ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: **«Автоматизація процесів реєстрації та обробки IT-інцидентів на основі агрегації та фільтрації подій моніторингу»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_2022
Гавриленко Максим Миколайович
Керівник: Поночовний Юрій
Леонідович
Рецензент:

Полтава – 2026 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Юрій УТКІН
«10» червня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Гавриленка Максима Миколайовича

1. Тема роботи: «Автоматизація процесів реєстрації та обробки IT-інцидентів на основі агрегації та фільтрації подій моніторингу»,
керівник роботи д.т.н., професор, професор кафедри інформаційних систем та технологій Поночовний Юрій Леонідович.
Затверджено наказом закладу вищої освіти від «10» квітня 2026 року № 383-ст

2. Строк подання здобувачем вищої освіти роботи «08» червня 2026 року

3. Вихідні дані до роботи: технічні та методологічні матеріали, зокрема методологія ITIL 4 (AXELOS), стандарт ISO/IEC/IEEE 29148:2018, офіційна документація Microsoft System Center Operations Manager, Python 3.11, FastAPI, SQLAlchemy, React 18, СКБД PostgreSQL 15 та брокера повідомлень RabbitMQ 3.12.

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

Розділ 1. Аналіз предметної області та постановка задачі
Розділ 2. Проектування мікросервісної системи автоматизації обробки IT-інцидентів
Розділ 3. Реалізація та тестування інформаційної системи

5. Перелік графічного матеріалу: схеми, рисунки, графіки, діаграми за темою та об’єктом дослідження.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Дивнич О. Д., к. е. н., доцент, доцент кафедри економіки та публічного управління	01.04.2026 р.	29.05.2026 р.

7. Дата видачі завдання «10» червня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Вибір і затвердження теми роботи	02.06.2025 р. – 04.06.2025	
2	Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу	05.06.2025 р. – 10.06.2025 р.	
3	Опрацювання джерел інформації	11.06.2025 р. – 15.06.2025 р.	
4	Збір, вивчення і обробка інформації, необхідної для виконання роботи	16.06.2025 р. – 20.06.2025 р.	
5	Виконання теоретико-методологічного розділу роботи	08.09.2025 р. – 01.11.2025 р.	
6	Виконання дослідницько-аналітичного розділу роботи	19.01.2026 р. – 14.02.2026 р.	
7	Виконання проєктно-рекомендаційного розділу роботи	16.02.2026 р. – 31.03.2026 р.	
8	Оцінювання економічної ефективності результатів дослідження	01.04.2026 р. – 29.05.2026 р.	
9	Оформлення тексту роботи	01.06.2026 р. – 06.06.2026р.	
10	Попередній захист роботи на кафедрі	08.06.2026 р.	
11	Доопрацювання роботи з урахуванням зауважень і пропозицій	09.06.2026 р. – 10.06.2026 р.	
12	Нормоконтроль	11.06.2026 р. – 15.06.2026 р.	
13	Захист кваліфікаційної роботи	16.06.2026 р – 18.06.2026 р.	

Здобувач вищої освіти **Максим ГАВРИЛЕНКО**

Керівник роботи **Юрій ПОНОЧОВНИЙ**

ЗМІСТ

ВСТУП

Сучасні організації незалежно від галузі діяльності у все більшій мірі залежать від безперебійного функціонування інформаційно-технологічної інфраструктури. Будь-яке порушення роботи IT-сервісів призводить до прямих фінансових втрат, зниження продуктивності персоналу та погіршення клієнтського досвіду, тому управління IT-інцидентами є одним із ключових процесів забезпечення стабільності бізнес-операцій. Першим етапом цього процесу є виявлення та реєстрація інцидентів, що у зрілих організаціях здебільшого ініціюється автоматичними сповіщеннями систем моніторингу.

Актуальність теми зумовлена тим, що системи інфраструктурного моніторингу на тисячу контрольованих об’єктів здатні генерувати десятки тисяч сповіщень на добу, значна частина яких є дублікатами або похідними від однієї кореневої причини. Без проміжної обробки кожне таке сповіщення перетворюється на потенційний інцидент, що спричиняє перевантаження операторів служби підтримки та ефект «alert fatigue». Готові спеціалізовані рішення для інтелектуальної агрегації та фільтрації подій є вартісними й орієнтованими на enterprise-сегмент, що робить їх недоступними для багатьох українських організацій середнього розміру. Це обумовлює потребу в розробленні доступного проміжного програмного шару між системою моніторингу та ITSM-платформою.

Метою кваліфікаційної роботи є розроблення програмної системи, що виступає проміжним шаром між системою моніторингу Microsoft System Center Operations Manager та ITSM-платформою і забезпечує автоматизовану реєстрацію, фільтрацію, агрегацію та кореляцію подій моніторингу для зниження операційного навантаження на службу підтримки.

Для досягнення поставленої мети у роботі розв’язано такі *задачі:*

1) Проаналізувати процеси управління IT-інцидентами та практики ITIL, дослідити ринок систем моніторингу й ITSM в Україні та методи агрегації, кореляції і фільтрації подій.

2) Сформулювати функціональні та нефункціональні вимоги до проєктованої системи.

3) Розробити мікросервісну архітектуру системи, модель предметної області та алгоритми обробки подій.

4) Спроєктувати структуру бази даних і реалізувати програмні компоненти системи.

5) Розгорнути систему в контейнерному середовищі та провести функціональне й навантажувальне тестування.

6) Виконати техніко-економічне обґрунтування ефективності розробленої системи.

Об'єктом дослідження є процеси реєстрації та обробки IT-інцидентів у корпоративних інформаційних системах.

Предметом дослідження є методи й засоби автоматизації цих процесів на основі агрегації та фільтрації подій моніторингу.

Методи дослідження: у роботі застосовано теоретичний аналіз і синтез наукової літератури та документації виробників, системний аналіз для декомпозиції системи на компоненти, порівняльний аналіз при обґрунтуванні вибору технологічного стеку, методи проєктування реляційних баз даних, методи розроблення розподіленого програмного забезпечення, експериментальні методи навантажувального тестування та методи техніко-економічного аналізу.

Практичне значення одержаних результатів полягає у створенні працездатної мікросервісної системи, розгорнутої в контейнерному середовищі та апробованої на матеріалах ТОВ «КЕРНЕЛ ДІДЖИТАЛ». Розроблене рішення може бути впроваджене в організаціях з інфраструктурою на основі Microsoft-стеку та тиражоване на підприємства з аналогічним профілем.

Результати роботи апробовані в рамках наукової конференції здобувачів вищої освіти за результатами науково-дослідної роботи у 2025-2026 рр.

Структура кваліфікаційної роботи логічно пов'язана з задачами досліджень і містить перелік умовних позначень, вступ, три розділи основної

частини, висновки, список використаних джерел. Загальний обсяг текстової частини кваліфікаційної роботи складає 57 сторінок формату А4. Вона містить 11 рисунків і 20 таблиць.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд процесів управління IT-інцидентами та практик ITIL

Сучасні організації, незалежно від галузі діяльності, у все більшій мірі залежать від безперебійного функціонування інформаційно-технологічної інфраструктури. Будь-яке порушення роботи IT-сервісів – від короткочасної недоступності корпоративної електронної пошти до повного простою бізнес-критичних систем – призводить до прямих фінансових втрат, зниження продуктивності персоналу та погіршення клієнтського досвіду [1]. У зв'язку з цим управління IT-інцидентами розглядається як один із ключових процесів забезпечення стабільності бізнес-операцій та якості обслуговування [26].

Згідно з бібліотекою найкращих практик ITIL (Information Technology Infrastructure Library), під IT-інцидентом розуміють будь-яку незаплановану подію, що не є елементом нормального функціонування IT-сервісу і яка впливає або може вплинути на роботу цього сервісу шляхом його переривання або зниження якості. Основною метою процесу управління інцидентами є якнайшвидше відновлення нормального рівня надання сервісу з мінімальним впливом на бізнес-діяльність організації. ITIL версії 4 розглядає управління інцидентами як одну з 34 ключових практик у складі системи Service Value System, тісно пов'язану з практиками управління проблемами, змінами, рівнем обслуговування та запитами на обслуговування [2].

Типовий процес обробки IT-інциденту в організації, що дотримується ITIL-практик, складається з послідовних етапів життєвого циклу [28]. На рисунку 1.1 представлено узагальнену схему такого життєвого циклу, що базується на стандарті ISO/IEC 27035 та рекомендаціях ITIL [27].

Першим і одним із найважливіших етапів є виявлення та реєстрація інциденту, що може ініціюватися як зверненням користувача до служби Service Desk, так і автоматичним сповіщенням від системи моніторингу. Саме на цьому

етапі особливо актуальною стає автоматизація – за статистикою, у великих організаціях понад 60 % інцидентів виявляються саме засобами моніторингу до моменту, коли їх помітять кінцеві користувачі. Після реєстрації інцидент проходить етап класифікації, на якому йому призначаються категорія, відповідальна команда та пріоритет.

Рисунок 1.1 – Життєвий цикл обробки IT-інциденту згідно з практиками ITIL

Пріоритет інциденту в ITIL традиційно обчислюється на основі двох вимірів – впливу (impact) на бізнес та терміновості (urgency) усунення. Загальна формула розрахунку пріоритету має вигляд:

$$P = f(I, U), \tag{1.1}$$

де P – пріоритет інциденту,

I – рівень впливу на бізнес-процеси (кількість залучених користувачів, критичність сервісу),

U – рівень терміновості (швидкість поширення наслідків).

На практиці функція $f(I, U)$ реалізується як матриця пріоритетів, приклад якої наведено в таблиці 1.1.

Таблиця 1.1 – Матриця визначення пріоритету IT-інциденту

Вплив \ Терміновість	Висока	Середня	Низька
Високий	P1 (Critical)	P2 (High)	P3 (Medium)
Середній	P2 (High)	P3 (Medium)	P4 (Low)
Низький	P3 (Medium)	P4 (Low)	P5 (Planning)

Кожному рівню пріоритету в межах угоди про рівень обслуговування (Service Level Agreement, SLA) відповідають цільові показники часу реакції та часу розв'язання інциденту [5]. Наприклад, для пріоритету P1 типове значення часу реакції в українських організаціях становить від 15 до 30 хвилин, а часу розв'язання – від 2 до 4 годин.

Ефективність процесу управління інцидентами в організації прийнято оцінювати за допомогою низки кількісних метрик, серед яких найвідомішими є MTTA (Mean Time To Acknowledge) – середній час підтвердження інциденту, та MTTR (Mean Time To Resolve) – середній час розв'язання інциденту. Метрика MTTR обчислюється за формулою:

$$MTTR = \frac{\sum_{i=1}^{n}\left(T_{resolve,i} - T_{detect,i}\right)}{n}, \tag{1.2}$$

де $T_{detect,i}$ – момент часу виявлення i-го інциденту,

$T_{resolve,i}$ – момент часу його розв'язання,

n – загальна кількість розв'язаних інцидентів за досліджуваний період.

Зниження значень MTTA та MTTR є одним із головних показників успішності впровадження автоматизованих систем обробки інцидентів [6].

В українській корпоративній практиці процеси управління інцидентами реалізуються переважно на базі спеціалізованих програмних рішень класу ITSM таких як BAS Service Desk, Creatio Service, BMC Helix, ServiceNow, Jira Service Management. Для об'єктів критичної інфраструктури України дотримання процесних практик ITIL та стандартів ISO/IEC 27035 є не лише рекомендацією, а й вимогою національного законодавства у сфері кібербезпеки [3,7], що додатково підкреслює актуальність автоматизації відповідних процесів.

Таким чином, управління IT-інцидентами є комплексним процесом, який охоплює весь життєвий цикл події – від її автоматичного виявлення системами моніторингу до закриття та аналізу. Ключовими напрямами підвищення ефективності цього процесу є скорочення часу реакції за рахунок автоматизованої реєстрації подій, зменшення обсягу ручної роботи операторів через агрегацію та фільтрацію надлишкових алертів, а також забезпечення прозорості виконання SLA-зобов'язань.

1.2 Аналіз існуючих систем моніторингу та обробки подій

Для забезпечення ефективного управління IT-інцидентами в умовах сучасних організацій застосовується два класи програмних рішень – системи інфраструктурного моніторингу, що відповідають за виявлення подій, та системи класу ITSM, що відповідають за реєстрацію та обробку інцидентів. На ринку України спостерігається стійкий тренд до застосування цих двох класів систем у зв'язці, при цьому ефективність процесу безпосередньо залежить від якості інтеграції між ними. Серед систем моніторингу, що набули поширення в Україні, можна виділити чотири основні рішення – Zabbix, Prometheus, Nagios та Microsoft System Center Operations Manager (SCOM). Zabbix є найпоширенішою системою моніторингу з відкритим кодом серед українських компаній завдяки централізованій архітектурі, можливості масштабування на тисячі вузлів та значній кількості готових інтеграцій. Prometheus, побудований на власній базі даних часових рядів (TSDB) та pull-моделі збору метрик, домінує в проєктах, що орієнтовані на cloud-native та контейнеризовані середовища [8]. Натомість у корпоративному сегменті, особливо у фінансовій галузі, телекомі та державному секторі, де переважає інфраструктура на основі продуктів Microsoft, базовою системою моніторингу залишається SCOM [9]. Узагальнене порівняння цих рішень за ключовими характеристиками наведено в таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика поширених систем моніторингу

Характеристика	SCOM	Zabbix	Prometheus	Nagios
Тип ліцензії	Комерційна	Open Source	Open Source (Apache 2.0)	Open Source / Enterprise
Модель збору даних	Push (агент)	Push/Pull	Pull (HTTP /metrics)	Push/Pull
Сховище даних	MS SQL Server	MySQL/PostgreSQL	TSDB (вбудована)	Текстові файли
Платформа	Windows-орієнтована	Кросплатформна	Кросплатформна	Linux-орієнтована
Корпоративна підтримка	Microsoft	Zabbix LLC	Спільнота	Nagios Enterprises
Типове застосування	Enterprise (Microsoft-стек)	Універсальне	Cloud-native, DevOps	Мережева інфраструктура

Оскільки система, що проєктується в межах цієї роботи, орієнтована на типове українське корпоративне середовище, доцільно розглянути SCOM детальніше. SCOM є частиною пакету Microsoft System Center та забезпечує моніторинг здоров’я, продуктивності та доступності серверів, сервісів і застосунків як у локальній інфраструктурі, так і в гібридних хмарних середовищах. Основною одиницею функціональності в SCOM є management group – група управління, що складається з одного або кількох серверів управління (Management Servers), оперативної бази даних (Operations Database) та сховища даних звітності (Data Warehouse), реалізованих на платформі Microsoft SQL Server. На контрольовані хости встановлюється агент, що збирає події з джерел операційної системи та передає їх на сервер управління через TCP-порт 5723. Логіка виявлення подій визначається management packs – наборами правил, моніторів та залежностей, специфічних для конкретного продукту або технології [10]. Узагальнена архітектура SCOM зображена на рисунку 1.2.

Другим класом систем, що використовуються українськими організаціями для обробки інцидентів, є ITSM-платформи. На вітчизняному ринку поширені як локалізовані рішення (BAS Service Desk, що базується на платформі 1С/BAS), так і світові продукти – Creatio Service, Jira Service Management від компанії Atlassian, ServiceNow та BMC Helix. BAS Service Desk популярний серед

середнього бізнесу завдяки повній локалізації та інтеграції з обліковими системами BAS. Jira Service Management та ServiceNow є лідерами Magic Quadrant у сегменті enterprise-ITSM та широко застосовуються у великих IT-компаніях, банках і телеком-операторах. Усі ці системи реалізують стандартний набір ITIL-практик – управління інцидентами, проблемами, змінами та конфігураціями.

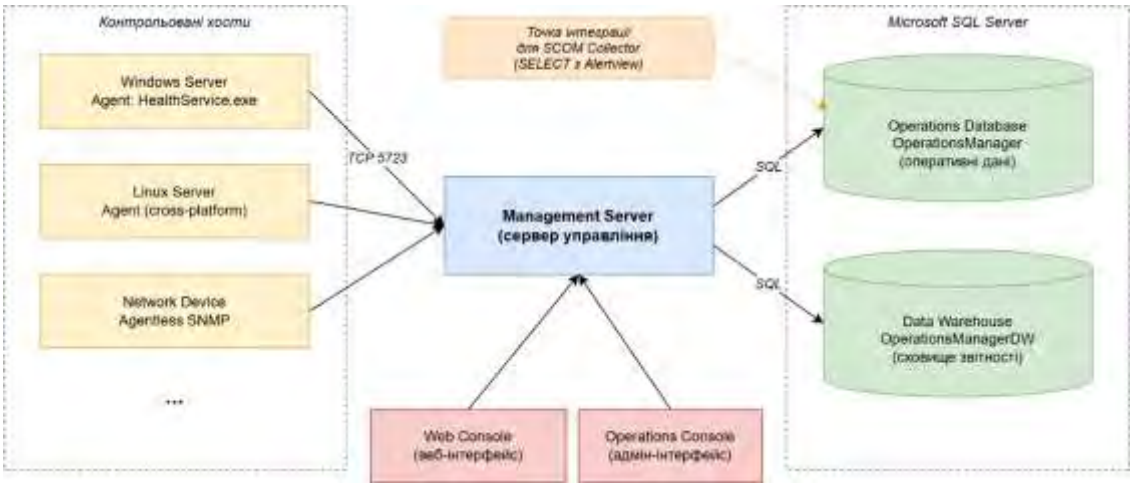

Рисунок 1.2 – Узагальнена архітектура SCOM

Ключовою проблемою, що виникає при спільному використанні систем моніторингу та ITSM-платформ, є проблема надлишку алертів (alert fatigue) – ситуації, коли система моніторингу генерує настільки велику кількість сповіщень, що оператори фізично не встигають їх опрацьовувати. Згідно з галузевими дослідженнями, у середній корпоративній SCOM-інсталяції на тисячу контрольованих об’єктів за добу може генеруватися кілька десятків тисяч алертів, при цьому до 80 % із них є дублікатами або похідними від однієї кореневої причини [9]. Кількість унікальних інцидентів, що мають бути зареєстровані в ITSM-системі, можна оцінити за формулою:

$$N_{inc} = N_{alerts} \cdot \left(1 - R_{filter}\right) \cdot \left(1 - R_{aggr}\right), \tag{1.3}$$

де N_{inc} – кількість зареєстрованих інцидентів,

N_{alerts} – загальна кількість алертів від системи моніторингу,

R_{filter} – коефіцієнт фільтрації шумових подій,

R_{aggr} – коефіцієнт агрегації пов'язаних подій.

Очевидно, що ефективність роботи всього процесу обробки інцидентів напряму залежить від значень R_{filter} та R_{aggr}, які визначаються якістю проміжного шару обробки подій між системою моніторингу та ITSM–платформою.

На практиці більшість з розглянутих ITSM-систем не мають вбудованих механізмів інтелектуальної кореляції подій з SCOM – інтеграція реалізується через вартісні рішення на кшталт ServiceNow Event Management чи BMC TrueSight. Це створює значну нішу для розроблення спеціалізованого проміжного шару, який би виконував агрегацію та фільтрацію подій моніторингу перед їх передачею в ITSM-систему. Саме таку задачу й вирішує система, що проєктується в межах цієї дипломної роботи.

1.3 Дослідження методів агрегації, кореляції та фільтрації подій

Як було показано в попередньому підрозділі, ключовою проблемою інтеграції систем моніторингу з ITSM-платформами є надлишок алертів, що генеруються джерелами на кшталт SCOM, Zabbix чи Prometheus. Без проміжної обробки кожна окрема подія перетворюється на потенційний інцидент, що призводить до надмірного навантаження операторів служби підтримки [29] та ефекту «alert fatigue». Для розв'язання цієї проблеми у промислових системах класу SIEM та Event Management застосовується багатоетапний конвеєр обробки подій, що складається з фільтрації, агрегації, нормалізації та кореляції [4].

Узагальнена схема такого конвеєра наведена на рисунку 1.3. На вхід конвеєра надходить сирий потік подій із одного або кількох джерел моніторингу, а на виході формується потік консолідованих інцидентів, що передаються до ITSM-системи для подальшої обробки операторами.

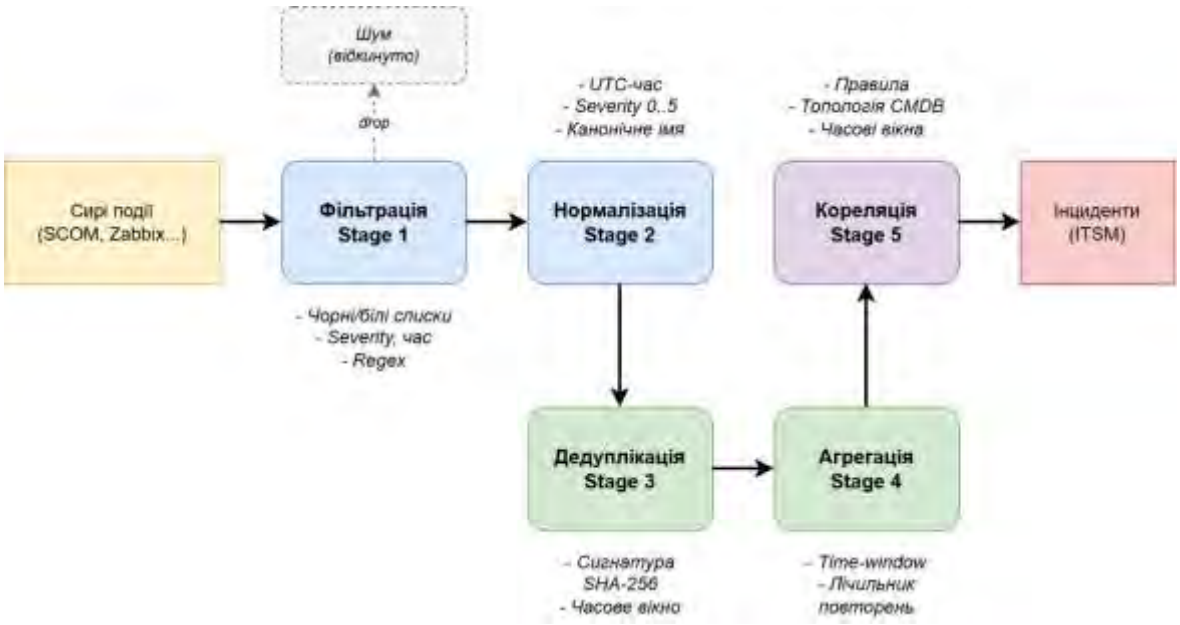

Рисунок 1.3 – Конвеєр обробки подій моніторингу

Фільтрація подій є першим і найпростішим етапом обробки, мета якого – відсікти події, що не несуть корисної інформації. Згідно з галузевою практикою, правильно налаштована фільтрація разом з агрегацією дозволяє зменшити обсяг інформації, що обробляється системою, у 5–10 разів [11]. Серед найпоширеніших підходів до фільтрації виділяють чорні та білі списки джерел подій, фільтри за рівнем критичності, часом виникнення або регулярними виразами над текстом події.

Дедуплікація та агрегація відповідають за усунення повторних і споріднених подій. Дедуплікація передбачає виявлення подій, що описують ту саму проблему, на основі їхніх атрибутивних ознак – джерела, типу, об'єкта моніторингу та опису. Класичний підхід базується на формуванні «сигнатури» події у вигляді хеш-значення від фіксованого набору полів, після чого нові події з тією самою сигнатурою відкидаються або інкрементують лічильник наявної події [30]. Агрегація узагальнює цей підхід та об'єднує події у часові вікна – так зване віконне групування (time-window aggregation). У межах вибраного вікна Δt усі події з однаковою сигнатурою об'єднуються в одну агреговану подію з лічильником повторень. Базова метрика подібності двох подій e_1 та e_2 обчислюється як зважена сума збігів за окремими атрибутами:

$$S(e_1, e_2) = \sum_{i=1}^{k} w_i \cdot \mathbb{1}[a_i(e_1) = a_i(e_2)], \quad (1.4)$$

де $S(e_1, e_2)$ – міра подібності двох подій,

k – кількість атрибутів, що враховуються,

w_i – ваговий коефіцієнт i-го атрибута,

$a_i(e)$ – значення i-го атрибута події e,

$\mathbb{1}[.]$ – індикаторна функція.

Якщо значення $S(e_1, e_2)$ перевищує визначений поріг, події вважаються дублікатами та об'єднуються.

Кореляція подій є найбільш інтелектуальним етапом обробки, що дозволяє виявляти причинно-наслідкові зв'язки між зовні різнорідними подіями та формувати з них єдиний інцидент. У сучасних SIEM і ITOM-системах прийнято виділяти чотири основні підходи до кореляції [30], узагальнення яких наведено в таблиці 1.3.

Таблиця 1.3 – Порівняння методів кореляції подій моніторингу

Метод	Принцип роботи	Переваги	Обмеження
Правила (Rule-based)	Експертні правила «якщо A та B протягом Δt – то інцидент C»	Прозорість, передбачуваність	Потребує ручного супроводу
Часові (Time-based)	Групування подій у межах часового вікна	Простота реалізації	Не враховує семантику
Топологічні (Topology-based)	Кореляція на основі CMDB та залежностей між CI	Висока точність	Залежить від актуальності CMDB
Статистичні / ML	Виявлення аномалій та патернів через ML-алгоритми	Адаптивність	«Чорна скринька», потрібні дані

Метод правил є найбільш поширеним у промислових системах через свою прозорість – оператор завжди може пояснити, чому певні події об'єдналися в інцидент. Часова кореляція є його окремим випадком та використовується як базовий механізм у багатьох конфігураціях. Топологічна кореляція базується на знанні архітектури IT-інфраструктури – так, падіння одного сервера баз даних

може спричинити каскад алертів від десятків залежних застосунків, які мають бути об’єднані в один кореневий інцидент [12]. Алгоритми машинного навчання та статистичного аналізу набувають дедалі більшої популярності в наукових дослідженнях [13], проте їхня практична реалізація залишається складною через потребу у великих обсягах розмічених даних та проблему пояснюваності рішень.

Загальну ефективність подібних систем прийнято оцінювати за допомогою коефіцієнта зменшення обсягу подій (event reduction ratio):

$$R_{red} = 1 - \frac{N_{inc}}{N_{events}}, \tag{1.5}$$

де R_{red} – коефіцієнт зменшення,

N_{events} – загальна кількість сирих подій на вході,

N_{inc} – кількість сформованих інцидентів на виході.

У промислових системах SIEM значення R_{red} зазвичай перебуває у межах 0,8-0,95, що відповідає 5–20-кратному зменшенню навантаження на оператора.

З огляду на специфіку задачі, що розв’язується в межах цієї роботи, доцільним є комбінування двох найпрактичніших підходів – фільтрації за правилами на основі атрибутивних характеристик подій SCOM та правил-кореляції з часовими вікнами з групуванням за ключовими атрибутами. Цей вибір забезпечує прозорість роботи системи, прийнятну якість зменшення обсягу подій без необхідності у попередньо розмічених навчальних вибірках, що було б критичним для застосування ML-підходів [14].

1.4 Постановка задачі та формування функціональних і нефункціональних вимог до системи

На основі проведеного в попередніх підрозділах аналізу можна сформулювати ключову проблему, що мотивує розробку системи в межах цієї

дипломної роботи. Типове корпоративне IT-середовище в Україні характеризується одночасним використанням систем інфраструктурного моніторингу та ITSM-платформ для обробки інцидентів. При цьому інтеграція між ними є або відсутньою, або реалізується найпростішим способом – кожен алерт системи моніторингу автоматично перетворюється на окремий інцидент в ITSM, що призводить до перевантаження операторів служби підтримки надлишковими, дубльованими або шумовими подіями. Готові спеціалізовані рішення для інтелектуальної агрегації та фільтрації подій (як-от ServiceNow Event Management чи BMC TrueSight) є вартісними та орієнтованими на enterprise-сегмент, що робить їх недоступними для багатьох українських організацій середнього розміру.

Виходячи з цього, метою роботи є розробка програмної системи, що виступає як проміжний шар між системою моніторингу SCOM та ITSM-платформою, та забезпечує автоматизовану реєстрацію, фільтрацію, агрегацію та кореляцію подій моніторингу для зниження операційного навантаження на службу підтримки. Система повинна бути побудована на сучасному технологічному стеку, мати мікросервісну архітектуру та забезпечувати горизонтальне масштабування за зростання обсягу подій.

Розробка вимог до програмного забезпечення є першим і одним із найважливіших етапів життєвого циклу проєкту, оскільки якісно сформульовані вимоги визначають як спільне розуміння продукту між зацікавленими сторонами, так і подальшу архітектуру та технологічні рішення [31]. Згідно з прийнятою в інженерії програмного забезпечення класифікацією, вимоги поділяються на функціональні та нефункціональні [15]. Загальний контекст функціонування проєктованої системи та її основні зовнішні актори зображені на рисунку 1.4.

Функціональні вимоги до системи, що наведені у таблиці 1.4, описують конкретні функції, які вона повинна реалізовувати, та формулюються за принципом «система повинна виконати [дію] за [умови]».

Рисунок 1.4. – Контекстна діаграма проєктованої системи

Таблиця 1.4 – Функціональні вимоги до системи

Код	Опис вимоги
FR-01	Система повинна періодично отримувати нові алерти з бази даних SCOM та публікувати їх у внутрішню чергу повідомлень
FR-02	Система повинна виконувати фільтрацію вхідних подій за заданими правилами (severity, source, monitoring object, час доби)
FR-03	Система повинна виявляти та об’єднувати дублікати подій у межах налаштовуваного часового вікна
FR-04	Система повинна виконувати кореляцію пов’язаних подій у єдиний інцидент згідно з правилами
FR-05	Система повинна автоматично призначати пріоритет інцидентам на основі впливу та терміновості
FR-06	Система повинна надавати REST API для роботи з інцидентами (CRUD-операції, фільтрація, пошук)
FR-07	Система повинна забезпечувати веб-інтерфейс для перегляду, фільтрації та зміни статусу інцидентів
FR-08	Система повинна підтримувати автентифікацію користувачів та розмежування прав доступу за ролями
FR-09	Система повинна формувати звіти та статистичні показники (MTTR, MTTA, динаміка інцидентів)
FR-10	Система повинна вести аудит-журнал усіх змін стану інцидентів

Нефункціональні вимоги визначають якісні характеристики системи – продуктивність, надійність, масштабованість, безпеку та зручність використання [16, 32]. Їх перелік наведено в таблиці 1.5.

Таблиця 1.5 – Нефункціональні вимоги до системи

Код	Характеристика	Зміст вимоги
NFR-01	Продуктивність	Затримка обробки події від моменту її надходження до збереження в базі даних – не більше 5 секунд за нормального навантаження
NFR-02	Пропускна здатність	Підтримка обробки не менше 100 подій на секунду на одному екземплярі сервісу обробки
NFR-03	Надійність	Збереження подій у разі тимчасової недоступності сервісу обробки (буферизація в черзі) без втрат повідомлень
NFR-04	Доступність	Цільовий показник доступності (uptime) – не менше 99 % на місяць
NFR-05	Масштабованість	Можливість горизонтального масштабування сервісу обробки шляхом запуску додаткових екземплярів
NFR-06	Безпека	Шифрування з'єднань (TLS), автентифікація через JWT, розмежування прав доступу за ролями (RBAC)
NFR-07	Зручність використання	Веб-інтерфейс адаптований для роботи на типових моніторах (роздільна здатність 1366×768 та вище)
NFR-08	Контейнеризація	Усі компоненти системи постачаються у вигляді Docker-образів та розгортаються через Docker Compose

Ключовою кількісною метрикою ефективності системи є коефіцієнт зменшення обсягу подій R_{red}, введений у попередньому підрозділі. Цільовий критерій якості для проєктованої системи можна сформулювати як умову одночасного досягнення прийнятного рівня скорочення обсягу подій та збереження повноти виявлення інцидентів:

$$F = \alpha \cdot R_{red} + \beta \cdot R_{recall} \rightarrow max, \quad (1.6)$$

де F – інтегральний критерій якості,

R_{red} – коефіцієнт зменшення обсягу подій,

R_{recall} – повнота виявлення реальних інцидентів (recall),

α та β – вагові коефіцієнти, що визначають баланс між скороченням навантаження на оператора та повнотою виявлення реальних проблем (за замовчуванням $\alpha = \beta = 0{,}5$).

Цільові значення обох метрик визначено як $R_{red} \geq 0{,}7$ та $R_{recall} \geq 0{,}95$, що відповідає кращим практикам сучасних SIEM і ITOM-систем [12].

Окрім перелічених вимог, до системи висуваються технологічні обмеження, що випливають із прийнятої в розділі 2 архітектури – використання мови Python та фреймворку FastAPI для серверних мікросервісів, СКБД PostgreSQL для зберігання даних, RabbitMQ як брокера повідомлень, а також React і Ant Design для веб-інтерфейсу. Ці обмеження є обґрунтованими з точки зору наявності зрілих бібліотек, кваліфікованих фахівців на ринку праці України та вартості володіння системою. Сформульовані функціональні та нефункціональні вимоги виступатимуть основою для проєктування архітектури системи, моделі предметної області та алгоритмів обробки подій у наступному розділі.

РОЗДІЛ 2
ПРОЕКТУВАННЯ МІКРОСЕРВІСНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ ОБРОБКИ ІТ-ІНЦИДЕНТІВ

2.1 Розроблення мікросервісної архітектури системи та схеми взаємодії

Сформульовані в попередньому розділі функціональні та нефункціональні вимоги передбачають побудову системи, що здатна приймати потенційно високий потік подій від системи моніторингу, виконувати нетривіальну логіку їх обробки, надавати веб-інтерфейс операторам та масштабуватися горизонтально за зростання навантаження. Класична монолітна архітектура, за якої всі компоненти системи виконуються в межах одного процесу, погано підходить для таких задач – вона ускладнює незалежне масштабування, підвищує ризики каскадних відмов та обмежує гнучкість у виборі технологій для окремих компонентів. Натомість мікросервісна архітектура дозволяє розкласти систему на набір невеликих, слабко зв’язаних сервісів, кожен з яких відповідає за одну чітко визначену бізнес-функцію та може розроблятися, розгортатися й масштабуватися незалежно від інших [18].

Базовими принципами, на яких будується мікросервісна архітектура проєктованої системи, є декомпозиція за бізнес-функціями (Single Responsibility Principle на рівні сервісу), слабка зв’язаність (loose coupling) через використання чітко визначених API та асинхронних повідомлень, незалежне розгортання компонентів у вигляді Docker-контейнерів, а також ізоляція станів – кожен сервіс володіє власними даними і не звертається до баз даних інших сервісів напряму [19]. Зокрема, незалежність сервісів забезпечує ізоляцію збоїв – відмова одного компонента не призводить до зупинки всієї системи, що особливо критично в задачах обробки подій моніторингу [33].

З огляду на ці принципи система декомпозована на чотири основні мікросервіси та два інфраструктурні компоненти. Узагальнена архітектура системи зображена на рисунку 2.1.

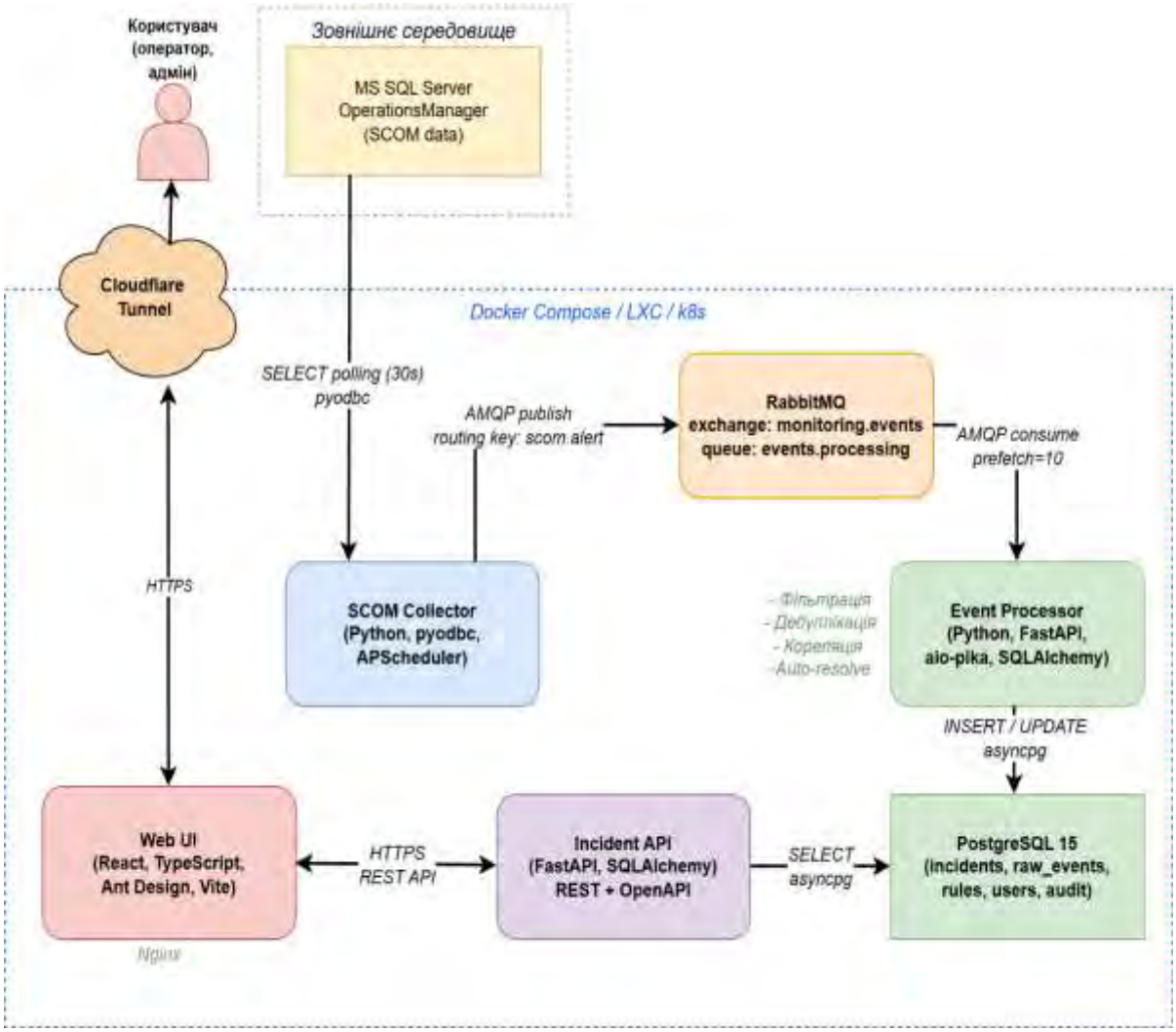

Рисунок 2.1. – Мікросервісна архітектура системи автоматизації обробки IT-інцидентів

Розподіл відповідальності між компонентами наведено в таблиці 2.1. Кожен мікросервіс має чітко окреслену зону відповідальності та взаємодіє з рештою системи лише через визначені інтерфейси, що відповідає принципу Single Responsibility Principle.

Важливою часиною є вибір механізмів міжсервісної взаємодії, оскільки в мікросервісних системах саме комунікаційний шар часто стає найвразливішим місцем. У проєктованій системі застосовуються два паттерни взаємодії – синхронний і асинхронний. Синхронна взаємодія між Web UI та Incident API реалізована через REST API поверх HTTPS – це забезпечує негайний відгук на дії користувача та відповідає очікуванням типового веб-застосунку. Асинхронна

взаємодія між SCOM Collector та Event Processor реалізована через брокер повідомлень RabbitMQ з використанням протоколу AMQP 0.9.1 [20]. Такий підхід забезпечує буферизацію подій у разі тимчасової недоступності Event Processor, гарантовану доставку через механізм підтверджень, а також можливість горизонтального масштабування Processor шляхом запуску кількох екземплярів-споживачів, що паралельно обробляють події з однієї черги [20]. Топологія RabbitMQ передбачає одну точку обміну (exchange) типу direct [35] з ім'ям monitoring.events та одну робочу чергу events.processing, до якої прив'язується Processor.

Таблиця 2.1 – Опис компонентів мікросервісної архітектури

Компонент	Технологія	Відповідальність
SCOM Collector Service	Python, pyodbc, APScheduler	Періодичне опитування БД SCOM, виявлення нових алертів, серіалізація та публікація у чергу
RabbitMQ	RabbitMQ 3.12+	Брокер повідомлень, буферизація подій, гарантована доставка
Event Processor Service	Python, FastAPI, aio-pika	Споживання подій із черги, фільтрація, дедуплікація, агрегація, кореляція, формування інцидентів
Incident API Service	Python, FastAPI, SQLAlchemy	REST API для роботи з інцидентами, автентифікація, авторизація, формування статистики
PostgreSQL	PostgreSQL 15+	Зберігання інцидентів, правил обробки, аудит-журналу, користувачів
Web UI	React, TypeScript, Ant Design	Графічний інтерфейс для операторів – перегляд, фільтрація та зміна стану інцидентів

Узагальнений сценарій проходження події системою описується наступною послідовністю кроків: SCOM Collector раз на 30 секунд робить SELECT-запит до представлення OperationsManager.dbo.Alertview за умовою TimeRaised > last_processed_timestamp, отримує нові алерти, серіалізує їх у формат JSON та публікує до exchange monitoring.events. RabbitMQ розподіляє повідомлення в чергу events.processing, звідки їх споживає Event Processor. Processor виконує фільтрацію, дедуплікацію та кореляцію, після чого записує

сформований або оновлений інцидент у PostgreSQL. Оператор через Web UI робить HTTP-запити до Incident API, який, своєю чергою, читає дані з PostgreSQL. Така архітектура дозволяє повністю розв'язати потік запису (від моніторингу) і потік читання (від користувача), що відповідає принципу CQRS на рівні логіки взаємодії.

Для оцінки пропускної здатності системи в цілому використовується наступний розрахунок. Загальна пропускна здатність обмежується найповільнішим компонентом у конвеєрі обробки [21]:

$$T_{system} = \min\left(T_{collector}, T_{queue}, T_{processor} \cdot N_{instances}\right), \qquad (2.1)$$

де T_{system} – пропускна здатність системи в подіях за секунду,

$T_{collector}$ – пропускна здатність сервісу-полера,

T_{queue} – пропускна здатність брокера RabbitMQ (для типової конфігурації досягає кількох тисяч повідомлень на секунду на один вузол),

$T_{processor}$ – пропускна здатність одного екземпляра Event Processor,

$N_{instances}$ – кількість запущених екземплярів Processor.

Як правило, найповільнішим компонентом є Processor через нетривіальну логіку обробки, тому горизонтальне масштабування саме цього сервісу є основним механізмом підвищення продуктивності.

Запропонована архітектура задовольняє всім нефункціональним вимогам, сформульованим у підрозділі 1.4 – масштабованість (NFR-05) досягається через незалежне розгортання та горизонтальне масштабування Event Processor, надійність (NFR-03) – через буферизацію в RabbitMQ та збереження повідомлень на диск, відмовостійкість – через ізоляцію збоїв між сервісами, контейнеризація (NFR-08) – через Docker-образи кожного компонента та оркестрацію через Docker Compose. Обґрунтування конкретного вибору технологічних засобів реалізації кожного компонента наведено в наступному підрозділі.

2.2 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку є одним із ключових проєктних рішень, що визначає не лише швидкість і вартість розробки, а й довгострокову підтримуваність системи, можливості масштабування та доступність кваліфікованих кадрів на ринку праці. Для систематичного обґрунтування вибору технологій застосовується підхід зваженої оцінки альтернатив за визначеним набором критеріїв, що дозволяє формалізувати прийняття рішень у проєктній діяльності. Інтегральна оцінка технології в межах такого підходу обчислюється за формулою:

$$S_{tech} = \sum_{i=1}^{n} w_i \cdot s_i, \tag{2.2}$$

де S_{tech} – інтегральна оцінка технології,

s_i – оцінка технології за i-м критерієм за шкалою від 1 до 5,

w_i – ваговий коефіцієнт i-го критерію ($\sum w_i = 1$),

n – кількість критеріїв.

Як критерії оцінки в межах цієї роботи розглянуто продуктивність, простоту розробки, наявність документації та спільноти, зрілість екосистеми бібліотек і доступність фахівців на ринку праці України. Обраний технологічний стек зображений на рис. 2.2.

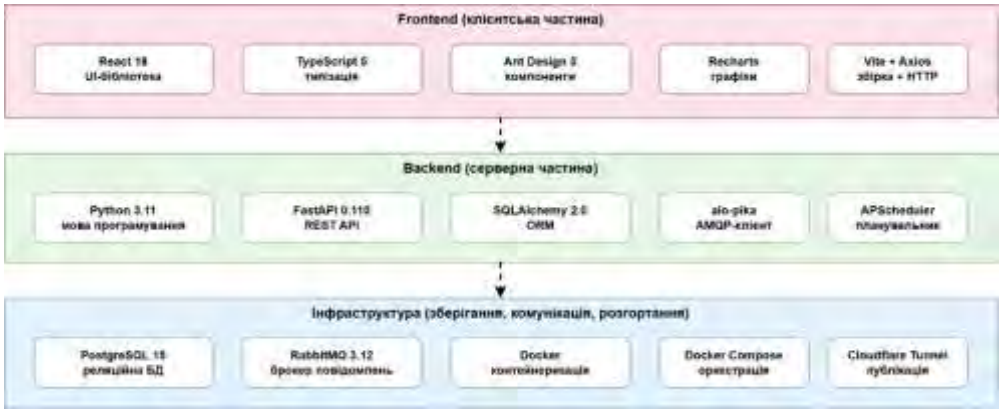

Рисунок 2.2 – Технологічний стек проєктованої системи

Для серверної частини обрана мова Python 3.11+, оскільки вона домінує в задачах автоматизації, обробки даних та DevOps-сценаріях, що відповідає предметній області проєкту [22]. Серед популярних Python-фреймворків (Django, Flask та FastAPI) оптимальним вибором є FastAPI [17] завдяки нативній підтримці асинхронності, високій продуктивності, автоматичній генерації OpenAPI-документації та валідації даних через Pydantic. Порівняння цих фреймворків наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння Python-фреймворків для розробки REST API

Критерій	Django	Flask	FastAPI
Тип фреймворку	Full-stack	Мікрофреймворк	API-орієнтований
Підтримка async	Часткова	Через розширення	Нативна
Продуктивність (RPS)	Середня	Середня	Висока
Автогенерація OpenAPI	Через DRF	Ні	Так (вбудовано)
Типізація / Pydantic	Ні	Ні	Так (вбудовано)
Поріг входу	Високий	Низький	Низький

За результатами порівняння FastAPI випереджає Flask за продуктивністю приблизно вдвічі, а Django REST Framework – у 4–5 разів, що відповідає результатам незалежних бенчмарків.

Для асинхронної взаємодії між сервісом-полером та сервісом обробки подій розглядалися три кандидати – RabbitMQ, Apache Kafka та Redis Pub/Sub [34]. Узагальнене порівняння цих рішень наведено в таблиці 2.3.

Таблиця 2.3 – Порівняння брокерів повідомлень для міжсервісної взаємодії

Критерій	RabbitMQ	Apache Kafka	Redis Pub/Sub
Модель	Класичні черги	Лог-орієнтована	In-memory pub/sub
Гарантії доставки	At-least-once, exactly-once	At-least-once	At-most-once
Підтримка протоколів	AMQP, MQTT, STOMP	Власний протокол	RESP
Складність налаштування	Низька	Висока	Дуже низька
Типова пропускна здатність	До 20 000 повід./с	Сотні тисяч повід./с	До 100 000 повід./с
Зрілість, спільнота	Висока	Висока	Висока

Обрано RabbitMQ як збалансоване рішення, що поєднує надійні гарантії доставки повідомлень, гнучку маршрутизацію через механізм exchange та просте розгортання. Apache Kafka, попри значно вищу пропускну здатність, є надлишковим для задач масштабу даної роботи та потребує суттєво складнішої експлуатації. Redis Pub/Sub натомість не забезпечує персистентності повідомлень, що неприйнятно для системи обробки інцидентів, де втрата подій є критичною [36]. RabbitMQ забезпечує гарантовану доставку через механізм підтверджень та збереження повідомлень на диск – необхідні умови для коректної обробки потоку алертів від SCOM.

Для зберігання інцидентів, правил обробки та даних користувачів обрана PostgreSQL 15 – об'єктно-реляційна СКБД з відкритим вихідним кодом і повною ACID-сумісністю [23]. Серед основних переваг PostgreSQL це підтримка типу даних JSONB для зберігання структурованих даних події з можливістю індексації за окремими полями та вбудована підтримка повнотекстового пошуку, що корисно для пошуку за описами інцидентів. Альтернативи – MySQL та MongoDB – поступаються PostgreSQL за гнучкістю індексації JSON-даних та повнотою реалізації SQL-стандарту відповідно. Для роботи з PostgreSQL із Python-сервісів використовується ORM-бібліотека SQLAlchemy 2.0, а для управління схемою БД – інструмент міграцій Alembic.

Користувацький інтерфейс реалізовано на бібліотеці React 18 з типізацією через TypeScript та компонентною бібліотекою Ant Design. React домінує на ринку фронтенд-розробки в Україні, що забезпечує доступність кваліфікованих фахівців. TypeScript додає статичну типізацію, що зменшує кількість помилок у production. Ant Design це корпоративна UI-бібліотека з готовими компонентами (таблиці, форми, діаграми, модальні вікна), що відповідає задачі побудови dashboard-інтерфейсу оператора. Для виконання HTTP запитів до Incident API використовується бібліотека Axios, для управління станом – вбудовані хуки React.

Усі компоненти системи постачаються у вигляді Docker-образів, а їх спільне розгортання описане в файлі docker-compose.yml. Це забезпечує

відтворюваність середовища розробки та спрощує демонстрацію роботи системи. У майбутньому система може бути перенесена на Kubernetes без значних змін у коді – Docker-образи готові до такої міграції.

2.3 Проєктування структури бази даних PostgreSQL

Якісне проєктування структури бази даних є фундаментом надійної та продуктивної інформаційної системи. Помилки, допущені на цьому етапі, важко й вартісно виправляти на пізніших стадіях розробки, оскільки вони впливають як на код сервісів, що працюють з даними, так і на історичні дані, накопичені в експлуатації. Класичний підхід до проєктування БД передбачає проходження трьох послідовних етапів – концептуального (виокремлення сутностей предметної області та зв’язків між ними), логічного (трансформація концептуальної моделі в реляційну схему з урахуванням нормалізації) та фізичного (визначення типів даних, індексів, обмежень для конкретної СКБД) [24].

На концептуальному рівні предметна область проєктованої системи охоплює такі ключові сутності: «Сира подія» (Raw Event) – алерт, що надходить від SCOM; «Інцидент» (Incident) – консолідована сутність, що формується внаслідок обробки однієї або кількох подій; «Правило обробки» (Correlation Rule) – параметризований опис логіки фільтрації, дедуплікації або кореляції; «Конфігураційна одиниця» (CI Item) – об’єкт інфраструктури, на якому виникла подія; «Користувач» (User) та «Роль» (Role) – суб’єкти системи з різними правами доступу; «Запис історії» (Incident History) – фіксація змін стану інциденту для цілей аудиту. Узагальнена ER-діаграма, що відображає виокремлені сутності та зв’язки між ними, наведена на рисунку 2.3.

На логічному рівні концептуальна модель була трансформована в реляційну схему із застосуванням процедури нормалізації до третьої нормальної форми (3НФ). Нормалізація забезпечує усунення надмірності даних та гарантує

їхню цілісність, що особливо важливо в системі, де щодоби може реєструватися велика кількість подій.

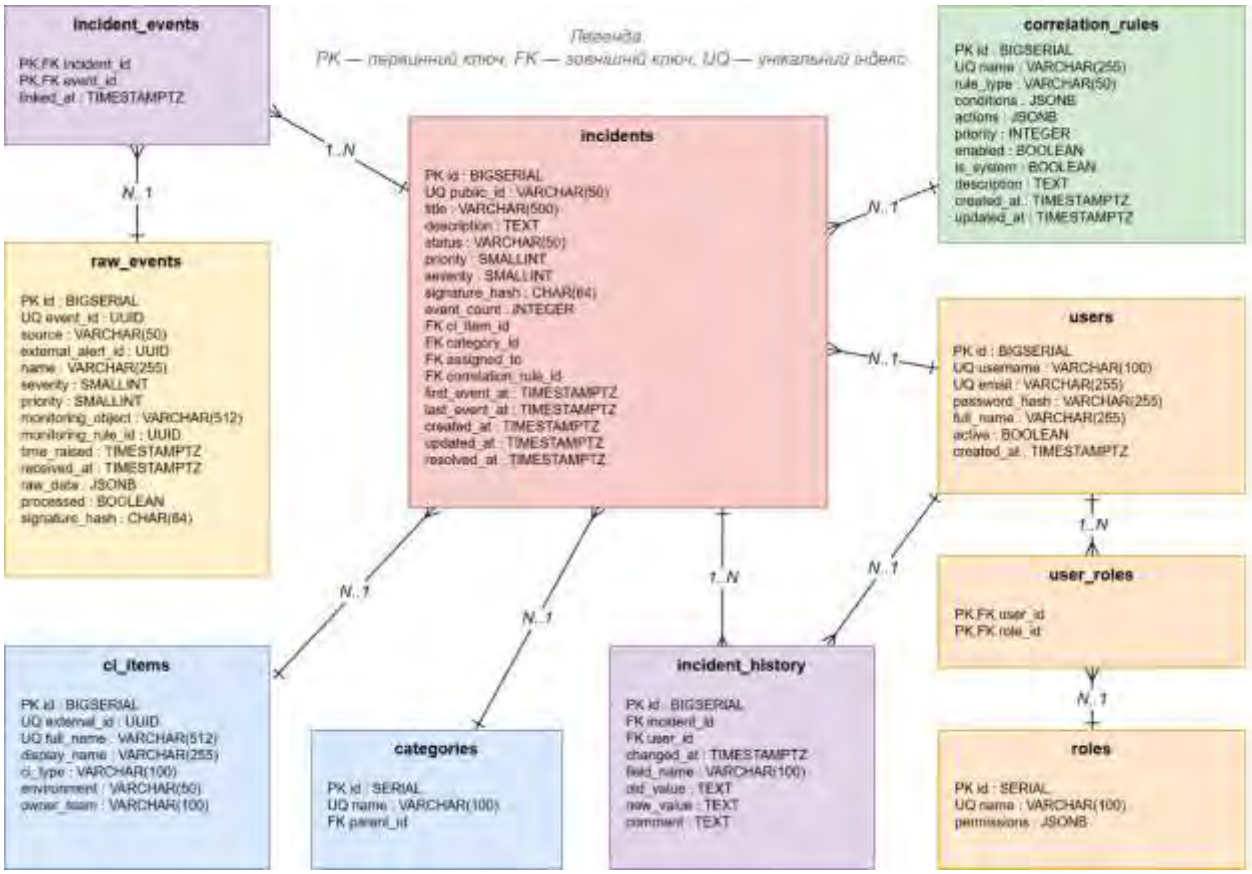

Рисунок 2.3. – ER-діаграма бази даних системи обробки IT-інцидентів

Формальний критерій належності відношення до 3НФ можна записати наступним чином – для будь-якого нетривіального функціонального залежного $X \rightarrow A$ має виконуватись:

$$(X \supseteq K) \vee (A \in K), \tag{2.3}$$

де K – будь-який потенційний ключ відношення,

X – детермінант,

A – атрибут, що залежить від детермінанта.

Виконання цієї умови означає, що кожен неключовий атрибут залежить лише від первинного ключа й не має транзитивних залежностей від інших

неключових атрибутів. На практиці це призвело до винесення довідникових даних (категорії, статуси, рівні критичності) в окремі таблиці та використання посилань через зовнішні ключі.

Обрано використання типу даних JSONB для полів raw_data (вихідні дані події), conditions та actions (умови та дії правил кореляції). Тип JSONB у PostgreSQL зберігає JSON у бінарному вигляді з можливістю індексації через GIN-індекс і виконання запитів за окремими полями вкладеної структури. Це дозволяє зберігати у БД події з SCOM зі всіма їхніми вихідними атрибутами, водночас гнучко адаптуючись до можливих змін у структурі даних з боку SCOM без потреби міграції схеми [25].

Перелік основних таблиць проєктованої бази даних та їх призначення наведено в таблиці 2.4.

Таблиця 2.4 – Основні таблиці бази даних

Таблиця	Призначення
raw_events	Зберігання сирих подій, отриманих із SCOM; містить поля id, source, severity, monitoring_object, raw_data (JSONB), timestamp, processed
incidents	Інциденти, сформовані сервісом обробки; містить id, title, description, status, priority, severity, ci_item_id, assigned_to, created_at, resolved_at, sla_deadline
incident_events	Зв’язкова таблиця many-to-many між інцидентами та подіями (incident_id, event_id)
correlation_rules	Правила обробки подій: id, name, rule_type, conditions (JSONB), actions (JSONB), priority, enabled
ci_items	Конфігураційні одиниці інфраструктури: id, name, type, environment, owner_team
users	Користувачі системи: id, username, email, password_hash, active, created_at
roles	Ролі для RBAC: id, name, permissions (JSONB)
user_roles	Зв’язкова таблиця між користувачами та ролями
incident_history	Журнал змін інцидентів: id, incident_id, user_id, changed_at, field_name, old_value, new_value
categories	Довідник категорій інцидентів: id, name, parent_id

Для забезпечення прийнятної продуктивності пошукових запитів на таблиці накладено набір індексів, що відповідають типовим сценаріям доступу. Ключові індекси наведено в таблиці 2.5.

Таблиця 2.5 – Індекси для забезпечення продуктивності запитів

Таблиця	Індекс	Тип	Призначення
raw_events	idx_events_timestamp	B-tree	Швидкий пошук подій за часовим діапазоном
raw_events	idx_events_processed	B-tree	Вибірка необроблених подій
raw_events	idx_events_raw_data	GIN	Пошук за полями JSONB-структури події
incidents	idx_incidents_status	B-tree	Фільтрація відкритих/закритих інцидентів
incidents	idx_incidents_priority	B-tree	Сортування за пріоритетом
incidents	idx_incidents_assigned	B-tree	Перегляд інцидентів конкретного оператора
incidents	idx_incidents_ci	B-tree	Пошук інцидентів за конфігураційною одиницею
incident_history	idx_history_incident	B-tree	Завантаження історії конкретного інциденту

GIN-індекси на JSONB-полях створюються з оператором класу jsonb_path_ops, оскільки він забезпечує менший розмір індексу та вищу продуктивність для типових сценаріїв пошуку за вкладеними ключами порівняно з класом jsonb_ops за замовчуванням [25]. На таблиці incidents додатково створено складений індекс (status, priority, created_at), що оптимізує запит головної сторінки UI зі списком активних інцидентів, відсортованих за пріоритетом і часом.

Цілісність даних забезпечується низкою обмежень – NOT NULL для обов’язкових полів (наприклад, severity у raw_events), UNIQUE для природних ключів (наприклад, username у users), CHECK для обмежень діапазонів (наприклад, priority IN (1,2,3,4,5)), а також FOREIGN KEY для всіх посилальних зв’язків з опцією ON DELETE RESTRICT для довідників та ON DELETE CASCADE для зв’язкових таблиць.

Управління схемою БД реалізовано через інструмент Alembic, що забезпечує версіоновані міграції та можливість зворотного відкату при необхідності. Усі зміни схеми проходять через міграційні скрипти, що зберігаються разом із кодом сервісів у системі контролю версій, що відповідає сучасним практикам Database DevOps. Спроєктована структура задовольняє нефункціональним вимогам до продуктивності (NFR-01) та цілісності,

сформульованим у підрозділі 1.4, та слугує основою для реалізації алгоритмів обробки подій, опис яких наводиться у наступному підрозділі.

2.4 Розроблення алгоритмів фільтрації, кореляції та агрегації подій

Алгоритми обробки подій є ядром проєктованої системи та визначають, наскільки ефективно вона зменшує навантаження на оператора служби підтримки. На вхід цих алгоритмів надходить потік сирих подій із SCOM, що споживається сервісом обробки з черги RabbitMQ, а на виході формується значно менший потік консолідованих інцидентів, які зберігаються в PostgreSQL та надалі відображаються в інтерфейсі оператора. Проєктований конвеєр обробки складається з чотирьох послідовних етапів – фільтрації, нормалізації, дедуплікації з агрегацією та кореляції – що відповідає узагальненому підходу промислових SIEM-систем, розглянутому в підрозділі 1.3. Послідовність обробки кожної події в межах сервіс-контексту наведена на рисунку 2.4.

Етап фільтрації реалізований за принципом правил-предикатів. Кожне правило фільтрації описується у форматі JSONB та зберігається в таблиці correlation_rules з типом filter. Правило містить набір умов, що накладаються на атрибути події (severity, source, monitoring_object, час доби), та дію – accept або drop. Подія проходить етап фільтрації, якщо вона задовольняє хоча б одне правило з дією accept, і не задовольняє жодного правила з дією drop. Правила застосовуються в порядку зменшення пріоритету. На цьому етапі відсікаються події з низьким рівнем критичності (Information), події під час планових технологічних вікон та події від тестових середовищ.

Алгоритм виконує пошук у таблиці incidents активного запису зі статусом Open або In Progress, що має ту саму сигнатуру $H(e)$ та був створений у межах часового вікна Δt (за замовчуванням 15 хвилин). Якщо такий інцидент знайдено, нова подія прив’язується до нього через таблицю incident_events та інкрементує

лічильник event_count. Якщо ж відповідного інциденту немає, починається наступний етап обробки.

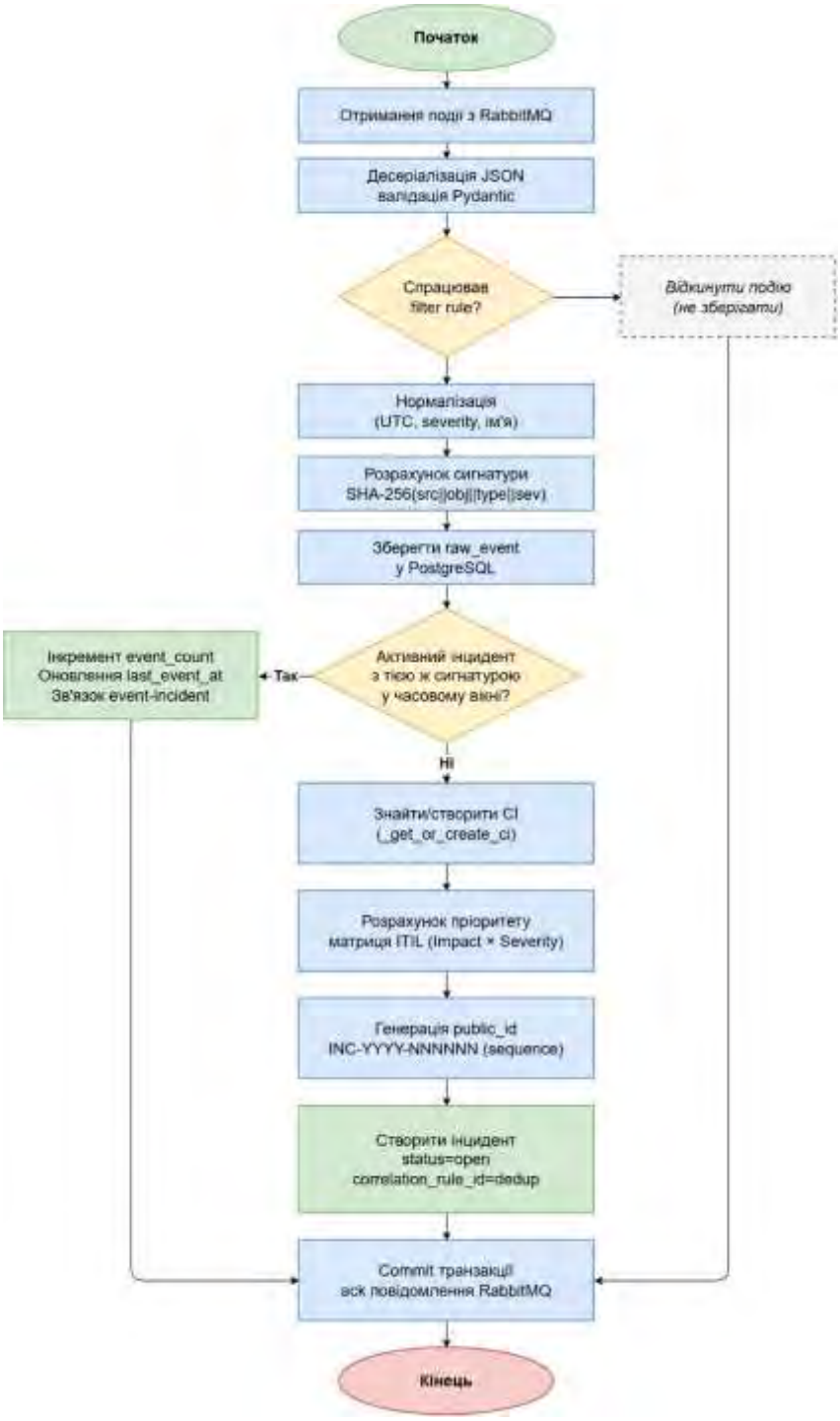

Рисунок 2.4 – Блок-схема алгоритму обробки події системи

Етап дедуплікації та агрегації реалізує об’єднання повторюваних подій у межах налаштовуваного часового вікна. Ключову роль на цьому етапі відіграє

функція обчислення сигнатури події $H(e)$ – хеш-значення від конкатенації фіксованого набору атрибутів події, що однозначно ідентифікують її як «той самий тип проблеми» [11]:

$$H(e) = \text{SHA256}(src \parallel obj \parallel type \parallel sev), \quad (2.4)$$

де *src* – джерело події,

obj – об’єкт моніторингу (наприклад, ім’я сервера або сервісу),

type – тип події (rule_id у термінах SCOM),

sev – рівень критичності,

|| – операція конкатенації рядків.

Етап кореляції виконує об’єднання подій, що мають різні сигнатури, але відповідають одній кореневій причині. Правила кореляції описуються в таблиці correlation_rules з типом correlation і містять набір умов відповідності події та посилання на правило-матч. Прикладами правил кореляції в проєктованій системі є об’єднання усіх подій від застосунків, що залежать від певного сервера БД, у разі його падіння, або об’єднання подій від багатьох вузлів, що належать до одного кластера. Перелік прикладів правил наведено в таблиці 2.6.

Таблиця 2.6 – Приклади правил кореляції подій

Назва правила	Тип	Умови	Дія
FILTER-INFO	filter	severity = 0 (Information)	drop
FILTER-MAINTENANCE	filter	час події ∈ maintenance_window	drop
DEDUP-DEFAULT	dedup	сигнатура збігається у вікні 15 хв	aggregate
CORR-DB-DOWN	correlation	source = "SQL Server", severity ≥ 4	match dependent apps протягом 5 хв
CORR-NETWORK	correlation	type = "Network unreachable"	об’єднати всі події з тієї ж VLAN

Розрахунок пріоритету. Для кожного нового інциденту обчислюється пріоритет на основі впливу та терміновості, як це було сформульовано в підрозділі 1.1. У проєктованій системі вплив I визначається на основі типу та

критичності конфігураційної одиниці, на якій виникла подія (production \ staging \ dev), а терміновість U – на основі рівня severity події та швидкості накопичення подібних подій. Пріоритет обчислюється за зваженою формулою:

$$P = \lceil w_I \cdot I + w_U \cdot U \rceil, \quad (2.5)$$

де P – числовий пріоритет від 1 (найвищий) до 5 (найнижчий),

I та U – нормовані значення впливу і терміновості за шкалою від 1 до 5,

w_I та w_U – вагові коефіцієнти (за замовчуванням w_I = 0,6, w_U= 0,4),

$\lceil \cdot \rceil$ – операція округлення вгору.

Така формула забезпечує детерміновану та прозору логіку призначення пріоритету, що відповідає рекомендаціям ITIL. Окрім перелічених алгоритмів, сервіс обробки реалізує механізм автоматичного закриття інцидентів – якщо протягом контрольного часового вікна (наприклад, 30 хвилин) не надходить жодної нової події з тією самою сигнатурою, інцидент автоматично переводиться у стан Auto-Resolved. Це дозволяє розвантажити операторів від ручного закриття типових короткочасних інцидентів і відповідає шаблону roll-back у моделі ITSM-процесів.

Усі правила обробки зберігаються у вигляді JSONB-структур у таблиці correlation_rules, що дозволяє адмінстраторам системи додавати, редагувати та вимикати правила через адміністративний інтерфейс без перезапуску сервісу. Алгоритми реалізовані з урахуванням принципу ідемпотентності – повторна обробка тієї самої події не призводить до створення дублікатів інцидентів, що є критично важливим у разі відновлення після збоїв і повторної доставки повідомлень з RabbitMQ [20]. Сукупно описані алгоритми дозволяють досягти цільових показників $R_{red} \geq 0{,}7$ та $R_{recall} \geq 0{,}95$, сформульованих у підрозділі 1.4.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Реалізація сервісу збору подій із SCOM та інтеграції через чергу повідомлень RabbitMQ

Сервіс збору подій (SCOM Collector) є першою ланкою конвеєра обробки інформації від системи моніторингу. Його основним завданням є періодичне опитування представлення dbo.Alertview бази даних System Center Operations Manager, виявлення нових алертів від попереднього циклу опитування та публікація відповідних повідомлень у брокер RabbitMQ для подальшої асинхронної обробки. Реалізацію здійснено мовою Python 3.11 з використанням бібліотек pyodbc для роботи з MS SQL Server, aio-pika для асинхронної взаємодії з RabbitMQ та APScheduler для періодичного запуску циклу опитування.

Архітектурно сервіс реалізує патерн «періодичний пуллер» (Polling Pattern) з підтримкою інкрементального читання. Замість того, щоб щоразу читати всю таблицю алертів, Collector зберігає на диску стан (state.json) – мітку часу TimeRaised останнього успішно обробленого алерту та його ідентифікатор AlertId. Це дозволяє після перезапуску продовжити роботу з того ж місця, де було зупинено попередній цикл, без втрати чи дублювання повідомлень. Файл стану зберігається у Docker-volume, що залишається доступним після перестворення контейнера.

Період опитування становить 30 секунд встановлює компроміс між затримкою реакції та навантаженням на сервер SCOM. Алгоритм одного циклу опитування подано на рисунку 3.1. Кожен прочитаний рядок з dbo.Alertview перетворюється на повідомлення у форматі JSON, що відповідає схемі IncomingAlert (валідація через Pydantic-моделі). Повідомлення публікується у direct-exchange monitoring.events із routing-key scom.alert і прив’язаною до нього довговічною (durable) чергою events.processing. Використовується режим

persistent – повідомлення зберігаються на диску RabbitMQ, що гарантує відсутність втрат у випадку перезапуску брокера.

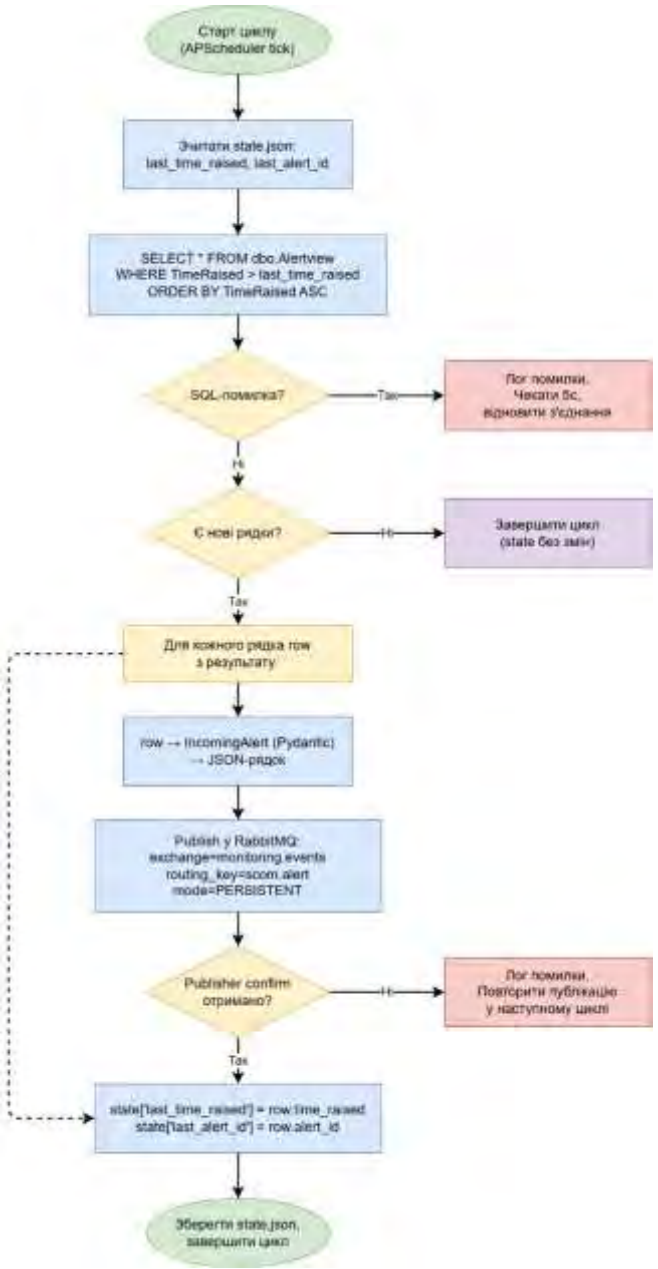

Рисунок 3.1. – Алгоритм одного циклу опитування SCOM Alertview

Цикл публікації побудовано як неблокуючий: pyodbc-курсор працює синхронно (бо MS SQL ODBC-драйвер не підтримує async), але публікація в RabbitMQ через aio-pika виконується асинхронно. Це дозволяє паралельно виконувати читання з SQL і публікацію в брокер. При збою повідомлення не

підтверджується і state.json не оновлюється, тож у наступному циклі алерт надсилається повторно, без втрат. Налаштування взаємодії з RabbitMQ наведено в таблиці 3.1.

Таблиця 3.1 – Параметри взаємодії з RabbitMQ

Параметр	Значення
Тип обміну (exchange type)	direct
Назва обміну	monitoring.events
Routing key для алертів SCOM	scom.alert
Назва черги	events.processing
Довговічність черги (durable)	TRUE
Режим публікації	persistent (DeliveryMode.PERSISTENT)
Підтвердження публікації	publisher confirms enabled
Prefetch count (споживач)	10

Сам процес опитування запускається через APScheduler'ів AsyncIOScheduler з триггером IntervalTrigger (seconds=30). Це гарантує, що навіть якщо черговий цикл затягнеться через тимчасові проблеми з SQL-сервером, наступний цикл не накладатиметься на попередній – диспетчер чекатиме завершення попереднього. Для адміністрування RabbitMQ використовується вбудований Management Plugin, що дозволяє спостереження за станом черг і статистикою в реальному часі.

3.2 Реалізація серверних мікросервісів обробки подій та API інцидентів, користувацького інтерфейсу і розгортання системи

Серверна частина системи побудована на основі мікросервісної архітектури, описаної у підрозділі 2.1. Кожен з мікросервісів виконує одну вузько спеціалізовану функцію та взаємодіє з іншими через брокер повідомлень RabbitMQ (для подій реального часу) або через REST API (для оперативних запитів від інтерфейсу). У цьому підрозділі розглянуто реалізацію двох ключових мікросервісів – Event Processor та Incident API, веб-інтерфейсу на React та налаштування розгортання у контейнерному середовищі Docker.

Event Processor – це асинхронний споживач повідомлень з черги events.processing. Реалізований мовою Python 3.11 на основі бібліотеки aio-pika, з використанням SQLAlchemy 2.0 в асинхронному режимі (драйвер asyncpg для PostgreSQL). Архітектурно Processor побудовано за принципом конвеєра (Pipeline Pattern): кожне повідомлення послідовно проходить через два етапи – фільтрації (filter_stage) та дедуплікації з можливим створенням інциденту (dedup_stage).

Етап фільтрації перевіряє відповідність події активним правилам типу filter. У системі за замовчуванням створено одне таке правило – FILTER-INFO, що відсікає події з рівнем severity=0 (Information) як шум. Адміністратор може створювати додаткові правила через REST API або веб-інтерфейс – наприклад, для відсікання подій від тестових серверів чи службових категорій. Якщо подія відповідає хоча б одному filter-правилу з дією drop, вона відкидається без подальшої обробки. У цьому випадку запис у raw_events не створюється – це принципове рішення, що дозволяє не «забивати» базу шумом.

Етап дедуплікації – найскладніша частина Processor'а. На цьому етапі:

– обчислюється криптографічна сигнатура події SHA-256 за формулою (2.4), що враховує джерело, об'єкт моніторингу, ідентифікатор правила SCOM і рівень severity;

– у таблиці raw_events створюється запис сирої події з оригінальним payload у форматі JSONB та обчисленою сигнатурою;

– виконується пошук активного інциденту (статус open або in_progress), що має ту саму сигнатуру і отримав останню подію не раніше ніж задане часове вікно тому (за замовчуванням – 900 секунд, тобто 15 хвилин);

– якщо такий інцидент знайдено – інкрементується його лічильник event_count, оновлюється last_event_at, а сама подія прив'язується до інциденту через таблицю incident_events;

– якщо ні, то створюється новий інцидент із порядковим публічним ідентифікатором формату INC-YYYY-NNNNNN, що генерується через PostgreSQL-послідовність incident_seq.

Пріоритет нового інциденту визначається за матрицею Impact × Severity згідно з методологією ITIL. Замість зваженого лінійного підходу (формула 2.5 з підрозділу 2.2) використано пряму матричну індексацію – це гарантує однозначне співставлення комбінації середовища (production/staging/dev) та критичності алерту (Critical/Warning/Information) з рівнем пріоритету від P1 до P5.

Паралельно з обробкою повідомлень Event Processor виконує фоновий планувальник автоматичного закриття застарілих інцидентів. Раз на хвилину перевіряється список інцидентів зі статусом open, для яких остання подія надійшла понад 30 хвилин тому. Такі інциденти переводяться у статус auto_resolved. Інциденти зі статусом in_progress не закриваються автоматично – це робить оператор вручну, бо переведення в in_progress означає, що ним вже займаються.

Incident API – другий ключовий мікросервіс, що реалізує REST-інтерфейс для взаємодії з користувацьким веб-інтерфейсом і зовнішніми системами (у перспективі можлива інтеграція з Jira Service Management через WebHook-механізм). Реалізовано на основі бібліотеки FastAPI з Pydantic 2.0 для валідації запитів та формування відповідей. Усі endpoint автоматично документуються у форматі OpenAPI 3.0, доступному за адресою inc-api.gnu.pp.ua/docs.

Структура API включає чотири основні групи endpoint:

– /api/v1/incidents – операції з інцидентами (отримання списку з фільтрацією та сортуванням, отримання деталей одного інциденту, оновлення статусу та призначення відповідального);

– /api/v1/rules – управління правилами обробки (CRUD для filter, dedup, correlation типів);

– /api/v1/stats – статистичні запити для Dashboard (розподіл за пріоритетами, статусами, топ-N «найгучніших» інцидентів за кількістю подій);

– /api/v1/health – endpoint для перевірки доступності сервісу.

Додатково реалізовано захист системних правил. Правила з прапорцем is_system=TRUE – це базові правила, що завжди мають бути присутніми в

системі (наприклад, FILTER-INFO та DEDUP-DEFAULT). DELETE-запит на такі правила завжди повертає HTTP 403 Forbidden, а PATCH-запит не дозволяє змінювати критичні поля conditions та rule_type, але дозволяє вмикати/вимикати правило та змінювати його description. Це захищає систему від випадкового видалення базової логіки.

Користувацький веб-інтерфейс реалізовано на основі React 18 з TypeScript і бібліотеки компонентів Ant Design 5. Для побудови інтерактивних графіків застосовано бібліотеку Recharts. Білд виконується через Vite – сучасний інструмент, що забезпечує швидке оновлення під час розробки та оптимізовану збірку для продукції.

Структура інтерфейсу побудована навколо трьох головних екранів. Перший – Dashboard – головний оглядовий екран з картками статистики (загальна кількість активних інцидентів, розподіл за пріоритетами), круговою діаграмою розподілу за пріоритетами, стовпчиковою діаграмою кількості нових інцидентів за останні 24 години та списком топ-5 «найгучніших» інцидентів за кількістю агрегованих подій. Другий – картки угорі зі статистичними даними (кількість all/open/in_progress/auto_resolved/closed) та список інцидентів (таблиця з можливістю фільтрації за пріоритетом, статусом, періодом створення, а також сортування за будь-яким стовпцем). Третій – деталі інциденту де вказана повна картка з усією інформацією про інцидент, списком пов’язаних сирих подій, історією змін статусу та елементами управління (зміна статусу, призначення відповідального).

Зовнішній вигляд головного екрана Dashboard у режимі експлуатації наведено на рисунку 3.2. Веб-інтерфейс взаємодіє з Incident API через axios у асинхронному режимі. Для управління станом використовується React Query (TanStack Query) – бібліотека, що забезпечує кешування результатів запитів, автоматичне оновлення при зміні параметрів та фонове повторне отримання даних. Це дозволяє Dashboard оновлюватися кожні 30 секунд, демонструючи свіжу статистику без зайвих ручних дій від оператора.

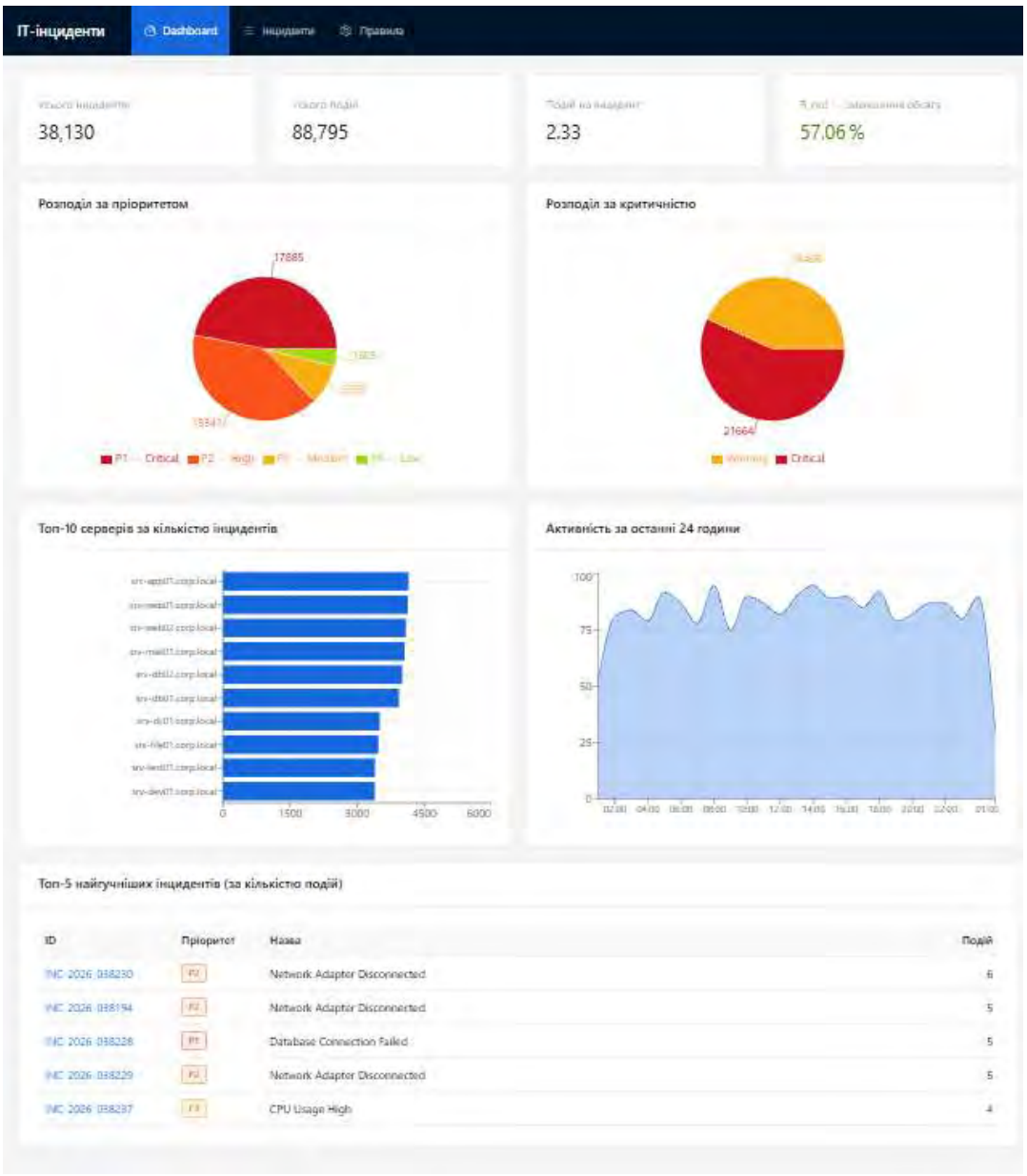

Рисунок 3.2. – Екран Dashboard у режимі експлуатації системи

Усі сервіси системи розгорнуто в контейнерному середовищі Docker із використанням Docker Compose для оркестрації. Це забезпечує відтворюваність розгортання – будь-який вузол з підтримкою Docker може за одну команду docker compose up -d підняти повний стек системи. Усі сервіси описано у файлі docker-compose.yml (повний лістинг наведено у додатку Д), де визначено образи, мережі, томи (volumes) для збереження даних, змінні середовища та залежності.

Особливу увагу приділено правильному порядку запуску сервісів. Це досягається через секцію depends_on з умовами condition: service_healthy. Наприклад, scom-collector чекає, доки mock-scom-db не пройде healthcheck-тест

(виконання простого SELECT-запиту), і лише після цього починає опитування. Це усуває класичну проблему «гонок при старті», коли клієнт намагається підключитися до бази, що ще не завершила ініціалізацію. Список усіх контейнерів системи з їх ролями узагальнено в таблиці 3.2.

Таблиця 3.2 – Контейнери системи в Docker Compose

Контейнер	Базовий образ	Роль у системі
mock-scom-db	mcr.microsoft.com/mssql/server:2022-latest	Мок-сервер MS SQL Server для імітації SCOM Alertview
alert-generator	python:3.11-slim (custom)	Генератор тестових алертів у Alertview
scom-collector	python:3.11-slim (custom)	Опитує Alertview, публікує події в RabbitMQ
rabbitmq	rabbitmq:3.12-management	Брокер повідомлень з Management UI
event-processor	python:3.11-slim (custom)	Споживає черги, фільтрує, дедуплікує, створює інциденти
postgres	postgres:15-alpine	Основна СУБД для зберігання інцидентів і подій
incident-api	python:3.11-slim (custom)	REST API на основі FastAPI
web-ui	nginx:alpine (multi-stage)	Веб-інтерфейс React, що віддається через Nginx

Доступ до системи з зовнішніх мереж реалізовано через Cloudflare Tunnel – захищений тунель між сервером і мережею Cloudflare без необхідності прокидувати NAT-портфорвардинг чи мати публічну IP-адресу. Три піддомени відповідають трьом точкам входу: inc.gnu.pp.ua – для веб-інтерфейсу, inc-api.gnu.pp.ua – для REST API з документацією Swagger, inc-mq.gnu.pp.ua – для адміністративного UI RabbitMQ. Кожен з піддоменів захищено через Cloudflare Access із вимогою аутентифікації Google SSO – це забезпечує контроль доступу на рівні організації без зміни коду самої системи.

Розгортання в ізольованому середовищі без прямого доступу ззовні супроводжувалося двома практичними труднощами. По-перше, на етапі складання Docker-образів контейнери не успадковували DNS-резолвер хоста, через що звернення до зовнішніх репозиторіїв пакетів завершувалися таймаутом; проблему вирішено явним заданням DNS-серверів у конфігурації демона Docker.

По-друге, інтерактивна автентифікація Google SSO, прийнятна для веб-інтерфейсу, блокувала програмні звернення до REST API, тому для відповідного піддомену політику Cloudflare Access переведено в режим Bypass із перенесенням контролю доступу до API на рівень самого застосунку.

3.3 Тестування реалізованої системи

Перевірка коректності реалізованої системи здійснювалася у двох напрямках. Перший це функціональне тестування яке спрямоване на встановлення відповідності фактичної поведінки системи функціональним вимогам, сформульованим у підрозділі 1.4. Другий – навантажувальне тестування, що орієнтоване на вимірювання кількісних показників продуктивності системи: пропускної здатності конвеєра обробки, ступеня скорочення обсягу подій, а також стійкості до пікових навантажень, що відповідають реальним сценаріям роботи системи моніторингу в корпоративному середовищі.

Для проведення тестування створено експериментальне середовище, що відтворює структуру реальної інсталяції SCOM у мініатюрі. Замість продуктивного екземпляра SCOM використано мок-сервер на базі Microsoft SQL Server 2022 Express із спеціально розробленою схемою dbo.Alertview, що повторює сигнатуру однойменного представлення реальної бази OperationsManager. Підтримуються десять об’єктів моніторингу різного типу (контролери баз даних, веб-сервери, файлові сервери, поштові сервери, тестові середовища) та десять шаблонів типових алертів, що охоплюють найпоширеніші категорії: CPU/Memory/Disk usage, Service Down, Database Connection Failed, IIS Application Pool Stopped, Network Adapter Disconnected, Failed Login Attempts, Backup Job Failed, Certificate Expiring.

Результати функціонального тестування за всіма функціональними вимогами FR-01 – FR-08, сформульованими у підрозділі 1.4, наведено у таблиці 3.3.

Таблиця 3.3 – Результати функціонального тестування

Код	Вимога	Процедура перевірки	Результат
FR-01	Отримання алертів зі SCOM	Перевірено наявність повідомлень у RabbitMQ exchange monitoring.events через Management UI; ручне порівняння кількості рядків у Alertview та повідомлень у черзі	Пройдено
FR-02	Фільтрація шумових подій	Системне правило FILTER-INFO відсікає події з severity=0; перевірено, що такі події не створюють інцидентів у БД	Пройдено
FR-03	Дедуплікація подій	Перевірено агрегацію подій з однаковою сигнатурою (формула 2.4) у часовому вікні 15 хвилин; у пікових прогонах підтверджено кратність 49–95 подій на один інцидент	Пройдено
FR-04	Управління правилами	Перевірено CRUD-операції через веб-інтерфейс та REST API; підтверджено захист системних правил (is_system=TRUE) від видалення та зміни критичних полів	Пройдено
FR-05	Пріоритизація інцидентів	Перевірено відповідність матриці ITIL: production+Critical→P1, staging+Critical→P2, dev+Warning→P4	Пройдено
FR-06	Автоматичне закриття	Перевірено перехід open→auto_resolved через 30 хвилин тиші; підтверджено, що in_progress інциденти не закриваються автоматично	Пройдено
FR-07	Перегляд деталей інциденту	Перевірено відображення повного списку пов’язаних подій та назви правила обробки, що спричинило створення інциденту	Пройдено
FR-08	Зміна статусу оператором	Перевірено перехід open→in_progress→resolved через PATCH-запит API та відповідне відображення зміни в UI у реальному часі	Пройдено

Для оцінки продуктивності системи виконано серію навантажувальних випробувань, що включала три прогони з різною інтенсивністю надходження подій. Прогон A (базовий) – імітує штатне навантаження зі сталою інтенсивністю генерації алертів (один алерт кожні 5 секунд від 10 серверів плюс випадкові каскадні сценарії з імовірністю 5%). Прогони B і C імітують пікове навантаження («шторм алертів») – короткочасний сплеск, що виникає при каскадних збоях у корпоративній інфраструктурі. Для генерації штормів розроблено окремий модуль storm_injector, що виконує прямі INSERT-запити до таблиці Alertview батчами по 50 рядків із заданою інтенсивністю.

Параметри та результати трьох прогонів узагальнено у таблиці 3.4. Усі прогони виконано на одному стенді (LXC-контейнер з 4 ГБ оперативної пам’яті

та 4 vCPU під керуванням Debian GNU/Linux 13) із попереднім очищенням бази даних інцидентів та сирих подій перед кожним прогоном.

Таблиця 3.4 – Параметри та результати навантажувальних випробувань

Параметр / показник	A. Базовий	B. Шторм 5000	C. Шторм 10000
Тривалість прогону	10 хв	5 хв	8 хв
Інтенсивність генерації	~0,2 події/с	83 події/с (60 с)	166 події/с (60 с)
Подій оброблено	133	5 009	10 017
Подій відкинуто (filter)	6	510	1 005
Подій агреговано	62	4 408	8 917
Інцидентів створено	65	91	95
Кратність (подій/інцидент)	1,9	49,4	94,9
R_{red} (коеф. зменшення)	0,51	0,980	0,990
Пікова інтенсивність публікації, повід./с	2,8	300,4	307,0
Пікова інтенсивність обробки, повід./с	2,8	299,6	307,4
Максимальна глибина черги, повід.	0	15	22
Втрачено повідомлень	0	0	0

Прогон A демонструє роботу системи в режимі звичайного навантаження. За 10 хвилин неперервної обробки в черзі RabbitMQ не накопичилося жодного повідомлення, оскільки інтенсивність надходження залишалася значно нижчою за пропускну здатність процесора подій. Усі 133 повідомлення, що надійшли від мок-SCOM, успішно оброблені. 6 з них відкинуто системним правилом FILTER-INFO, 62 – агреговано до існуючих інцидентів, 65 – створили нові інциденти.

Прогони B і C імітували каскадні сценарії пікового навантаження. У прогоні B протягом 60 секунд у систему вкинуто 5000 алертів зі швидкістю 83 події/с (близько до граничного значення нефункціональної вимоги NFR-02 – 100 події/с). У прогоні C протягом тих самих 60 секунд вкинуто 10000 алертів зі швидкістю 166 події/с – у 1,66 рази вище за вимогу NFR-02. У обох прогонах система обробила всі повідомлення без втрат: на черзі RabbitMQ максимальна глибина черги не перевищила 22 повідомлень навіть під час прогону C, а інтенсивність обробки досягла 307 повідомлень/с у пікові моменти – більш ніж утричі вище за номінальну вимогу 100 подій/с. Перші навантажувальні прогони

виявили проблему конкурентного доступу, характерну саме для асинхронної обробки. За паралельного споживання повідомлень із черги (prefetch_count = 10) два обробники одночасно отримували події від одного й того самого об’єкта моніторингу і намагалися створити для нього однакову конфігураційну одиницю, через що друга вставка завершувалася помилкою порушення унікального обмеження (duplicate key value violates unique constraint). Послідовність «перевірка наявності – вставка» виявилася неатомарною в умовах паралельних транзакцій. Проблему усунено заміною цієї послідовності на атомарну операцію PostgreSQL INSERT … ON CONFLICT DO NOTHING з подальшим перечитуванням запису, після чого повторні штормові прогони проходили без помилок у журналі обробника та без втрати подій.

Найкритичніший показник – коефіцієнт зменшення обсягу подій R_{red}. На базовому прогоні A значення R_{red} = 0,51 – нижче за цільовий показник 0,70 з підрозділу 1.4. Це пояснюється тим, що за 10 хвилин при поточному наборі з 10 шаблонів алертів і 10 серверів виникає лише 100 унікальних комбінацій сигнатур, тому потенціал дедуплікації значною мірою не реалізується. У реальній інфраструктурі з сотнями серверів і тривалими часовими інтервалами R_{red} на штатному навантаженні очікувано буде вищим.

Натомість прогони B і C демонструють реальну ефективність системи в умовах, коли потенціал дедуплікації використовується повністю: R_{red} = 0,980 та 0,990 відповідно. Це означає, що з кожних ста подій моніторингу оператор бачить лише одну-дві позиції у списку інцидентів. Решта 98–99% подій агрегується механізмом дедуплікації за сигнатурою SHA-256 (формула 2.4) або відсікається системним правилом фільтрації. Подібний рівень скорочення відповідає типовим показникам промислових SIEM-систем класу Splunk, IBM QRadar та аналогічних – за даними галузевих звітів, очікуваний рівень R_{red} становить 90–98%.

На рисунку 3.4 наведено динаміку публікації та обробки повідомлень у черзі RabbitMQ під час прогону C (шторм 10 000 подій). Видно, що публікація (Publish rate) досягає 307 повідомлень/с у пікові моменти, проте швидкість

обробки (Deliver rate) практично точно відстежує її, з мінімальним відставанням. Глибина черги жодного разу не перевищила 22 повідомлень – це означає, що система працює як «прозорий конвеєр», а не як «бутилочне горло». Беклогу повідомлень у черзі не виникає, що свідчить про відповідну паралелізацію обробки та достатню продуктивність базового рівня (PostgreSQL з пулом з'єднань і асинхронним драйвером asyncpg).

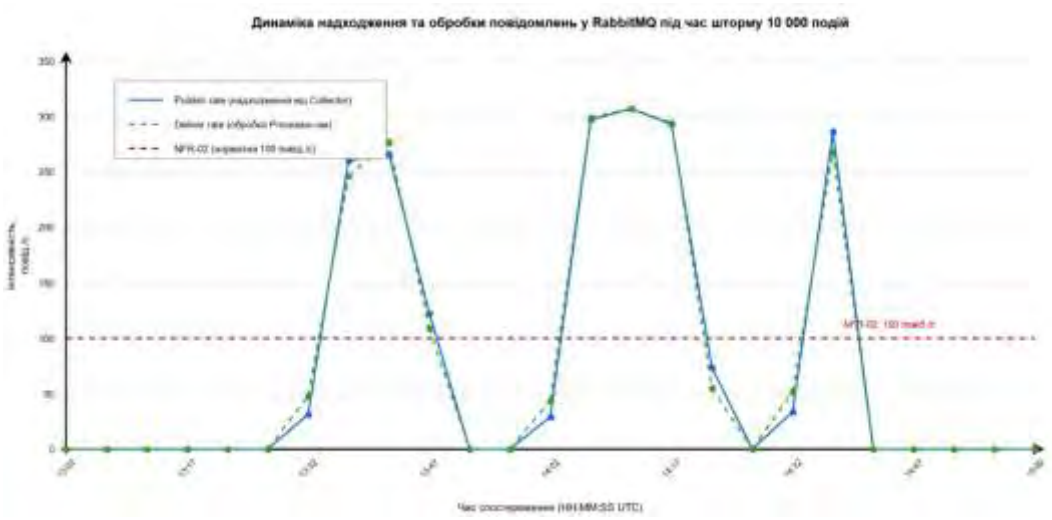

Рисунок 3.4. – Динаміка надходження та обробки повідомлень у RabbitMQ під час прогону C

Окремим показником стабільності системи є рівномірний розподіл створених інцидентів за пріоритетами, що відповідає матриці ITIL – таблиця 3.5. У всіх прогонах P1 (критичні події на production-серверах) становлять близько 40-48%, P2 (важливі) – 38-42%, P3 (середні) – 8-11%, P4 (низькі) – 2-4%. Такий розподіл реалістично відтворює корпоративну інфраструктуру, де більшість критичних об'єктів моніторингу належать до production-середовища.

Таблиця 3.5 – Розподіл інцидентів за пріоритетами

Пріоритет	A. Базовий	B. Шторм 5000	C. Шторм 10000
P1 – Критичний	31 (47,7%)	40 (44,0%)	43 (45,3%)
P2 – Високий	25 (38,5%)	38 (41,8%)	38 (40,0%)
P3 – Середній	7 (10,8%)	9 (9,9%)	10 (10,5%)
P4 – Низький	2 (3,1%)	4 (4,4%)	4 (4,2%)
Разом	65	91	95

Перевірка відповідності системи нефункціональним вимогам (NFR-01–NFR-04 з підрозділу 1.4) підсумована у таблиці 3.6. Усі чотири нефункціональні вимоги виконано з суттєвим запасом – фактичні значення перевищують встановлені пороги з істотним запасом.

Таблиця 3.6 – Відповідність нефункціональним вимогам

Код	Вимога	Поріг	Фактичне значення	Результат
NFR-01	Затримка обробки події	≤ 5 с	< 0,1 с (глибина черги ≤ 22 повід. за піку; за штатного навантаження черга порожня)	Виконано
NFR-02	Пропускна здатність обробки	≥ 100 подій/с	307 подій/с (пік прогону C)	Виконано
NFR-03	Надійність (збереження подій)	без втрат	0 втрачених повідомлень у всіх прогонах	Виконано
NFR-04	Доступність сервісу	≥ 99 %	100 % за час тестування, без перезапусків контейнерів	Виконано

Загальна стабільність системи підтверджена тривалими сумарними прогонами: за майже 30 хвилин активного навантаження жоден з контейнерів сервісу не зазнав перезапуску чи аварійного завершення. Розхід оперативної пам’яті подієвого процесора залишався стабільним – у межах 150–220 МБ під час пікового навантаження. Сервер бази даних PostgreSQL не перевищив 350 МБ оперативної пам’яті, RabbitMQ – 200 МБ. Загальне використання пам’яті стенду під час найважчого прогону C склало близько 60% від виділених LXC-контейнеру 4 ГБ, що залишає значний запас для масштабування на більший обсяг інфраструктури.

Підсумовуючи результати тестування, можна стверджувати, що реалізована система відповідає усім функціональним і нефункціональним вимогам, сформульованим у розділі 1, з суттєвим запасом за ключовими показниками. Особливо вирізняється здатність системи витримувати пікові навантаження, що перевищують номінальні значення NFR-02, без накопичення черги повідомлень і без втрат даних. Це підтверджує правильність архітектурних рішень, прийнятих у розділі 2 – зокрема, використання асинхронного брокера

повідомлень RabbitMQ як буфера між компонентами, неблокуючої реалізації процесора подій на бібліотеці aio-pika та SQLAlchemy 2.0 в асинхронному режимі.

3.4 Техніко-економічне обґрунтування ефективності розробленої системи

Економічна доцільність розробки та впровадження системи автоматизації обробки IT-інцидентів визначається насамперед підвищенням продуктивності праці диспетчерів служби моніторингу за рахунок скорочення обсягу інформаційного шуму, а також низкою якісних ефектів, що знижують операційні ризики. У цьому підрозділі виконано оцінку собівартості розробки, щорічних експлуатаційних витрат та чистого річного економічного ефекту з консервативним урахуванням лише прямої економії робочого часу. Розрахунок здійснено згідно з підходом, рекомендованим у методичних посібниках з організації IT-проєктів.

Собівартість розробки прикладного програмного забезпечення складається з фонду оплати праці виконавця, нарахувань на заробітну плату, амортизації засобів виробництва та накладних витрат. Розрахункова трудомісткість оцінена виходячи з фактичних обсягів реалізованого коду – близько 3 500 рядків Python (мікросервіси SCOM Collector, Event Processor, Incident API та допоміжний інструментарій) і 2200 рядків TypeScript (веб-інтерфейс). За продуктивністю 12 рядків коду на годину сумарна трудомісткість становить 480 людино-годин. Розрахунок витрат на оплату праці розробника наведено в таблиці 3.7.

Накладні витрати за період розробки (3 місяці) включають амортизацію робочого ноутбука (15000 грн з періодом амортизації 3 роки – 1250 грн), оплату електроенергії (250 кВт·год × 4,32 грн = 1080 грн) та доступ до Інтернету

(близько 2000 грн), що сумарно становить 4330 грн. Сумарна собівартість розробки складає 326410 грн.

Таблиця 3.7 – Розрахунок фонду оплати праці розробника

Показник	Значення
Трудомісткість, людино-годин	480
Середня годинна ставка middle-розробника (Україна, 2026), грн/год	500
Основна заробітна плата, грн	240000
Додаткова заробітна плата (10%), грн	24000
Єдиний соціальний внесок (22%), грн	58080
Разом фонд оплати праці, грн	322080

Експлуатаційні витрати визначено для варіанту розгортання системи на орендованому віртуальному сервері хмарного провайдера у конфігурації 4 vCPU/8 ГБ ОЗП/50 ГБ SSD. Розрахунок наведено в таблиці 3.8.

Таблиця 3.8 – Щорічні експлуатаційні витрати

Стаття витрат	Сума, грн/рік
Оренда віртуального сервера (4 vCPU/8 ГБ)	9600
Підписка на домен	400
Cloudflare Tunnel (Free Plan)	0
Резервне копіювання і обслуговування	7600
Разом експлуатаційних витрат, грн/рік	17600

Економічний ефект від впровадження системи формується прямою економією робочого часу диспетчерів IT-моніторингу. Важливо враховувати, що диспетчер не опрацьовує кожен алерт окремо: переважна частина подій оцінюється візуально безпосередньо в консолі моніторингу, а під час каскадних збоїв (наприклад, масове відпадання агентів унаслідок мережевої аварії) оператор розпізнає спільну першопричину без поелементного аналізу, тоді як похідні алерти в такому потоці здебільшого закриваються автоматично. Тому економія праці не масштабується лінійно з кількістю відфільтрованих подій, а обмежується фактичним часом, який команда витрачає на тріаж потоку.

За оцінкою, отриманою в межах переддипломної практики, витрати часу на ручні операції з обробки потоку SCOM-алертів становлять близько 25 %

фонду робочого часу одного штатного диспетчера. Обробку потоку забезпечують двоє диспетчерів, тому сукупна вивільнювана частка відповідає половині штатної ставки. При середньорічній вартості диспетчера 396000 грн (з нарахуваннями) реалістична оцінка вивільненої цінності праці становить:

$$\mathrm{E_{пр}} = 0{,}5 \times 396000 = 198000\,\text{грн/рік}.$$

Складова економії від скорочення часу простою критичних сервісів має непрямий характер і залежить переважно від роботи інженерних команд, що усувають збій, а не від системи реєстрації подій, тому в консервативному розрахунку вона у грошову оцінку не включається. Чистий річний економічний ефект з урахуванням експлуатаційних витрат обчислюється як:

$$E_{\text{нет}} = E_{\text{пр}} - \mathrm{C_{оп}} = 198000 - 17600 = 180400\ \text{грн/рік}.$$

Простий термін окупності капітальних вкладень визначається як відношення собівартості розробки до чистого річного ефекту:

$$T_{\text{ок}} = \frac{C_{\text{розробки}}}{E_{\text{нет}}} = \frac{326\,410}{180\,400} \approx 1{,}8\ (22\ \text{місяці}).$$

Узагальнені результати техніко-економічного аналізу зведено в таблиці 3.9.

Таблиця 3.9 – Узагальнені результати ТЕО

Показник	Значення
Собівартість розробки, грн	326 410
Щорічні експлуатаційні витрати, грн	17 600
Економія від вивільнення часу диспетчерів $E_{пр}$, грн/рік	198 000
Чистий річний ефект $E_{нет}$, грн/рік	180 400
Простий термін окупності, місяців	≈ 22

Отриманий термін окупності перебуває в межах нормативного діапазону для ІТ-проєктів (1-3 роки), що свідчить про економічну доцільність розробленої системи навіть за найбільш консервативного припущення, коли в розрахунок беруться лише пряма економія робочого часу. Понад цей розрахунок система забезпечує низку якісних ефектів, які свідомо не переведено у грошову форму: пришвидшене виявлення значущих інцидентів з-під потоку шуму, рівномірне

покриття подій у нічні та вихідні зміни, а також зниження ризику пропуску значущих подій, які за поточного процесу нерідко реєструються лише після звернення постраждалого користувача. Урахування цих ефектів додатково скоротило б термін окупності, тому наведену оцінку слід розглядати як верхню межу. Це підтверджує доцільність впровадження системи в експлуатацію та можливість тиражування рішення на інші організації з аналогічним профілем інфраструктури.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну задачу автоматизації реєстрації та обробки IT-інцидентів на основі агрегації та фільтрації подій моніторингу. Поставленої мети досягнуто – розроблено, реалізовано та апробовано працездатну програмну систему, що виступає проміжним шаром між системою моніторингу SCOM та ITSM-платформою.

У результаті аналізу предметної області розглянуто сутність процесу управління IT-інцидентами згідно з практиками ITIL, проаналізовано ринок систем моніторингу та ITSM-платформ в Україні й досліджено методи агрегації, кореляції та фільтрації подій. Встановлено, що в типовому корпоративному середовищі України відсутня доступна вбудована інтелектуальна інтеграція між системами моніторингу класу SCOM та ITSM-платформами, що створює нішу для розроблення спеціалізованого проміжного шару. За результатами аналізу сформульовано десять функціональних та вісім нефункціональних вимог до проєктованої системи.

На етапі проєктування обґрунтовано мікросервісну архітектуру з чотирьох основних сервісів та двох інфраструктурних компонентів, обрано технологічний стек на основі мови Python з фреймворком FastAPI, брокера повідомлень RabbitMQ, СКБД PostgreSQL та фронтенд-стеку React з Ant Design. Спроєктовано структуру бази даних з нормалізацією до третьої нормальної форми та застосуванням типу JSONB і GIN-індексів для зберігання атрибутів подій, а також розроблено алгоритми фільтрації, дедуплікації та кореляції з обчисленням сигнатури події за алгоритмом SHA-256.

На етапі реалізації систему побудовано та розгорнуто в контейнерному середовищі Docker з оркестрацією через Docker Compose. Функціональне тестування підтвердило відповідність системи сформульованим вимогам. Навантажувальне тестування показало, що за штормових сценаріїв коефіцієнт зменшення обсягу подій сягає 0,98–0,99, пікова пропускна здатність обробки досягає 307 повідомлень на секунду при нормативній вимозі 100 повідомлень на

секунду, при цьому максимальна глибина черги не перевищувала двадцяти двох повідомлень, а втрати повідомлень були відсутні. Це підтверджує правильність прийнятих архітектурних рішень, зокрема використання асинхронного брокера повідомлень як буфера між компонентами.

За результатами техніко-економічного обґрунтування собівартість розробки системи становить 326410 грн, а простий термін окупності з урахуванням лише прямої економії робочого часу диспетчерів служби моніторингу складає близько 22 місяців (1,8 року), що перебуває в межах нормативного діапазону для ІТ-проєктів. Понад цей розрахунок система забезпечує низку якісних ефектів – пришвидшене виявлення значущих інцидентів, рівномірне покриття подій у нічні та вихідні зміни і зниження ризику пропуску значущих подій, які свідомо не переведено у грошову форму, що додатково підкреслює консервативність оцінки.

Розроблена система готова до пілотного впровадження в експлуатацію. Напрямами подальшого розвитку є інтеграція з Jira Service Management через механізм WebHook, доповнення правил кореляції методами машинного навчання, розширення засобів керування правилами через інтерфейс та посилення підсистеми безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ITIL: оптимізація підтримки застосунків та пріоритизація інцидентів. EPAM Україна. 2024. URL: https://careers.epam.ua/blog/itil-challenges-apps-support-optimization-and-incident-prioritization-webinar-key-take-aways

2. ITSM і ITIL 4: сучасний підхід до управління IT-послугами. Навчальний центр "Мережні технології". 2025. URL: https://www.nt.ua/what-is-itsm-and-why-does-your-business-need-it

3. Сисоєнко С., Бабенко В., Лада Н. Управління інцидентами інформаційної безпеки на об'єктах критичної інфраструктури. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. Vol. 355, No. 4. P. 555–559. URL: https://doi.org/10.31891/2307-5732-2025-355-78

4. SIEM-система: централізований моніторинг подій безпеки та швидке реагування на інциденти. ITEZ. 2026. URL: https://itez.com.ua/blog/siem-systems-security-monitoring-and-incident-response.html

5. SLA для IT-послуг: Як скласти вигідну угоду з провайдером. ITEZ. 2026. URL: https://itez.com.ua/blog/what-is-sla-service-level-agreement-in-it.html

6. Що таке MTTR та як він відрізняється від інших показників управління інцидентами? Dynatrace. Bakotech. 2024. URL: https://dynatrace.bakotech.com/ua/what-is-mttr

7. ДСТУ ISO/IEC 27001:2023 (ISO/IEC 27001:2022, IDT) Інформаційна безпека, кібербезпека та захист конфіденційності. Вимоги. Київ: УкрНДНЦ, 2023. 45 с. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=104398

8. Prometheus vs Zabbix: відмінне та подібне цих систем моніторингу. IT Education Center. 2024. URL: https://itedu.center/ua/blog/comparisons/prometheus_vs_zabbix/

9. Operations Manager Key Concepts. Microsoft Learn. 2025. URL: https://learn.microsoft.com/en-us/system-center/scom/key-concepts

10. System requirements for System Center Operations Manager. Microsoft Learn. 2025. URL: https://learn.microsoft.com/en-us/system-center/scom/system-requirements

11. Черніков П. Кіберзахист міських цифрових систем: масштабування кореляції подій у SIEM для зменшення інцидентів. *Кібербезпека: освіта, наука, техніка*. 2025. Vol. 3, No. 31. P. 773–780. URL: https://doi.org/10.28925/2663-4023.2025.31.1066

12. Системний аналіз динаміки кібератак і процесів пентестингу на основі моделі ризику та графових структур. Кібербезпека: освіта, наука, техніка. 2025. URL: https://www.csecurity.kubg.edu.ua/index.php/journal/article/view/1056

13. Об'єднуй і володарюй: алгоритми кластеризації. Robot Dreams. 2025. URL: https://robotdreams.cc/uk/blog/93-obedinyay-i-vlastvuy-algoritmy-klasterizacii

14. Моніторинг розподіленої хмари: сервіси та інструменти. DOU. 2025. URL: https://dou.ua/forums/topic/54003/

15. Функціональні та нефункціональні вимоги: структура, SRS, приклади. Wezom. 2025. URL: https://wezom.com.ua/ua/blog/funktsionalni-ta-nefunktsionalni-vimogi-do-programnogo-zabezpechennya

16. Савченко Я., Левченко С. Аналіз вимог до програмного забезпечення для керування локальними комп'ютерними мережами. *Measuring and Computing Devices in Technological Processes*. 2026. № 1. С. 404–411. URL: https://doi.org/10.31891/2219-9365-2026-85-49

17. Солохін А. Є. Порівняльний аналіз Django, Flask та FastAPI. *Матеріали 28-го Міжнар. молодіж. форуму «Радіоелектроніка та молодь у XXI столітті».* Харків: ХНУРЕ, 2024. URL: https://openarchive.nure.ua/handle/document/28658

18. Мікросервісна архітектура: основні концепції та виклики. FoxmindEd. 2024. URL: https://foxminded.ua/mikroservisna-arkhitektura/

19. Мікросервіси та мікросервісна архітектура: сучасний підхід до розробки ПЗ. BizMag. 2025. URL: https://bizmag.com.ua/arkhitektura-mikroservisiv-dlia-biznesu/

20. Види протоколів в RabbitMQ. Друкарня. 2024. URL: https://drukarnia.com.ua/articles/rabbitmq-protokoli-FfYEy

21. Що таке Message Queue (черга повідомлень)? ITExpert. 2024. URL: https://itexpert.work/uk/cherga-povidomlen-message-queue-mq/

22. Гелей Р. Я., Богач І. В. Огляд фреймворків Python для веб-розробки, їх сильні та слабкі сторони. *Матеріали конференції ВНТУ*. 2024. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21788

23. PostgreSQL чи MySQL. Як обрати оптимальну базу даних для вашого проєкту. DOU. 2024. URL: https://dou.ua/forums/topic/51621/

24. PostgreSQL vs MongoDB vs MySQL vs Oracle: вибір бази даних. Webscraft. 2025. URL: https://webscraft.org/blog/postgresql-vs-mongodb-vs-mysql-vs-oracle-vibir-bazi-danih

25. PostgreSQL: Documentation. GIN Indexes. 2025. URL: https://www.postgresql.org/docs/current/gin.html

26. ISO/IEC 20000-1:2018. Information technology – Service management – Part 1: Service management system requirements. Geneva: ISO, 2018.

27. ISO/IEC 27035-1:2023. Information technology – Information security incident management – Part 1: Principles and process. Geneva: ISO, 2023.

28. Cichonski P., Millar T., Grance T., Scarfone K. Computer Security Incident Handling Guide: NIST Special Publication 800-61 Revision 2. Gaithersburg: NIST, 2012. URL: https://doi.org/10.6028/NIST.SP.800-61r2

29. Якименко Ю. М., Легомінова С. В., Щавінський Ю. В., Рабчун Д. І. Управління інцидентами інформаційної безпеки. Сучасні методи і засоби: навч. посіб. Київ: Державний університет телекомунікацій, 2023. 241 с.

30. Salah S., Maciá-Fernandez G., Díaz-Verdejo J. E. A model-based survey of alert correlation techniques. *Computer Networks*. 2013. Vol. 57, No. 5. P. 1289–1317. URL: https://doi.org/10.1016/j.comnet.2012.10.022

31. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. Geneva: ISO, 2018.

32. ISO/IEC 25010:2023. Systems and software engineering – SQuaRE – Product quality model. Geneva: ISO, 2023.

33. Parmar J., Sanghavi S., Prasad V., Shah P. Microservice Architecture Observability Tool Analysis. *Soft Computing and Signal Processing (ICSCSP 2022). Smart Innovation, Systems and Technologies*, vol. 313. Singapore: Springer, 2023. URL: https://doi.org/10.1007/978-981-19-8669-7_1

34. Fu G., Zhang Y., Yu G. A Fair Comparison of Message Queuing Systems. *IEEE Access*. 2021. Vol. 9. P. 421–432. URL: https://doi.org/10.1109/ACCESS.2020.3046503

35. AMQP 0-9-1 Model Explained. RabbitMQ. URL: https://www.rabbitmq.com/tutorials/amqp-concepts

36. Kleppmann M. Designing Data-Intensive Applications. Sebastopol: O'Reilly Media, 2017. 616 p. ISBN 978-1449373320.