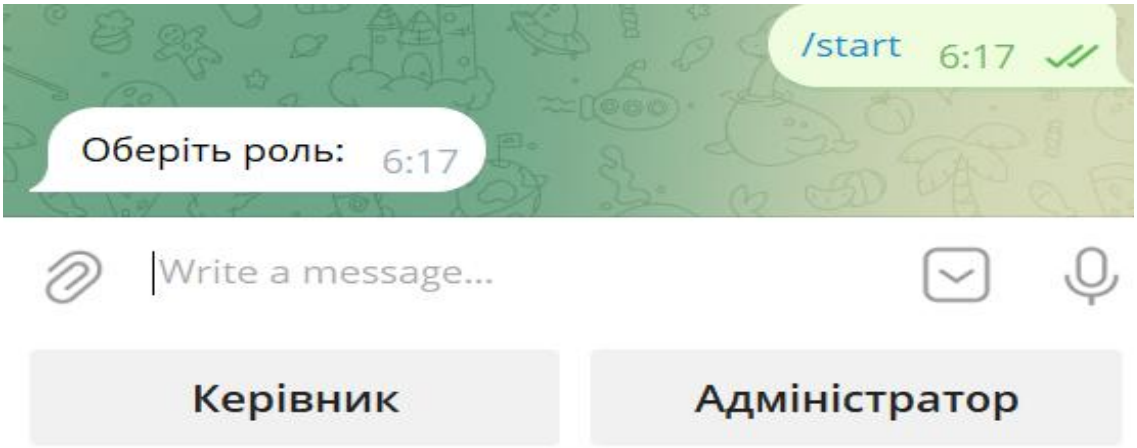


Рис.3.1. Вибір ролі

На цьому етапі система ще нічого не моніторить і не аналізує - вона просто чекає, кого перед собою має: людину, яка буде дивитися на інфраструктуру як адміністратор, чи того, хто цікавиться підсумковими показниками і звітами. Після вибору ролі працює наступний блок сценарію - перевірка пароля. Тут усе максимально прямолінійно: введений текст порівнюється з відповідним значенням у словнику `PASSWORDS`, а при успіху у `user_sessions` створюється запис про те, що цей `chat_id` авторизований з певною роллю і моніторинг у нього ще не запущений:

```
def password_entered(update: Update, context: CallbackContext):
    role = context.user_data.get('role')
    password = update.message.text
    chat_id = update.effective_chat.id

    if PASSWORDS.get(role) == password:
        user_sessions[chat_id] = {"role": role, "monitor": False}
        return STATE_MENU

    update.message.reply_text("Невірний пароль, спробуйте ще раз:")
    return STATE_PASSWORD
```

Якщо користувач, обирає потрібну роль, телеграм-бот запитує пароль для аутентифікації, це проілюстровано на рисунку 3.2.

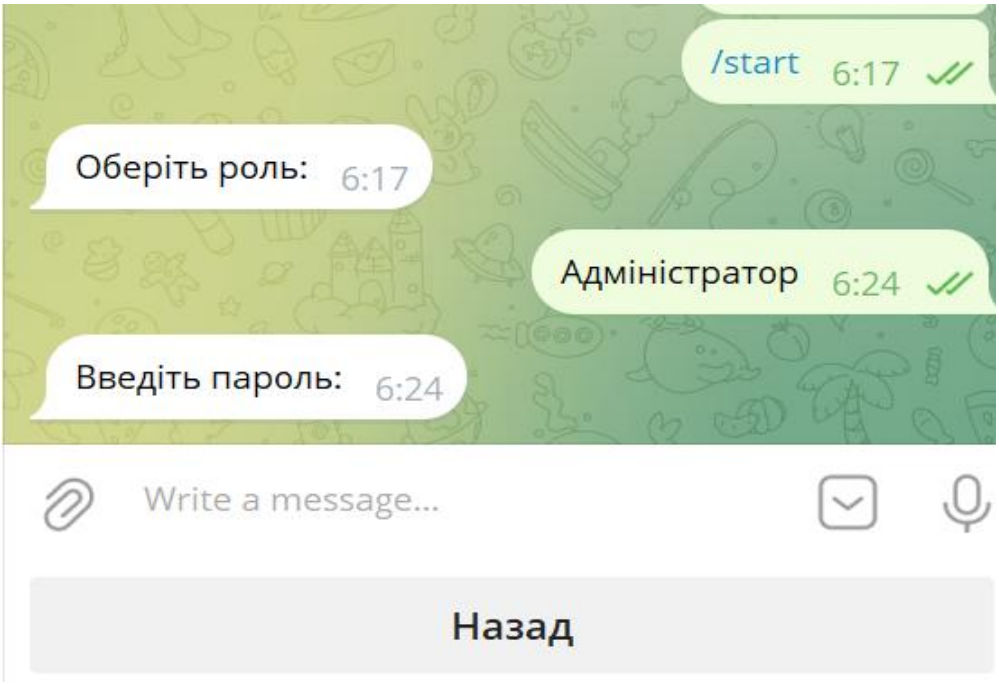


Рис.3.2. При виборі потрібної ролі, йде запит паролю

Якщо користувач ввів не правильний пароль – бот інформує користувача що це невірний пароль, потрібно ввести його правильно, або ж повернутися до головного меню, натиснувши кнопку “Назад”, ілюстрація продемонстрована на рисунку 3.3.

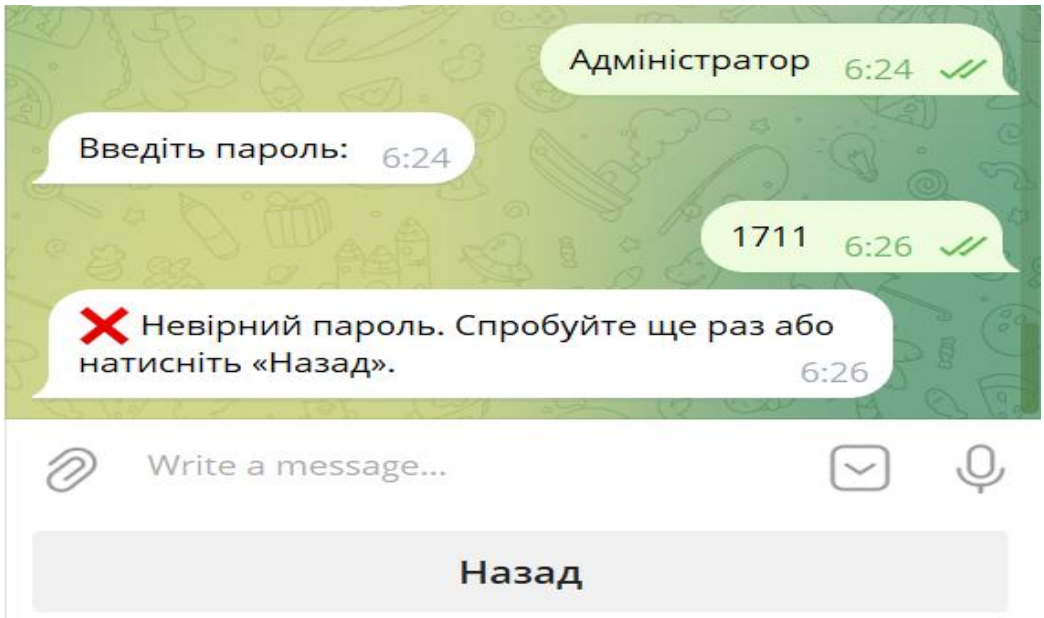


Рис.3.3. Неправильно вказаний пароль

Якщо користувач ввів потрібний та\або правильний пароль – бот повідомляє про це, і надає доступ до функцій які доступні для цієї ролі, проілюстровано на рисунку 3.4.

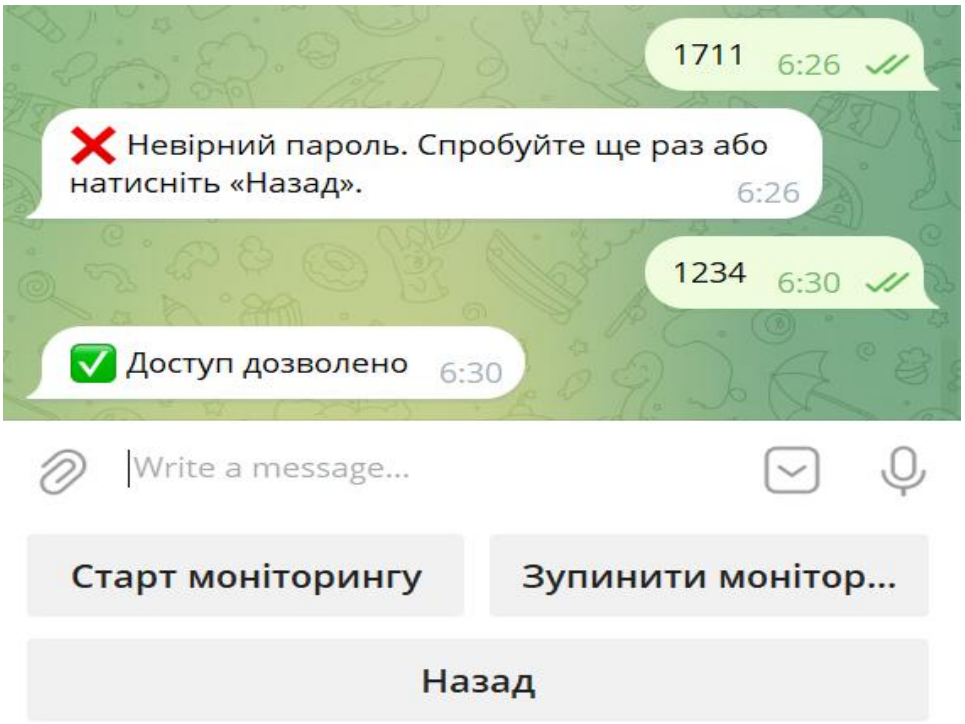


Рис.3.4. Доступ до функцій після успішної аутентифікації

У цей момент сценарій для користувача розходиться на дві гілки. Якщо це адміністратор, він бачить у меню дві кнопки - "Старт моніторингу" і "Вийти". Якщо це керівник - набір трохи ширший: "Старт моніторингу", "Звіт", "Вийти". Це формує два дещо різні стилі роботи з одним і тим самим комплексом. Адміністратор живе у режимі оперативного моніторингу, керівник - у режимі аналітики. Меню формується дуже прозоро:

```
if role == "boss":
    reply_markup = ReplyKeyboardMarkup(
        [["Старт моніторингу", "Звіт", "Вийти"]],
        resize_keyboard=True
    )
else:
    reply_markup = ReplyKeyboardMarkup(
```

```
        [["Старт моніторингу", "Вийти"]],
        resize_keyboard=True
    )
```

Так, як у ролі Керівник, більше повноважень, у цієї ролі з’являються додаткові функції - можливість отримати Звіт. Звіт можна отримати для всіх типів підключень - це підключення по кабелю та Wi-Fi (кнопка “Звіт (всі)”) або ж отримати звіт тільки підключень по Wi-Fi (кнопка “Звіт (тільки Wi-Fi)”. Функції Старт і Стоп моніторингу - для Керівника також доступні, як і для Адміністратора, проілюстровано на рисунку 3.5

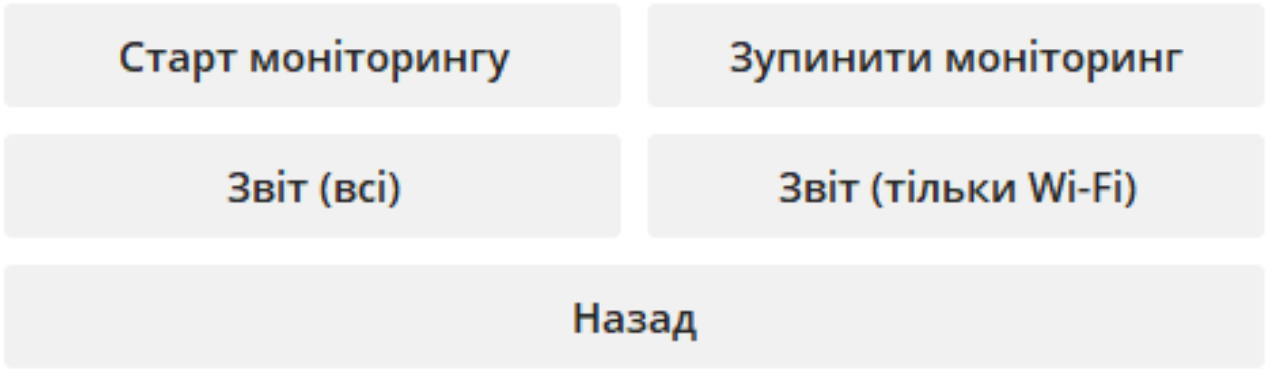


Рис.3.5. Функції, які доступні для ролі Керівник

У сценарії адміністратора наступний крок практично завжди один і той самий - запуск моніторингу. Він натискає "Старт моніторингу", після чого виконується гілка в `menu_handler`, яка створює для нього фоновий потік. Система ставить у його сесії `"monitor": True` і стартує `monitoring_thread` з передачею в нього об’єкта бота, `chat_id` та словника сесій:

```
elif text == "Старт моніторингу":
    session["monitor"] = True
    thread = threading.Thread(
        target=monitoring_thread,
        args=(context.bot, chat_id, hosts),
```

```
        daemon=True
    )
    thread.start()
    update.message.reply_text("Моніторинг запущено.")
    return STATE_MENU
```

З цього моменту адміністратор фактично отримує роль "спостерігача за подіями", хоча технічно він нічого більше не робить - просто дивиться на чат. У фоні ж працює окремий цикл, який по черзі проходить усі вузли зі списку `hosts`, виконує `ping` до кожного IP, порівнює поточний стан з попереднім і у разі зміни формує подію. Коли, наприклад, у головного бухгалтера падає мережа, боту надходить сигнал, і адміністратор бачить повністю зібране повідомлення, в якому є все - від IP до часу, скільки машина була онлайн до збою. Функція `format_node_message` додає до технічного інциденту людський контекст:

```
def format_node_message(h, online, duration=None):
    msg = (
        f"👤 Користувач: {h['name']}\n"
        f"🌐 IP: {h['ip']}\n"
        f"🔔 Статус: {'ONLINE' if online else 'OFFLINE'}\n"
    )
    if duration is not None:
        msg += f"⏱️ Тривалість попереднього стану: {duration}\n"
    return msg
```

Після натискання кнопки “Старт моніторингу”, нам надходить інформація про підключенні пристрої на той момент, коли ми запустили моніторинг, проілюстровано на рисунку 3.6.

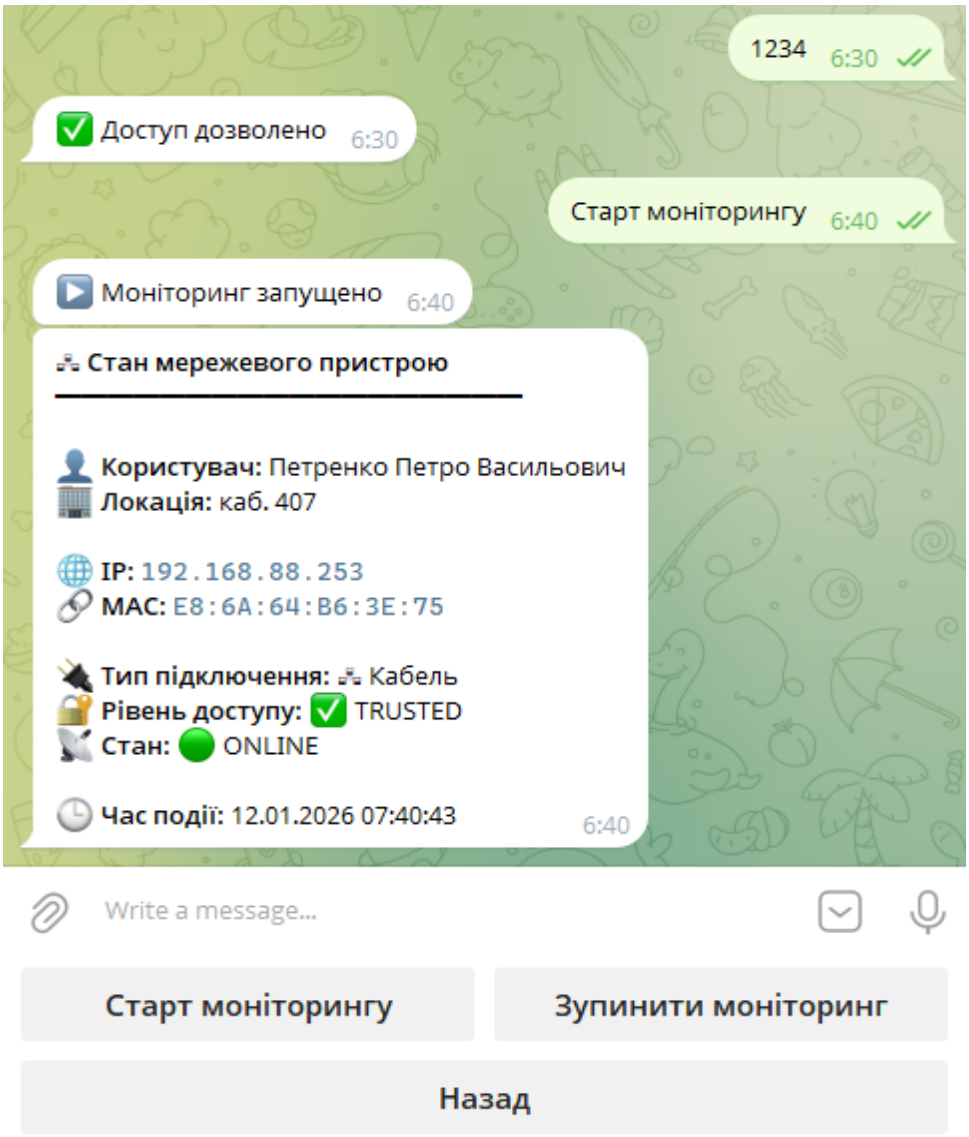


Рис.3.6. Функція запуску моніторингу

У цьому сценарії адміністратор отримує інструмент оперативного реагування: якщо якась станція починає "відвалюватися" кілька разів за день, це вже не випадкові скарги, а чіткий набір фактів, кожен з яких зафіксований і у чаті, і в журналі. У якийсь момент він може зупинити моніторинг для себе натиском "Вийти" - система видалить запис про його сесію з `user_sessions`, і потік просто завершить цикл, переставши надсилати йому повідомлення.

Для керівника сценарії виглядають інакше. Він теоретично теж може натиснути "Старт моніторингу" і дивитися на реальні події в режимі реального

часу, але основний інтерес з’являється в момент, коли накопичено достатньо даних для аналізу. У цьому випадку ключовою дією стає натискання "Звіт". Тут у `menu_handler` спрацьовує окрема гілка, причому система спочатку перевіряє, чи дійсно цей користувач має роль керівника:

```
elif text == "Звіт":
    if role != "boss":
        update.message.reply_text(
            "У вас недостатньо прав для перегляду звіту. Доступ лише для Керівника."
        )
        return STATE_MENU
    generate_and_send_excel(update, context)
    return STATE_MENU
```

Цей момент добре демонструє взаємодію між модулем ролевого доступу та модулем звітності: останній взагалі не запускається, якщо користувач не має достатніх прав. Якщо ж роль відповідає "boss", система переходить до іншого блоку - аналітики. Викликається `generate_and_send_excel`, яка на основі того самого CSV-файлу будує для керівника повноцінну картину активності. Під капотом це означає читання всіх записів журналу, сортування їх за часом, обробку інтервалів та обчислення сумарного онлайну за обрані періоди. Кінець сценарію для керівника виглядає досить просто - він отримує документ у чаті:

```
context.bot.send_document(
    chat_id=update.effective_chat.id,
    document=InputFile(excel_path)
)
```

Якщо Керівник хоче отримати звіт підключень по кабелю та Wi-Fi, він натискає кнопку “Звіт (всі)”, і отримує інформацію про підключення і відключення обох

типів, але якщо Керівник бажає отримати звіт тільки по Wi-Fi, він натискає “Звіт (тільки Wi-Fi)”, проілюстровано на рисунку 3.7.

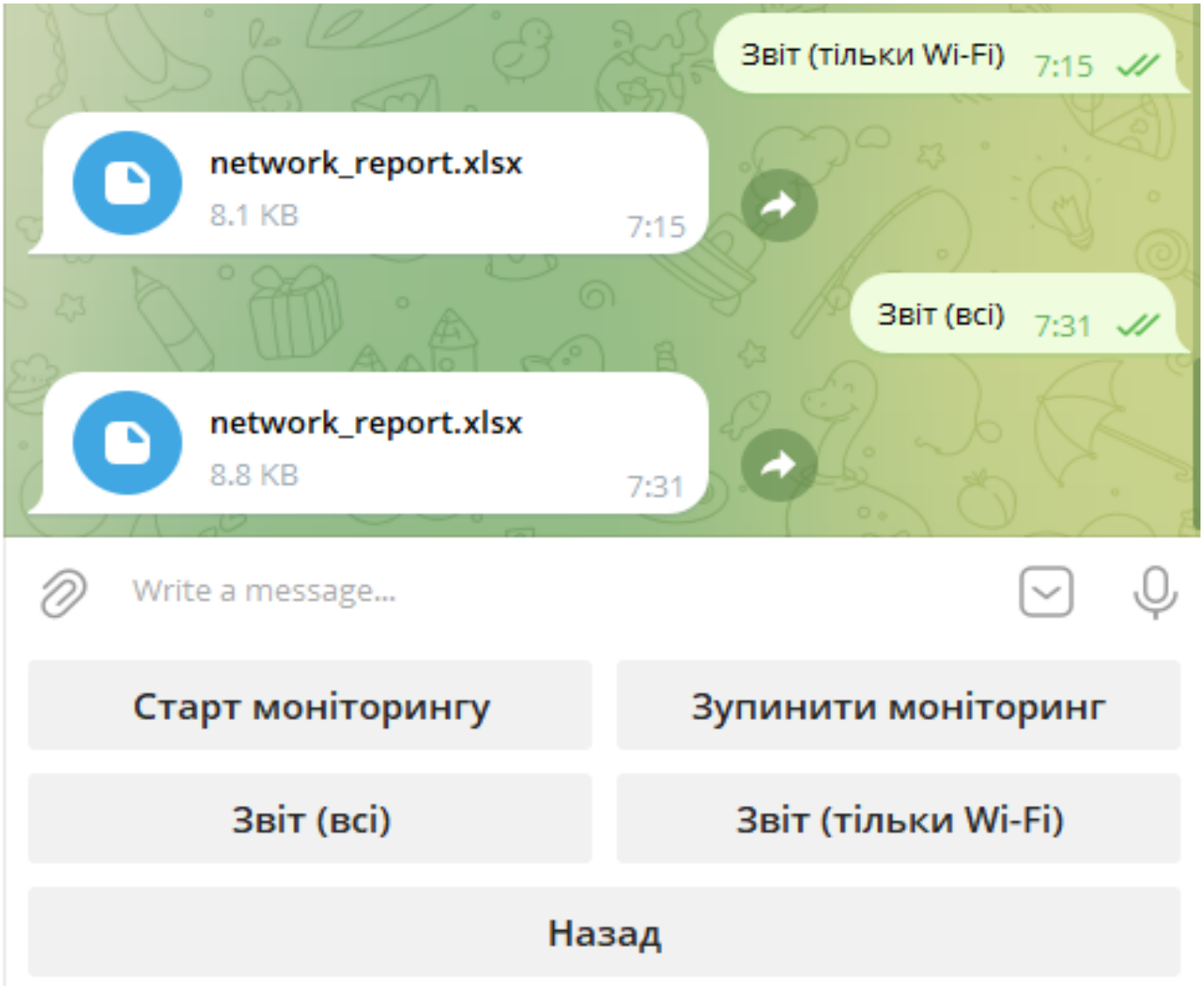


Рис.3.7. Отримання необхідного звіту у файлі Excel.

У цей момент для нього вся складна логіка розрахунків уже згорнута у вигляді трьох аркушів Excel - повний журнал, вибірка подій зі змінами статусу і зведена таблиця з часом перебування в мережі за день, тиждень, місяць та весь період. Сценарій виглядає дуже буденно: керівник відкриває файл і бачить, наприклад, що у певного працівника "Онлайн за тиждень" відверто менший за інших, або що в одного робочого місця аномально багато коротких офлайн-інтервалів. І вже тут рішення, що з цим робити, виходить за межі системи.

Окремо можна описати сценарій розбору конкретного інциденту. Припустимо, з’являється інформація, що "в бухгалтера вчора весь день падав

комп’ютер". У звичайній ситуації це залишилося б на рівні емоційної скарги. У цій системі керівник або адміністратор просто відкриває аркуш "Події" у звіті за потрібний період. Там, завдяки попередній обробці, залишилися лише рядки, де дійсно були переходи ONLINE/OFFLINE, і до кожного з них система додала текст на кшталт "Був ONLINE 2 год 17 хв" або "Був OFFLINE 13 хв". Тобто за один погляд можна побачити, скільки разів станція "падала", як довго не працювала кожного разу і чи це разовий випадок, чи вже тенденція.

Ще один характерний сценарій пов'язаний з роботою з "проблемними" вузлами. Якщо адміністратор помічає, що якийсь IP з’являється в чаті надто часто, він може поставити собі окреме завдання - відслідкувати його поведінку за тиждень. Для цього достатньо через деякий час попросити керівника згенерувати звіт або зробити це самому, якщо він має відповідні права. У файлі "Підсумок" для цього IP буде чітко видно, що онлайн за тиждень складав, наприклад, істотно менше, ніж у інших співробітників, а аркуш "Події" покаже, що падіння відбувались, скажімо, по 10 - 15 разів на день, з короткими, але регулярними офлайнами. Такий сценарій вже виводить систему на рівень не просто моніторингу, а діагностики, наведено в рисунку 3.8.

	1	2	3	4	5	6	7
1	datetime	ip	mac	name	room	status	connection
45	2026-01-08 02:23:16		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
46	2026-01-12 07:40:43	192.168.88.253	E8:6A:64:B6:3E:75	Петренко Петро Ва	каб. 407	ONLINE	Кабель
47	2026-01-12 07:45:07		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
48	2026-01-12 07:45:07		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
49	2026-01-12 07:48:47		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
50	2026-01-12 07:48:47		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
51	2026-01-12 07:50:54		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi

Log Summary Daily stats

Рис.3.8. Інформація про відключення\підключення пристроїв

Нарешті, є сценарій, який можна назвати "тихим" - коли система працює у фоновому режимі, а користувачі до неї екстрено не звертаються. Вона продовжує зберігати події у CSV, тихо опитувати вузли, не засипати чат повідомленнями, якщо все стабільно, і чекати моменту, коли керівнику знадобиться зріз ситуації. У цей момент вся накопичена історія подій перетворюється на звіт, що показує

поведінку інфраструктури і персоналу у ретроспективі. Саме цей сценарій добре демонструє ідею системи - вона не вимагає постійної уваги, але в потрібний момент має під рукою повний набір фактів.

У результаті вся логіка комплексу виявляється вплетеною у кілька життєвих сценаріїв: щоденний моніторинг адміністратором, періодичний запит звітів керівником, розбір інцидентів та аналіз стабільності окремих вузлів. Кожен із цих сценаріїв запускає свій набір функціональних блоків - FSM авторизації, фоновий потік моніторингу, механізм журналювання, аналітичний модуль формування Excel - але з точки зору користувача все це залишається "за кадром". Він працює через кілька кнопок у Telegram, а система робить за нього всю технічну частину - фіксує, рахує, структурує і віддає інформацію у вигляді, придатному для прийняття рішень.

ВИСНОВКИ

Поглиблене вивчення систем моніторингу IT-інфраструктури показало, що технологічний ландшафт давно вийшов за межі простих моделей контролю. Розвиток підходів до спостереження за технічними системами продемонстрував, що традиційні рішення вже не можуть забезпечити повноцінну видимість процесів, які постійно змінюються. Спершу здавалося, що достатньо відстежувати доступність вузлів і збирати логи, але аналіз історії розвитку моніторингових засобів поступово відкрив зовсім іншу картину: сучасні інфраструктури поводяться непередбачувано, а подієвий характер змін вимагає набагато делікатніших механізмів спостереження. Саме тому таке значення отримали моделі на кшталт гібридних архітектур, алгоритмів виявлення аномалій, а також підходи до нормалізації сирих телеметричних даних, які без обробки майже не дають користі.

Разом із тим, поступово вимальовувалася паралельна важлива лінія: поведінкові аспекти роботи персоналу. Величезний обсяг даних про активність співробітників, їх взаємодію з технікою та непередбачені зміни у робочих процесах виявилися не менш важливими, ніж технічні індикатори. Теоретичні передумови інтеграції цих двох сфер дали можливість побачити, що вони далеко не такі відокремлені, як може здатися. Технічна подія в інфраструктурі часто віддзеркалює процеси, які мають людське походження: зміна доступності системи, раптовий спад навантаження або повторювані паузи у роботі користувача. Поступове розуміння цього логічно підвело до формування концепції єдиного аналітичного ядра.

Під час дослідження інструментів для моніторингу стало очевидно, що навіть дуже розвинені системи не враховують специфічних вимог організацій щодо контролю персоналу. Багато з них добре справляються з технічними аспектами, але не здатні поєднати їх із поведінковими закономірностями. Дослідження обмежень таких платформ допомогло сформувати комплекс бізнес-вимог, серед яких одними з ключових стали оперативність, прозорість аналітики й можливість працювати з різними джерелами даних одночасно. Розгляд

моделей загроз та ризиків кібербезпеки додав важливий шар розуміння: система має не лише аналізувати поведінку, а й гарантувати, що доступ до даних відбувається в рамках чітких ролевих правил.

Формування методичних принципів побудови аналітичного ядра стало опорою для подальшої розробки. Поступово вималювалася необхідність працювати не з усіма даними підряд, а лише з тими, що мають статус події. Саме подія створює часову межу, яка потім перетворюється на інтервал аналізу. Такий підхід дозволив значно скоротити обсяг оброблюваної інформації, зберігши при цьому точність. Аналітичні обґрунтування інтегрованої моделі підтвердили, що система може працювати стійко навіть за великої кількості паралельних процесів.

Практична реалізація комплексу стала своєрідним випробуванням теоретичних висновків. Побудова серверної архітектури зі збором подій, базою даних і API продемонструвала, наскільки складно забезпечити синхронність різних процесів. Подієвий підхід довів свою ефективність: кожен перехід стану створював чітко визначений часовий проміжок, який вже можна було аналізувати без втручань. Розгортання модуля ролевого доступу дозволило забезпечити контроль, де різні типи користувачів отримують різні рівні даних, що підвищило керованість та безпечність системи.

Окремо проявили себе модулі моніторингу та контролю продуктивності. Перший показав, що автоматична фіксація інцидентів дозволяє будувати хронологію інфраструктурних подій без спотворень і пропусків. Другий – що підрахунок продуктивності за часовими інтервалами дає значно прозорішу картину, ніж спроби оцінювати поведінку за непрямими метриками. Вдалось побачити, як події в мережі перетворюються на упорядковані показники активності, і як ця інформація лягає в основу звітності, зручної для аналізу керівництвом.

Опис сценаріїв роботи системи дозволив переконатися, що вона не є статичною конструкцією. Вона реагує на зміни, масштабується, адаптується до різних моделей роботи організації. Взаємодія користувача з інтерфейсом, обробка запитів, формування звітів – усе це продемонструвало, що побудований

комплекс може функціонувати в середовищі з будь-якою динамікою, без втрати достовірності даних.

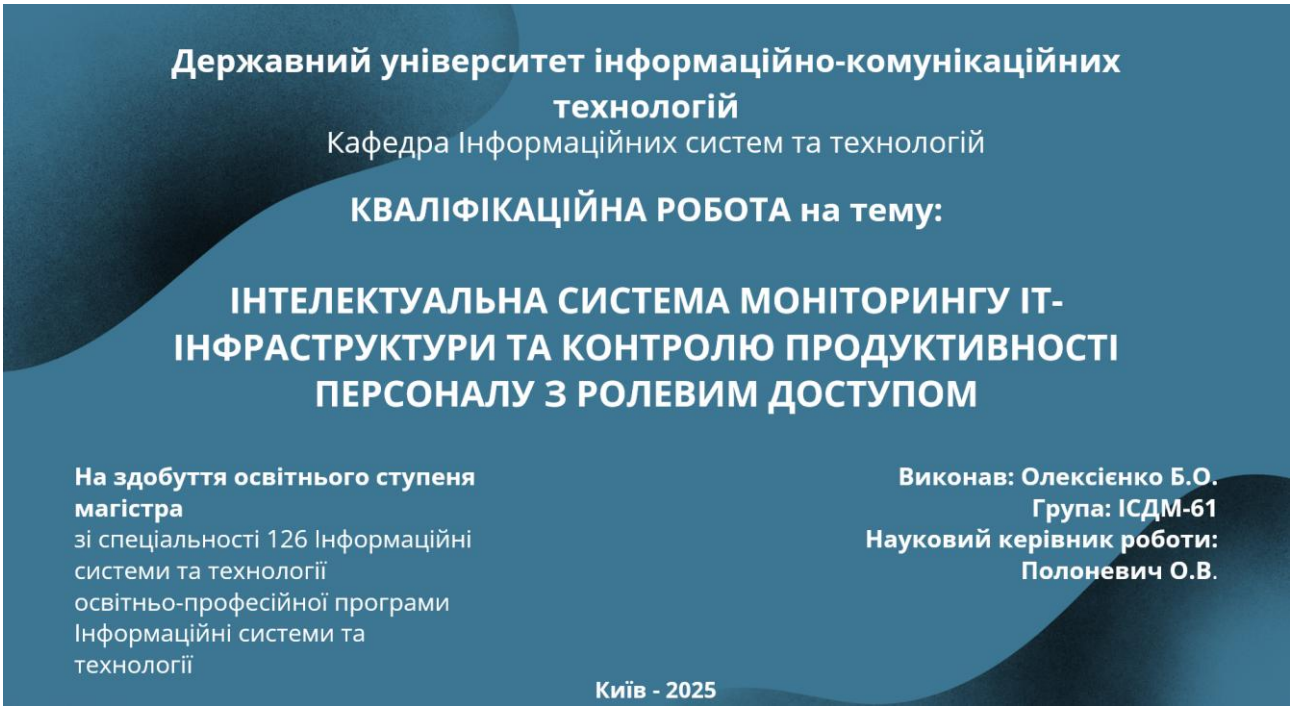
Узагальнюючи здобуті результати, можна сказати, що поєднання технічного моніторингу з контролем продуктивності стало не просто можливим, а природним. Виявилося, що ці два сегменти доповнюють один одного, утворюючи спільну аналітичну площину, де кожна подія має свій сенс і вплив на загальний результат. Запропонований підхід здатен стати основою для подальшого розвитку – від розширення аналітичних модулів до інтеграції прогнозних моделей та інтелектуальних механізмів реагування.

ПЕРЕЛІК ПОСИЛАНЬ

1. **Bot API** : official documentation. Telegram. URL: https://core.telegram.org/bots/api
2. **CSV File Format** : specification and usage. W3C. URL: https://www.w3.org/TR/tabular-data-model/
3. **Datadog Documentation** : monitoring and observability. Datadog Inc. URL: https://docs.datadoghq.com/
4. **Dynatrace Documentation** : software intelligence platform. Dynatrace LLC. URL: https://docs.dynatrace.com/
5. **Elastic Stack Overview** : official documentation. Elastic N.V. URL: https://www.elastic.co/elastic-stack/
6. **ENISA**. Guidelines on Security Monitoring. 2020. URL: https://www.enisa.europa.eu/publications/guidelines-on-security-monitoring
7. **Grafana Documentation** : visualization and analytics. Grafana Labs. URL: https://grafana.com/docs/
8. **Graylog Documentation** : log management platform. Graylog Inc. URL: https://go2docs.graylog.org/
9. **Icinga Documentation** : monitoring system. Icinga GmbH. URL: https://icinga.com/docs/
10. **IEEE**. Standard for System and Software Logging. IEEE Std 1003.1-2017 (reaffirmed 2018). URL: https://ieeexplore.ieee.org/document/8277153
11. **ITU-T**. Recommendation X.1051 : Information security management. 2019. URL: https://www.itu.int/rec/T-REC-X.1051
12. **Kleppmann M.** Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2018. 616 p.
13. **Nagios Documentation** : monitoring concepts. Nagios Enterprises. URL: https://www.nagios.org/documentation/
14. **New Relic Documentation** : observability platform. New Relic Inc. URL: https://docs.newrelic.com/
15. **Openpyxl Documentation** : Excel file processing. URL: https://openpyxl.readthedocs.io/

16. **Pandas Documentation** : data analysis library. URL: https://pandas.pydata.org/docs/
17. **PRTG Network Monitor Manual**. Paessler AG. URL: https://www.paessler.com/manuals/prtg
18. **Prometheus Documentation** : monitoring system and time series database. URL: https://prometheus.io/docs/
19. **Pythonping Documentation** : ICMP ping library. URL: https://pythonping.readthedocs.io/
20. **Python Software Foundation**. Python 3 Documentation. URL: https://docs.python.org/3/
21. **Splunk Documentation** : machine data analytics. Splunk Inc. URL: https://docs.splunk.com/
22. **VictoriaMetrics Documentation** : time series database. URL: https://docs.victoriametrics.com/
23. **Zabbix Documentation** : enterprise monitoring. Zabbix LLC. URL: https://www.zabbix.com/documentation
24. **ISO/IEC 27001:2022**. Information security management systems. URL: https://www.iso.org/standard/82875.html
25. **ISO/IEC 27002:2022**. Information security controls. URL: https://www.iso.org/standard/75652.html
26. **SRE Book**. Site Reliability Engineering. Google. 2018. URL: https://sre.google/sre-book/table-of-contents/
27. **Observability Engineering** / ed. Charity M. O'Reilly Media, 2022. URL: https://www.oreilly.com/library/view/observability-engineering/9781492076438/
28. **RFC 792**. Internet Control Message Protocol. IETF. URL: https://datatracker.ietf.org/doc/html/rfc792
29. **RFC 8633**. Network Telemetry Framework. IETF, 2019. URL: https://datatracker.ietf.org/doc/html/rfc8633
30. **Uptime Institute**. IT Infrastructure Monitoring Best Practices. 2021. URL: https://uptimeinstitute.com/resources

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

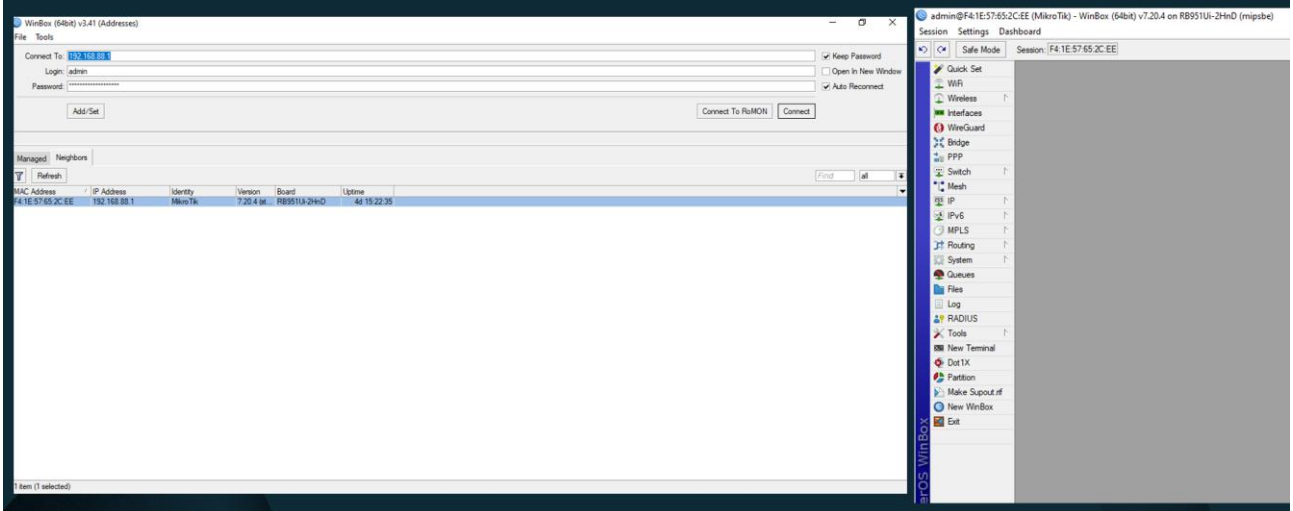


Актуальність	Сучасні системи моніторингу ІТ-інфраструктури переважно фіксують технічні події, але не формують управлінської аналітики. Тому актуальним є створення системи, що поєднує моніторинг, аналітику продуктивності та ролевий доступ до даних.
Мета	Створення підходу до побудови інтелектуальної системи моніторингу, здатної збирати різнорідні події в реальному часі, структурувати їх, визначати динаміку роботи мережевих вузлів і перетворювати технічні зміни на управлінські показники.
Об’єкт	Процес моніторингу та аналізу станів ІТ-інфраструктури у поєднанні з відстеженням активності персоналу.
Предмет	Методи інтеграції процесів моніторингу ІТ-інфраструктури та контролю продуктивності персоналу в єдину систему з ролевим доступом
Завдання	Проаналізувати сучасні підходи до моніторингу ІТ-інфраструктури, спроєктувати архітектуру системи з ролевим доступом, реалізувати механізми збору та обробки подій, формувати аналітичні звіти та оцінку ефективності.


Загальна архітектура системи
Робочі станції та IT-інфраструктура
Модуль збору подій (Online/Offline)
Модуль обробки та зберігання данних
Аналітичний модуль
Telegram-інтерфейс користувача
Аналітичні Excel звіти

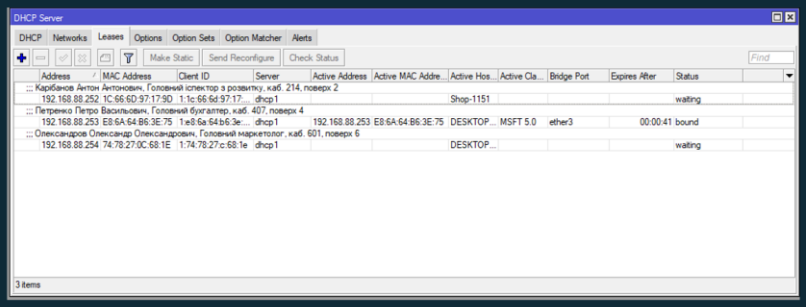

Інструменти реалізації
Для реалізації проекту я використовував маршрутизатор Mikrotik який базується на ОС - RouterOS
Для контролю цим мережевим пристроєм я використовував програмне забезпечення Winbox
Winbox

Головне меню ПЗ WinBox

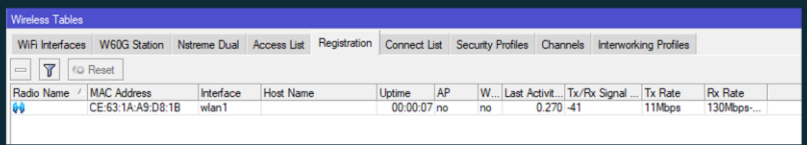


IP адресація та Wi-Fi клієнти

Інформація про підключення клієнтів по кабелю Ethernet



Інформація про підключення клієнтів по бездротової технології (Wi-Fi)

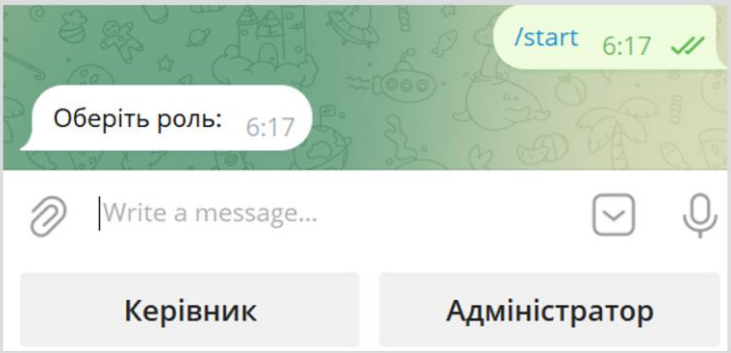


Назва та логотип Telegram-бота

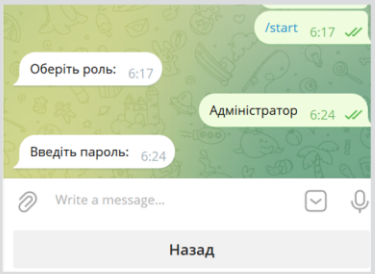
UP-DOWNBOT

Реалізація проекту

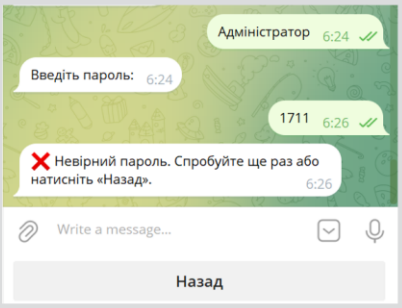
Після запуску скрипта і команди /start - бот надсилає повідомлення про вибір ролі.



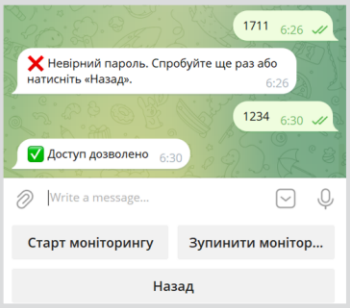
Після того, як ми обрали потрібну нам роль (в данному випадку - адміністратор), ми повинні ввести пароль



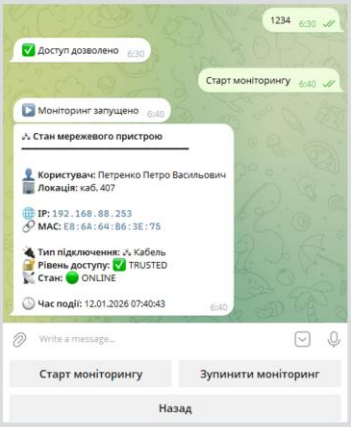
Якщо ж ми ввели пароль невірно - бот надсилає нам повідомлення про помилку, і дає можливість написати його ще раз, або повернутися до головного меню з вибором ролі



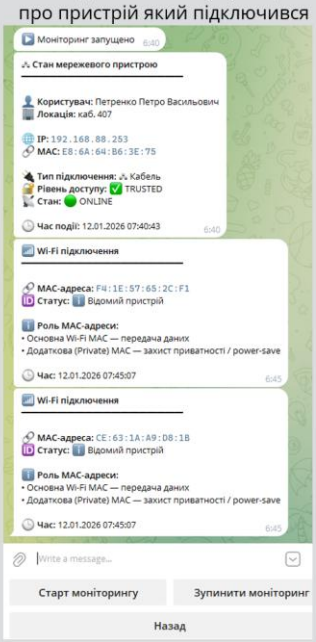
Якщо пароль вірний - бот повідомляє що доступ дозволено, та надає можливість скористатися функціями



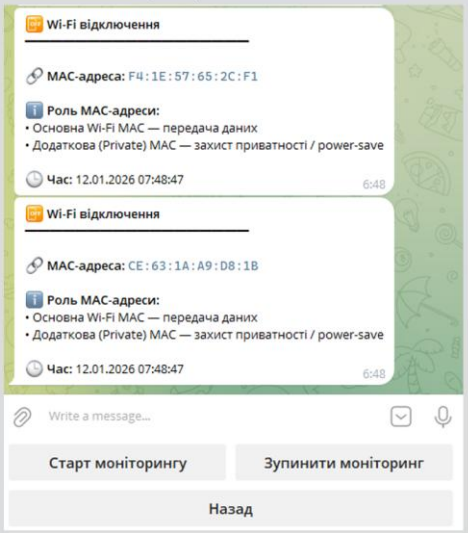
Після натискання кнопки “Старт моніторингу”, нам надходить інформація про підключенні пристрої на той момент, коли ми запустили моніторинг



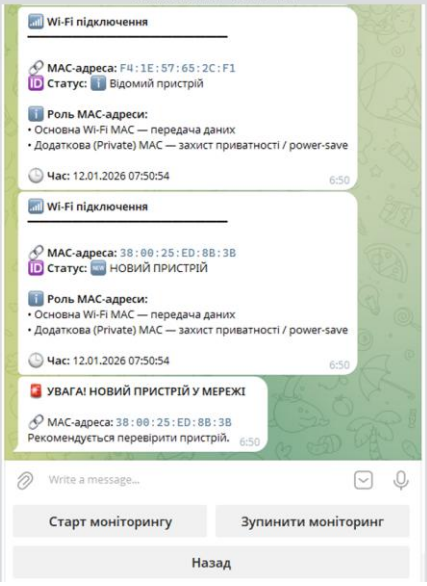
Якщо в момент моніторингу, до мережі під’єднуються пристрої, бот надсилає нам інформацію - про пристрій який підключився



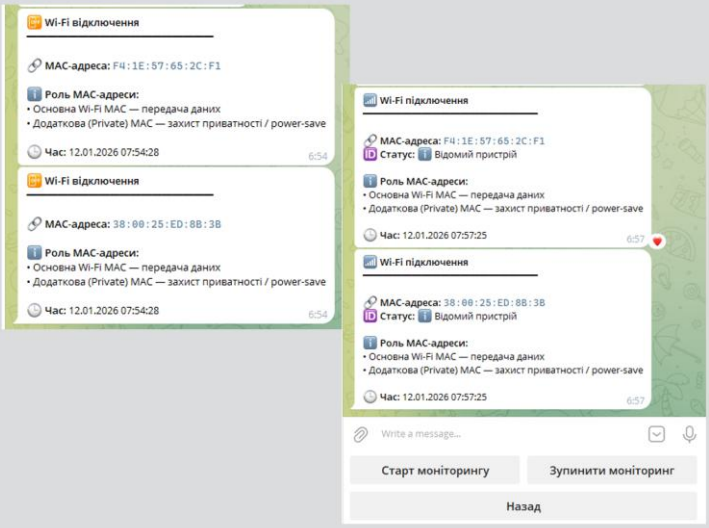
Якщо пристрій від’єднується, бот також надає нам інформацію, який пристрій від’єднався



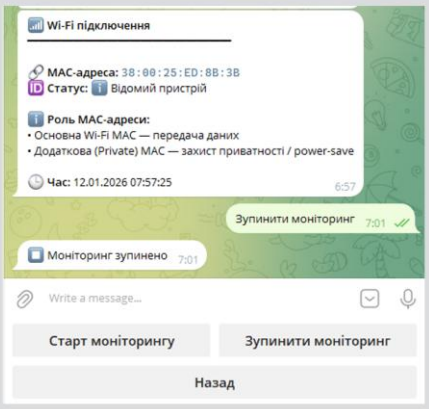
Якщо пристрій, який під'єднався до мережі вперше - бот надсилає інформацію з попередженням, що на цей пристрій потрібно звернути увагу, бо він раніше не був під'єднаний і в "Статус" - інформується що це НОВИЙ ПРИСТРІЙ



Але, якщо новий пристрій від'єднався, та ще раз підключився до мережі - він стає вже відомим раніше для мережі, тому в "Статус" - інформується що це раніше відомий пристрій і не надсилає попередження

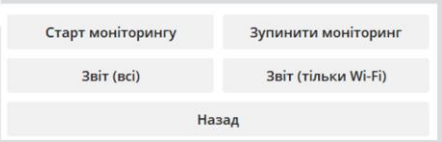


Якщо натиснути "Зупинити моніторинг" - інформація про відключення/підключення пристроїв буде недоступна

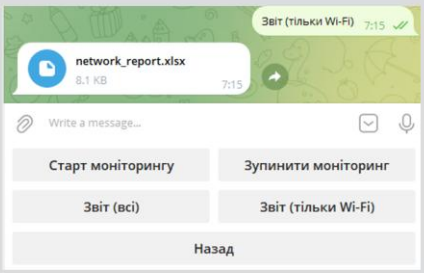


Якщо виникла потреба змінити користувача з адміністратора на керівника, ми натискаємо кнопку "Назад", нас повертає до меню вибору ролі

Так, як у ролі Керівник, більше повноважень, у цієї ролі з'являються додаткові функції - можливість отримати Звіт. Звіт можна отримати для всіх типів підключень - це підключення по кабелю та Wi-Fi (кнопка "Звіт (всі)") або ж отримати звіт тільки підключень по Wi-Fi (кнопка "Звіт (тільки Wi-Fi)". Функції Старт і Стоп моніторингу - для Керівника також доступні, як і для Адміністратора



При натисканні на кнопку “Звіт (тільки Wi-Fi)”, бот одразу автоматично генерує файл в Excel



В цьому звіті є вся інформація, яка відбувалася на протязі певного періоду часу. Дата і час, пристрій, статус, тип пікдлючення

	datetime	ip	mac	name	room	status	connection
9	2026-01-08 01:41:02		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
10	2026-01-08 01:42:02		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
11	2026-01-08 01:42:02		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
12	2026-01-08 01:43:41		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
13	2026-01-08 01:43:41		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
14	2026-01-08 02:07:29		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
15	2026-01-08 02:07:29		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
16	2026-01-08 02:10:04		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
17	2026-01-08 02:10:04		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
18	2026-01-08 02:10:04		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
19	2026-01-08 02:10:04		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
20	2026-01-08 02:10:10		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
21	2026-01-08 02:10:10		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
22	2026-01-08 02:10:10		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
23	2026-01-08 02:10:10		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
24	2026-01-08 02:10:57		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
25	2026-01-08 02:10:57		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
26	2026-01-08 02:10:57		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
27	2026-01-08 02:10:57		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
28	2026-01-08 02:12:41		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
29	2026-01-08 02:12:41		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
30	2026-01-08 02:12:58		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
31	2026-01-08 02:12:58		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
32	2026-01-08 02:22:37		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
33	2026-01-08 02:22:37		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
34	2026-01-08 02:23:16		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
35	2026-01-08 02:23:16		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
36	2026-01-12 07:45:07		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
37	2026-01-12 07:45:07		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
38	2026-01-12 07:48:47		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
39	2026-01-12 07:48:47		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
40	2026-01-12 07:50:54		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
41	2026-01-12 07:50:54		38:00:25:ED:8B:3B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
42	2026-01-12 07:54:28		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
43	2026-01-12 07:54:28		38:00:25:ED:8B:3B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
44	2026-01-12 07:57:25		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
45	2026-01-12 07:57:25		38:00:25:ED:8B:3B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi

Log | Summary | Daily stats

Якщо зайти на окрему вкладку “Щоденна статистика”, Керівник може побачити коли, скільки часу, який пристрій провів за день у мережі

	Date	Name	IP	MAC	Online minutes
2	2026-01-08	Wi-Fi device		F4:1E:57:65:2C:F1	6
3	2026-01-08	Wi-Fi device		CE:63:1A:A9:D8:1B	6
4	2026-01-12	Wi-Fi device		F4:1E:57:65:2C:F1	7
5	2026-01-12	Wi-Fi device		CE:63:1A:A9:D8:1B	3
6	2026-01-12	Wi-Fi device		38:00:25:ED:8B:3B	3

Log | Summary | Daily stats

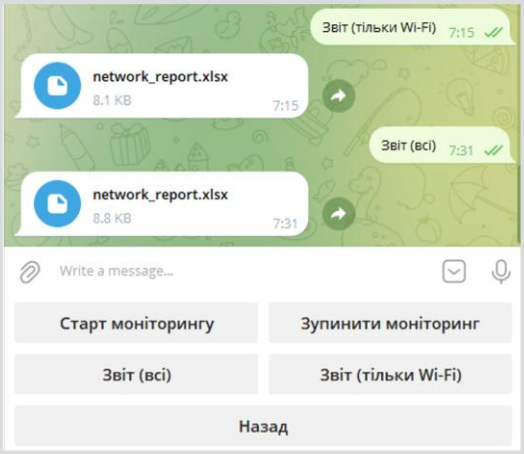
Готово

Якщо зайти на окрему вкладку “Загалом”, Керівник може побачити коли останній раз пристрій був у оффлайн, та скільки загалом часу він був в мережі за весь період з першого підключеннння до останнього відключення

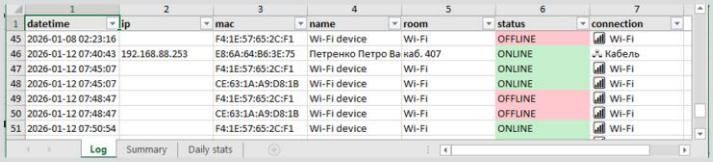
	Name	IP	MAC	First Online	Last Offline	Total Online (min
2	Wi-Fi device		F4:1E:57:65:2C:F1	2026-01-08 1:38:28	2026-01-12 7:54:28	31
3	Wi-Fi device		CE:63:1A:A9:D8:1B	2026-01-08 1:38:28	2026-01-12 7:48:47	10
4	Wi-Fi device		38:00:25:ED:8B:3B	2026-01-12 7:50:54	2026-01-12 7:54:28	21

Log | Summary | Daily stats

Якщо Керівник хоче отримати звіт підключень по кабелю та Wi-Fi, він натискає кнопку “Звіт (всі)”, і отримує інформацію про підключення і відключення обох типів



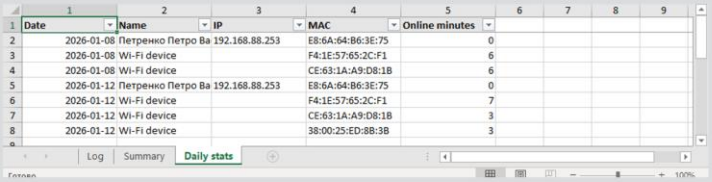
Всі логи



	datetime	ip	mac	name	room	status	connection
45	2026-01-08 02:23:16		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
46	2026-01-12 07:40:43	192.168.88.253	E8:6A:64:B6:3E:75	Петренко Петро Ва	каб. 407	ONLINE	Кабель
47	2026-01-12 07:45:07		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
48	2026-01-12 07:45:07		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi
49	2026-01-12 07:48:47		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
50	2026-01-12 07:48:47		CE:63:1A:A9:D8:1B	Wi-Fi device	Wi-Fi	OFFLINE	Wi-Fi
51	2026-01-12 07:50:54		F4:1E:57:65:2C:F1	Wi-Fi device	Wi-Fi	ONLINE	Wi-Fi

Log | Summary | Daily stats

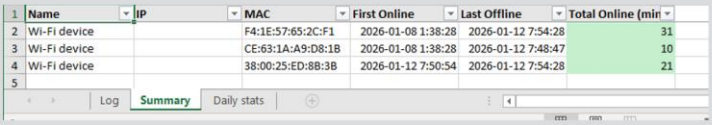
Щоденний звіт



	Date	Name	IP	MAC	Online minutes
2	2026-01-08	Петренко Петро Ва	192.168.88.253	E8:6A:64:B6:3E:75	0
3	2026-01-08	Wi-Fi device		F4:1E:57:65:2C:F1	6
4	2026-01-08	Wi-Fi device		CE:63:1A:A9:D8:1B	6
5	2026-01-12	Петренко Петро Ва	192.168.88.253	E8:6A:64:B6:3E:75	0
6	2026-01-12	Wi-Fi device		F4:1E:57:65:2C:F1	7
7	2026-01-12	Wi-Fi device		CE:63:1A:A9:D8:1B	3
8	2026-01-12	Wi-Fi device		38:00:25:ED:8B:3B	3

Log | Summary | Daily stats

Загальний звіт



	Name	IP	MAC	First Online	Last Offline	Total Online (min
2	Wi-Fi device		F4:1E:57:65:2C:F1	2026-01-08 1:38:28	2026-01-12 7:54:28	31
3	Wi-Fi device		CE:63:1A:A9:D8:1B	2026-01-08 1:38:28	2026-01-12 7:48:47	10
4	Wi-Fi device		38:00:25:ED:8B:3B	2026-01-12 7:50:54	2026-01-12 7:54:28	21
5						

Log | Summary | Daily stats

ВИСНОВОК

Під час виконання дипломної роботи було здійснено:

1. Аналіз сучасних підходів до моніторингу IT-інфраструктури та засобів контролю мережевих підключень.

2. Спроєктовано архітектуру системи моніторингу з ролевим доступом користувачів.

3. Реалізовано механізми збору подій про підключення та відключення мережевих пристроїв на базі маршрутизатора MikroTik.

4. Розроблено Telegram-бот для оперативного сповіщення та взаємодії з користувачами.

5. Реалізовано формування аналітичних звітів у форматі Excel для подальшого аналізу та прийняття управлінських рішень.

АПРОБАЦІЇ

VII МІЖНАРОДНА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ «СУЧАСНИЙ СТАН ТА ПЕРСПЕКТИВИ РОЗВИТКУ IOT»

Тема : МЕТОДИ ЗАБЕЗПЕЧЕННЯ МАСШТАБОВАНОСТІ У ПЛАТФОРМАХ ДЛЯ ОБРОБКИ ВЕЛИКИХ ТРАНЗАКЦІЙ У ЦИФОРОВОМУ БАНКІНГУ

Дата : 15 квітня 2025 року

III Всеукраїнська науково-технічна конференція "Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу"

Тема : ОГЛЯД СУЧАСНИХ ІНСТРУМЕНТІВ МОНІТОРИНГУ ІТ-ІНФРАСТРУКТУРИ У КОРПОРАТИВНОМУ СЕРЕДОВИЩІ

Дата : 18 листопада 2025 року

ДЯКУЮ ЗА УВАГУ!