

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ
ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці» Кафедра інформаційних
систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

Форма навчання: денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему: «Система моніторингу та аналізу лог-файлів в режимі реального часу»
здобувача Байрака Ігора Віталійовича

Науковий керівник:

д.е.н., професор
_____ Лондар С.Л.

Робота допущена до захисту перед
екзаменаційною комісією з атестації
здобувачів вищої освіти

завідувач кафедри:
к.е.н., доцент
_____ Тішков Б.О.

Київ 2025

Міністерство освіти і науки України
Київський національний економічний університет імені Вадима Гетьмана
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»

галузь знань 12 «Інформаційні технології»

спеціальність 122 «Комп'ютерні науки»

ПОГОДЖЕНО:

Керівник проєктної групи(гарант)
освітньо-професійної програми

_____Помазун О.М.

“ ____ ” _____ 2025 р.

ЗАТВЕРДЖУЮ:

Завідувач кафедри

_____Тішков Б.О.

“ ____ ” _____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Байрака Ігора Віталійовича

очної (денної) форми навчання
на підготовку кваліфікаційної бакалаврської роботи

на тему: «Система моніторингу та аналізу лог-файлів в режимі реального часу»

Тему затверджено наказом ректора Університету від « ____ » 2025 р.

№ _____ - *ст.*

Кваліфікаційна бакалаврська робота виконується на матеріалах
публікаціях в інтернеті та особистому досвіді користування подібними системами

План кваліфікаційної бакалаврської роботи

Розділ I Характеристика та аналіз предметної галузі

Розділ II Розробка вимог і моделювання інформаційної системи

Розділ III Проектування та реалізація компонентів системи

Об'єкт дослідження інформаційні системи, які збирають та аналізують лог-файли
в умовах динамічного змінного навантаження.

Предметом дослідження є інформаційні процеси та комп'ютерні технології, що
забезпечують моніторинг, збір, обробку й аналіз лог-файлів у режимі реального
часу для забезпечення стабільності IT-інфраструктури.

Мета кваліфікаційної бакалаврської роботи провести дослідження та
реалізувати проєктні рішення, спрямовані на розробку інформаційної системи для
моніторингу та аналізу лог-файлів у реальному часі, що забезпечує своєчасне
виявлення подій, помилок та аномалій у роботі цифрових систем

Конкретні завдання, які здобувач повинен виконати для досягнення
поставленої мети:

У розділі I необхідно описати предметну галузь та інформаційні процеси,
пов'язані з моніторингом і аналізом лог-файлів у режимі реального часу. Слід

надати характеристику об'єкта дослідження — інформаційної системи, що забезпечує збір, обробку та візуалізацію лог-даних. Також потрібно представити результати аналізу літературних джерел та існуючих рішень у сфері моніторингу логів, інструментів обробки великих обсягів даних та сучасних підходів до побудови систем спостереження.

У розділі II варто визначити бізнес-вимоги, функціональні та нефункціональні вимоги до проєктованої системи моніторингу. Ці вимоги необхідно подати у вигляді діаграм та специфікацій, створених з використанням CASE-засобів та мови SysML. Також потрібно сформулювати постановку задачі та запропонувати алгоритм її розв'язання. Для моделювання архітектури та функціонування системи слід використати діаграми мови UML.

У розділі III слід виконати проєктування основних компонентів системи моніторингу: інформаційного, технічного та програмного забезпечення. Необхідно реалізувати web-застосунок, що забезпечує збирання, обробку та аналіз лог-файлів у режимі реального часу. У цьому розділі також потрібно подати результати реалізації та обґрунтувати можливість і доцільність впровадження запропонованого рішення на практиці.

**Завдання підготував
науковий керівник**

Лондар Сергій Леонідович
« 10 » березня 2025 р.

**Завдання одержав
здобувач**

Байрак Ігор Віталійович
« 10 » березня 2025 р.

Відгук
на кваліфікаційну бакалаврську роботу
здобувача навчально-наукового інституту
«Інститут інформаційних технологій в економіці»
освітньо-професійної програми
«Комп'ютерні науки»

Байрака Ігора Віталійовича,

на тему

«Система моніторингу та аналізу лог-файлів в режимі реального часу»

Актуальність теми обумовлена необхідністю підтримки стабільного функціонування сучасних ІТ-систем та оперативного виявлення збоїв у їх роботі. З ускладненням розподілених архітектур, зростанням обсягів лог-файлів і підвищенням вимог до швидкості реагування на інциденти, системи моніторингу та аналізу логів у реальному часі стають невід'ємним елементом ІТ-інфраструктури. Сучасні компанії потребують ефективних інструментів для своєчасного виявлення аномалій, автоматизації моніторингових процесів і дотримання нормативних вимог у сфері аудиту подій.

У процесі виконання роботи було здійснено всебічний аналіз теоретичних засад і практичних підходів для систем логування та моніторингу лог-файлів. Порівняно низку популярних програмних рішень, таких як Kibana і OpenSearch, виявлено їхні сильні сторони та недоліки. На основі отриманих висновків спроектовано власну систему з чітко сформульованими функціональними й нефункціональними вимогами. Розроблено ключові UML-діаграми, включно з діаграмами варіантів використання, діяльності, послідовностей, класів, компонентів, трасування та прецедентів, а також створено даталогічну модель бази даних, необхідну для реалізації інформаційної системи.

Роботу виконано самостійно з дотриманням академічної доброчесності. Всі джерела, використані під час дослідження, належно процитовані, а запозичені фрагменти чітко відмежовані від авторського викладу. Автор

дослідив сучасні підходи до збору, обробки та аналізу лог-файлів у реальному часі, звернувши особливу увагу на особливості впровадження таких рішень в умовах зростання обсягів даних і складності ІТ-інфраструктур. У результаті аналізу існуючих програмних рішень було виявлено їх сильні та слабкі сторони, що дозволило сформулювати вимоги до власної розробки. Створено концепцію та реалізовано прототип авторської системи моніторингу логів, яка поєднує ефективність у виявленні аномалій, гнучкість налаштувань і зручність у використанні.

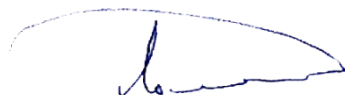
В рамках дослідження проаналізовано проблематику обробки великого обсягу лог-файлів, які генеруються ІТ-системами в режимі реального часу. Автором досліджено основні принципи побудови систем для оперативного відстеження подій, виявлення помилок та збоїв у роботі програмного забезпечення. Було спроектовано інтерфейс користувача, що забезпечує зручний перегляд логів, фільтрацію повідомлень, пошук за ключовими подіями, а також відображення критичних інцидентів. Реалізовано механізми візуального аналізу даних, генерації звітів та інструменти для формування сповіщень при настанні аномальних ситуацій.

В роботі є деякі недоліки, до яких можна віднести відсутність економічного обґрунтування створення нової інформаційної системи; відсутність опису авторизації та автентифікації застосунку; відсутність модульного та інтеграційного тестування серверної частини додатку;

Проте ці зауваження не зменшують теоретичної та практичної цінності роботи.

Вважаю, що бакалаврська робота **Байрака Ігоря Віталійовича** є завершеним, самостійним дослідженням, яке відповідає вимогам до бакалаврських проєктів у галузі комп'ютерних наук, робота може бути допущена до захисту із попередньою оцінкою 90 балів.

Науковий керівник,
д.е.н., професор



Сергій ЛОНДАР

“16” червня 2025 р.

РЕЦЕНЗІЯ

на кваліфікаційну бакалаврську роботу

здобувача вищої освіти

Байрака Ігора Віталійовича

Тема «Система моніторингу та аналізу лог-файлів в режимі реального часу»

Обрана тематика відповідає сучасним викликам у сфері ІТ, зокрема зростанню складності програмних рішень та необхідності швидкої реакції на збої. Сьогодні бізнеси критично залежать від стабільності інформаційних систем, а відтак від якості інструментів моніторингу. У зв'язку з цим дослідження, присвячене розробці засобів для обробки логів у реальному часі, є цілком обґрунтованим і відповідає нагальним потребам ринку. Особливої уваги заслуговує орієнтація автора на розв'язання практичних завдань, пов'язаних з виявленням аномалій і підтримкою надійності ІТ-інфраструктури.

Кваліфікаційна робота вирізняється системним аналітичним підходом та чіткою логікою викладу. Автором проведено порівняння актуальних рішень у галузі моніторингу логів, таких як Kibana та OpenSearch, виявлено ключові технічні особливості, переваги й обмеження. На основі цього було сформовано концепцію створення власного програмного продукту, з урахуванням практичних потреб цільових користувачів. Варто підкреслити наявність глибокого авторського технічного проєктування: всебічно обґрунтовано вимоги до системи, створено низку UML-діаграм, даталогічну модель БД тощо.

Проєктом демонструється впевнене володіння автором сучасними технологіями та принципами створення веб-застосунків, прототип реалізовано із застосуванням Next.js, WebSocket, PostgreSQL. Система дозволяє здійснювати перегляд логів, сортування за ключовими параметрами, візуалізацію критичних подій, а також формувати аналітичні звіти. Значною перевагою є реалізація механізмів фільтрації та пошуку, які роблять інструмент зручним і функціональним для системних адміністраторів та аналітиків.

Роботу виконано з дотриманням академічних стандартів, з посиланням на використані джерела та чітким відмежуванням авторських розробок. Студент самостійно провів аналіз наявних рішень, сформулював технічні вимоги та реалізував власне програмне рішення. Підхід до розробки свідчить про здатність до практичного впровадження теоретичних знань.

Застосунок, розроблений у межах дослідження, може бути адаптований до потреб малих та середніх підприємств, які не мають достатніх ресурсів для впровадження складних і дорогих комерційних рішень. Застосунок має потенціал для подальшого розширення: передбачено можливість інтеграції із зовнішніми

сервісами сповіщень, розгортання в хмарному середовищі та масштабування. Практичні рекомендації щодо покращення функціоналу свідчать про стратегічне бачення розвитку продукту.

Серед аспектів, які можуть бути поліпшені, варто зазначити відсутність тестування серверної частини (unit та integration tests), недостатньою є увага до питань автентифікації та авторизації, не наведено розрахунки економічної обґрунтованості проєкту.

Загалом, кваліфікаційна бакалаврська робота Байрака Ігора Віталійовича є завершеним прикладом прикладного дослідження з високим рівнем самостійності. Представлений прототип має чітко виражену практичну спрямованість, базується на сучасних технологіях і може слугувати основою для впровадження в реальних умовах. Роботу рекомендовано до захисту перед екзаменаційною комісією.

Місце роботи та посада рецензента:

ДНУ «Інститут освітньої аналітики»,
Науковий співробітник відділу
адміністрування освітніх
інформаційних систем, к.е.н.

 Ю. С. Тайдук
(підпис, ПІБ)

Підпис засвідчую:

Тайдук Ю.С.
(посада, підпис)

ЗАВІДУВАЧ
СЕКТОРУ КР
ДУДНИК О.І.



АНОТАЦІЯ

кваліфікаційної бакалаврської роботи

здобувача першого (бакалаврського) рівня вищої освіти, 4 курсу,
виконаної на тему: «Система моніторингу та аналізу лог-файлів в режимі
реального часу»

Київ: кафедра інформаційних систем в економіці, 2025 р.

У кваліфікаційному бакалаврському проєкті розглянуто процес створення інформаційної системи для моніторингу та аналізу лог-файлів у режимі реального часу. Метою системи є забезпечення оперативного виявлення подій, аномалій і потенційних загроз у роботі вебсервісів або ІТ-інфраструктури шляхом збору, обробки та візуалізації логів.

У роботі детально описано архітектуру інформаційної системи, основні функціональні модулі, вимоги до користувацького інтерфейсу та серверної частини. Для побудови інтерфейсу використано Figma, а діаграми бізнес-процесів і структури бази даних реалізовано в Enterprise Architect. Основний акцент зроблено на створенні ефективної системи оповіщення, фільтрації логів за різними критеріями та інтеграції з зовнішніми системами (CRM).

Розробка проведена на основі мови програмування JavaScript[30] з використанням фреймворку Next.js[31]. Для зберігання та обробки логів застосовано PostgreSQL[32]

Система орієнтована на адміністраторів і DevOps-фахівців, які мають потребу в інструментах для швидкого реагування на інциденти. Новизна проєкту полягає у застосуванні сучасних підходів до візуалізації подій у реальному часі, а також використанні шаблонів проєктування, які забезпечують масштабованість і надійність рішення.

РЕФЕРАТ

Кваліфікаційний бакалаврський проєкт містить 120 сторінок, 54 рисунки, список літератури з 27 найменувань, 3 додатки, 7 таблиць.

«Система моніторингу та аналізу лог-файлів в режимі реального часу»

Об'єкт дослідження: інформаційні системи, які збирають та аналізують лог-файли в умовах динамічного змінного навантаження.

Предметом дослідження є інформаційні процеси та комп'ютерні технології, що забезпечують моніторинг, збір, обробку й аналіз лог-файлів у режимі реального часу для забезпечення стабільності IT-інфраструктури.

Мета і завдання дослідження: Метою кваліфікаційної бакалаврської роботи є розробка інформаційної системи, яка дозволяє здійснювати моніторинг та аналіз лог-файлів у режимі реального часу для підвищення стабільності, прозорості та безпеки IT-інфраструктури.

Відповідно до поставленої мети визначені такі завдання:

- Аналіз існуючих методів та засобів моніторингу лог-файлів
- Проектування архітектури інформаційної системи з використанням UML-діаграм
- Розробка користувацького інтерфейсу системи за допомогою прототипування в Figma
- Реалізація системи з використанням JavaScript та PostgreSQL
- Розробка тест-кейсів для перевірки функціональних та нефункціональних вимог
- Проведення тестування та впровадження системи в тестовому середовищі

У процесі розробки були застосовані такі методи: моделювання системи за допомогою UML-діаграм, прототипування користувацького інтерфейсу в Figma, розробка тест-кейсів для перевірки функціональних та нефункціональних вимог.

Розробка інформаційної системи здійснювалася в середовищі WebStorm з використанням мови програмування JavaScript, а також системи управління реляційними базами даних PostgreSQL.

Для реалізації проекту використовувалося таке апаратне забезпечення:

- Процесор – Intel(R) Xeon(R) CPU E5-2660 V3 3,10GHz;
- Оперативна пам'ять – 16 Гб;
- Накопичувач – M2 SSD 128 Гб, SATA SSD 1 Тб, HDD 1 Тб;
- Дисплей – IPS 25" та IPS 27".

Теоретична, методична та практична значущість отриманих результатів: Результатом є створення гнучкої та масштабованої інформаційної системи, що забезпечує автоматичне виявлення аномалій у логах, сповіщення адміністраторів та збереження історії подій. Система може бути застосована в ІТ-компаніях, хостинг-провайдерах, банках та інших організаціях, де критично важливий постійний моніторинг стану інфраструктури. Ефективність розробленої системи підтверджена успішними результатами функціонального тестування та пілотного впровадження в тестовому середовищі.

Рік виконання кваліфікаційної бакалаврської роботи – 2025. Рік захисту роботи – 2025.

Ключові слова: JavaScript, PostgreSQL, лог-файли, реальний час, моніторинг, аналіз, API, система керування базами даних, діаграми UML, Elasticsearch, Kibana, Figma, тест-кейси, функціональні вимоги, нефункціональні вимоги, бізнес-вимоги.

ЗМІСТ

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ.....	2
Відгук.....	4
Рецензія.....	7
АНОТАЦІЯ.....	9
РЕФЕРАТ.....	10
ВСТУП.....	14
РОЗДІЛ 1	16
ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ.....	16
1.1 Характеристика предметної галузі та об'єкта дослідження	16
1.2 Аналіз літературних джерел та практичного досвіду використання ІС і технологій в предметній галузі	21
РОЗДІЛ 2	30
РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ	30
2.1 Аналіз і специфікація вимог до інформаційної системи	30
2.1.1 Бізнес-вимоги до системи.....	31
2.1.2. Функціональні вимоги до системи	32
2.1.3. Нефункціональні вимоги до системи	34
2.2 Постановка задачі та алгоритм розв'язання задачі	36
2.2.1. Вхідна інформація	38
2.2.2. Вихідна інформація	40
2.2.3. Алгоритм розв'язання задачі	43
2.3 Моделювання інформаційної системи	45
2.3.1. Моделювання поведінки системи	45
2.3.2. Моделювання структури системи	55
2.3.3. Моделювання користувацького інтерфейсу	61
РОЗДІЛ 3	65
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ.....	65
3.1 Інформаційне забезпечення	65
3.2 Технічне забезпечення	73
3.3 Програмне забезпечення.....	76
3.4 Результати реалізації інформаційної системи	81
РОЗДІЛ 4	92

ТЕСТУВАННЯ І ВЕРИФІКАЦІЯ ПРОЕКТНИХ РІШЕНЬ	92
4.1. Розробка тест-кейсів	92
4.2. Трасування вимог до системи.....	94
ВИСНОВОК	98
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	100
ДОДАТКИ	102
Додаток А	102
Додаток Б.....	107
Додаток В	121

ВСТУП

Тема дослідження "Система моніторингу та аналізу лог-файлів в режимі реального часу" була обрана через зростаюче значення аналізу операційних даних у сучасних ІТ-середовищах. У світі, де цифрова інфраструктура постійно ускладнюється, здатність ефективно моніторити та аналізувати логи є ключовою умовою стабільної роботи систем, своєчасного виявлення неполадок і забезпечення кібербезпеки.

Актуальність цього напрямку зумовлена низкою факторів. По-перше, обсяги даних, які щоденно генерують сучасні ІТ-системи, вже досягли таких масштабів, що їх аналіз вручну став практично неможливим. По-друге, розвиток мікросервісної архітектури, контейнеризації та хмарних рішень створює складну, розподілену інфраструктуру, де логи надходять з багатьох джерел, ускладнюючи централізований контроль. Крім того, бізнес потребує оперативного реагування на збої, що вимагає аналізу логів у режимі реального часу. Зі зростанням рівня загроз у кіберпросторі, необхідність швидко виявляти аномальну активність стала ще критичнішою. Не менш важливим чинником є дотримання численних регуляторних вимог, які передбачають збереження та аналіз логів для аудиту та відповідності стандартам. Нарешті, сучасні методології DevOps і SRE активно впроваджують підходи до проактивного виявлення проблем, де лог-моніторинг відіграє ключову роль.

Метою даного дослідження є розробка системи моніторингу й аналізу лог-файлів у режимі реального часу, яка забезпечить своєчасне виявлення аномалій, збоїв та потенційних загроз в ІТ-інфраструктурі. Проєкт передбачає створення архітектури рішення з урахуванням вимог до продуктивності, масштабованості, інтеграції з DevOps-практиками та можливостей предиктивного аналізу.

У цій сфері вже існує чимало рішень. Наприклад, відкритий стек ELK (Elasticsearch, Logstash, Kibana)[1] дозволяє збирати, індексувати та візуалізувати логи з високою гнучкістю та масштабованістю, хоча й вимагає значних ресурсів.

Комерційна платформа Splunk[2] пропонує потужний функціонал, але є дороговартісною. Graylog зосереджується на безпеці та відповідності, тоді як Datadog[3] та Sumo Logic[4] надають SaaS-рішення з інтеграціями та підтримкою машинного навчання. Однак навіть ці інструменти мають низку обмежень: від високої вартості й складності обслуговування до недостатньої продуктивності при роботі з великими обсягами логів та обмежених можливостей адаптивного аналізу.

З огляду на ці виклики, виникає потреба у створенні нової системи, здатної здійснювати інтелектуальний аналіз логів у режимі реального часу, виявляти приховані взаємозв'язки та аномалії, а також враховувати особливості конкретних сфер застосування. Важливою є також можливість поєднання високої продуктивності з раціональним використанням ресурсів, що дозволить знайти баланс між ефективністю та витратами. Інтеграція з сучасними DevOps-процесами, підтримка предиктивного аналізу та автоматизація реагування на інциденти є ключовими складовими такої системи.

Розробка подібного рішення дозволить значно підвищити надійність ІТ-інфраструктури, мінімізувати простої, підвищити ефективність використання ресурсів та зміцнити захист від кіберзагроз.

РОЗДІЛ 1

ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Характеристика предметної галузі та об'єкта дослідження

У сучасних інформаційних системах щодня генеруються великі обсяги лог-файлів - текстових записів, які фіксують події, операції, помилки та інші дії, що відбуваються в програмному та апаратному забезпеченні. Наприклад, коли користувач входить у систему, сервер відповідає на запит або виникає помилка у додатку всі ці події можуть бути зафіксовані у відповідних логах. Однак самотійний перегляд таких логів вручну стає малоефективним через їхній обсяг, складність та динамічний характер змін.

Системи моніторингу в реальному часі дозволяють адміністраторам виявляти потенційні проблеми до того, як вони вплинуть на користувачів, такі як уповільнення через збільшене навантаження або витoki пам'яті[5]. Моніторинг в реальному часі надає користувачам миттєвий доступ до інформації про активність системи та можливі проблеми в міру їх виникнення[6].

Лог-файли містять інформацію про події, що відбуваються в різних компонентах інфраструктури, включаючи операційні системи, сервери, бази даних, мережеве обладнання, прикладне програмне забезпечення та системи безпеки. Аналіз цих даних допомагає виявляти тренди, патерни поведінки, аномалії та відхилення від нормальної роботи систем. Вони можуть надходити з різноманітних компонентів системи, таких як операційні системи, що фіксують події запуску, збої та оновлення; веб-сервери, які реєструють запити клієнтів, коди помилок і час відповіді; бази даних, де відображаються запити, блокування та помилки; мережеве обладнання, яке відстежує активність, трафік і збої з'єднання; додатки, що фіксують використання функцій і помилки користувача;

а також системи безпеки, які ведуть облік входів у систему, спроб злому та політик доступу[7].

Централізоване логування дозволяє контролювати кілька джерел логів одночасно через збір та транспортування логів з використанням API або шляхом налаштування окремих систем логування для прямого запису в централізований менеджер логів[8].

Основна мета інформаційної системи полягає в автоматизованому збиранні, обробці, аналізі та візуалізації даних з лог-файлів різних систем з метою виявлення проблем, аномалій, загроз безпеці та оптимізації роботи інфраструктури. Такі системи використовуються в різних сферах: від забезпечення безпеки та стабільності веб-додатків до аналізу роботи мереж, серверів і хмарних сервісів. Вони дозволяють миттєво виявляти проблеми, реагувати на інциденти, а також здійснювати історичний аналіз для вдосконалення систем. Основні функції системи охоплюють:

- Збір та агрегація лог-даних.
- Обробка та парсинг логів.
- Аналіз та виявлення аномалій.
- Моніторинг продуктивності та доступності.
- Оповіщення та реагування на інциденти.
- Візуалізація та звітність.
- Аналіз безпеки.

Розглянемо кожен пункт більш детально. Збір та агрегація лог-даних є відправною точкою, де система повинна автоматично приймати логи з різних джерел, таких як сервери, додатки чи бази даних. Ці дані можуть надходити в різних форматах, тому на цьому етапі важливо забезпечити стабільне підключення до джерел, контроль обсягу інформації та її попередню фільтрацію або нормалізацію. Логи мають оброблятися швидко і без втрат, що вимагає ефективної організації черг обробки. Сучасні інструменти можуть обробляти

складну маршрутизацію, буферизацію та фільтрування логів перед доставкою. FluentD використовується для збору логів з понад 50,000 серверів, тоді як ефективний дизайн Fluent Bit дозволяє обробляти високу пропускну здатність навіть на скромному обладнанні[9].

Далі йде етап обробки та парсингу логів, де кожен лог-запис повинен бути структуровано. Це означає виділення ключових полів, наприклад, часу події, IP-адреси, рівня важливості чи повідомлення про помилку. Парсинг дозволяє привести неструктуровані записи до єдиного формату, що суттєво полегшує подальший аналіз. Унікальні рядкові значення можуть використовуватися для звуження пошуку, що особливо важливо в контейнерних системах[10].

Наступним кроком є аналіз та виявлення аномалій. Система повинна постійно контролювати потік подій, виявляючи відхилення від звичайної поведінки. Це може бути раптове збільшення кількості помилок, нехарактерні дії користувачів або повторювані збої. Аномалії можуть вказувати на технічні несправності або спроби атаки, тому виявлення таких випадків має відбуватись із мінімальною затримкою. Автоматичний аналіз лог-файлів забезпечує раннє виявлення відповідних інцидентів, таких як системні збої. Зокрема, самонавчальні методи виявлення аномалій захоплюють патерни в даних логів і надалі повідомляють операторам системи про неочікувані випадки подій логів без необхідності їх додаткового налаштування[11].

Моніторинг продуктивності та доступності є ще одним важливим компонентом. Система повинна відображати в реальному часі, наскільки стабільно працюють сервіси, які ресурси споживаються та які компоненти перевантажені або недоступні. Це дозволяє операторам швидко реагувати на збої, переглядати історію завантаженості й приймати обґрунтовані рішення щодо масштабування або оптимізації інфраструктури. Моніторинг логів в реальному часі надає аналітикам інформацію, необхідну для реагування на потенційні атаки

та їх зупинки до того, як вони стануть критичним порушенням даних, що впливає на фінансові рахунки клієнтів[12]

Система також повинна підтримувати оповіщення та реагування на інциденти. Як тільки фіксується потенційна проблема або порушення встановлених правил, адміністратори повинні миттєво отримувати повідомлення.

Візуалізація та звітність дозволяють представити зібрані та оброблені дані у зручному для сприйняття вигляді. Інтерактивні дашборди, графіки, часові лінії та таблиці дають змогу швидко зрозуміти поточну ситуацію, аналізувати історичні дані та будувати прогнози.

Останнім завданням є аналіз безпеки, який полягає у виявленні потенційних загроз, атак або спроб несанкціонованого доступу. Це вимагає аналізу як технічних подій (наприклад, невдалих спроб входу або спроб змінити конфігурацію), так і поведінкових патернів.

Переваги та виклики

- **Переваги:**
 - Безперервний моніторинг.
 - Автоматизація реагування.
 - Безперервний моніторинг.
- **Виклики:**
 - Складність масштабування.
 - Обробка великого обсягу даних.
 - Зберігання та архівація.

Системи моніторингу лог-файлів є надзвичайно універсальними та знаходять своє застосування в широкому спектрі галузей, де вони відіграють ключову роль у забезпеченні безпеки, продуктивності та відповідності вимогам. У сфері фінансових послуг, такі системи є незамінними для відстеження

транзакцій у реальному часі, що дозволяє оперативно виявляти та запобігати шахрайським діям. Крім того, вони допомагають фінансовим установам відповідати суворим регуляторним нормам та вимогам.

В електронній комерції моніторинг лог-файлів дає можливість глибоко аналізувати поведінку користувачів на вебсайтах та в додатках. Це дозволяє оптимізувати користувацький досвід, покращувати функціональність платформ та підвищувати загальну продуктивність систем, що безпосередньо впливає на задоволеність клієнтів та комерційний успіх.[13]

Для телекомунікаційних компаній ці системи є критично важливими для постійного моніторингу мережевого обладнання та якості зв'язку. Вони допомагають швидко виявляти й усувати збої, забезпечуючи стабільну роботу мереж та високу якість послуг для абонентів.

У галузі охорони здоров'я системи моніторингу лог-файлів відіграють життєво важливу роль у захисті конфіденційних медичних даних пацієнтів, гарантуючи їхню безпеку та цілісність. Це також допомагає медичним установам відповідати суворим вимогам, що є обов'язковим для збереження довіри пацієнтів та уникнення юридичних наслідків.

У державному секторі моніторинг лог-файлів є невід'ємною частиною забезпечення безпеки критичної інфраструктури. Ці системи допомагають захищати життєво важливі державні об'єкти та інформаційні системи від кібератак та інших загроз, що має вирішальне значення для національної безпеки.

Загалом ця інформаційна система повинна надавати ІТ-адміністраторам, фахівцям з безпеки та керівникам ІТ-відділів інструменти для ефективного управління інфраструктурою, швидкого реагування на інциденти та прийняття обґрунтованих рішень на основі аналізу даних.

1.2 Аналіз літературних джерел та практичного досвіду використання ІС і технологій в предметній галузі

Сьогодні використання інформаційних систем і технологій у сфері моніторингу та аналізу лог-файлів охоплює широкий спектр застосувань, починаючи від простих, але ефективних інструментів для базового аналізу логів, і завершуючи складними системами управління подіями безпеки (SIEM) [14]. Ці передові платформи не тільки збирають і зберігають лог-дані, але й надають потужні можливості для їх кореляції, аналізу та виявлення потенційних загроз безпеці. Протягом останніх десятиліть розвиток цих технологій значно сприяв підвищенню ефективності моніторингу ІТ-інфраструктури та забезпеченню інформаційної безпеки організацій будь-якого масштабу. Це дозволяє компаніям швидше реагувати на інциденти, мінімізувати ризики та підтримувати стабільну роботу своїх систем.

Важливо розуміти різницю між аналізом логів та моніторингом: аналіз логів зосереджується на дослідженні історичних даних для виявлення закономірностей, усунення неполадок та отримання інсайтів, тоді як моніторинг логів підкреслює спостереження за поведінкою системи в режимі реального часу, негайне сповіщення та постійні перевірки стану системи. Більшість сучасних інструментів поєднують обидві можливості[15].

Аналіз різних літературних джерел свідчить про широкий спектр теорій та підходів до використання ІС у сфері аналізу лог-файлів. Деякі дослідники підкреслюють роль технологій машинного навчання у підвищенні ефективності виявлення аномалій та автоматизації процесів аналізу великих обсягів логових даних.

Сучасні дослідження показують, що майже 85% компаній досліджують технології виявлення аномалій для промислових аномалій зображень[16]. Алгоритми машинного навчання покращують SIEM-рішення, дозволяючи їм

навчатися на даних, виявляти закономірності та виявляти аномалії з більшою точністю[17].

На відміну від ручного аналізу, який є часозатратним і часто непрактичним для великих наборів даних, системи з підтримкою ШІ відмінно справляються з просіюванням величезних обсягів даних, виявляючи складні закономірності та викиди з неймовірною точністю[18].

Усвідомлення проблем моніторингу лог-файлів полягає в необхідності комплексного підходу до аналізу логових даних. Це означає, що недостатньо просто збирати логи; необхідно їх систематизувати, аналізувати у контексті інших подій та інтерпретувати для отримання глибоких інсайтів. Хоча сучасні технології, безперечно, можуть значно полегшити процеси збору та обробки логів, їх ефективність безпосередньо залежить від ряду ключових факторів. Ці системи повинні забезпечувати високу продуктивність, щоб обробляти терабайти даних без затримок, надійність, гарантуючи безперебійний збір та зберігання інформації, та, що особливо важливо, можливість інтеграції з існуючими ІТ-системами. Без належної інтеграції системи моніторингу логів залишатимуться ізольованими, що ускладнить їх функціонування в загальній ІТ-екосистемі. Крім того, життєво важливо розвивати зручні та інтуїтивно зрозумілі інтерфейси для адміністраторів та аналітиків, які щоденно працюють з цими системами. Адже навіть найпотужніша технологія буде марною, якщо користувачі не зможуть ефективно нею користуватися та інтерпретувати отримані дані.

Аналізуючи основні категорії та поняття у цій галузі, ми стикаємося з широким списком термінів, що відображають складність та багатогранність моніторингу та аналізу лог-файлів. До них відносяться: потокова обробка даних, що дозволяє аналізувати інформацію в міру її надходження; аналіз в режимі реального часу, який є критично важливим для оперативного виявлення інцидентів безпеки та збоїв; виявлення аномалій, що є центральним елементом будь-якої системи моніторингу; агрегація логів, що полягає у зборі та об'єднанні

логів з різних джерел; індексування та пошук, які забезпечують швидкий доступ до конкретних подій у величезних сховищах даних; візуалізація даних, що перетворює складні числові дані на зрозумілі графічні представлення для полегшення аналізу (наприклад, за допомогою Kibana [19]), та багато інших.. Важливими питаннями для подальшого вивчення та розвитку є: підвищення швидкості обробки великих обсягів даних, що є постійним викликом у зв'язку зі зростанням обсягів інформації; забезпечення масштабованості систем для адаптації до динамічних потреб організацій; розробка ще ефективніших алгоритмів виявлення інцидентів, здатних розпізнавати складніші та приховані загрози; та створення ще більш інтуїтивних інтерфейсів для користувачів, що дозволить адміністраторам та аналітикам швидше та ефективніше реагувати на події. Ці напрямки є ключовими для подальшого вдосконалення систем моніторингу та аналізу лог-файлів, забезпечуючи надійний захист та стабільну роботу сучасних ІТ-інфраструктур.

Основна мета цього дослідження - аналіз поточного стану використання інформаційних систем (ІС) та інформаційних технологій (ІТ) у сфері моніторингу та аналізу лог-файлів, що відбувається в режимі реального часу. Це передбачає не лише опис існуючих підходів, а й виявлення ключових аспектів, які безпосередньо впливають на ефективність систем моніторингу. Ми зосереджуємося на тому, як ці аспекти позначаються на швидкості виявлення проблем в ІТ-інфраструктурі, що є критично важливим для безперервної роботи будь-якої організації. Крім того, робота спрямована на визначення можливостей для подальшого розвитку цієї галузі, окреслюючи перспективні напрямки для вдосконалення технологій та методик.

Це дослідження має значне значення та актуальність як для безпосередніх об'єктів дослідження, так і для всієї предметної галузі інформаційних технологій. Насамперед, воно є важливим для організацій, які прагнуть забезпечити ефективний моніторинг своїх ІТ-систем. В умовах зростаючих кіберзагроз та

складності ІТ-інфраструктур, здатність оперативно виявляти та реагувати на аномалії є запорукою стабільності та безпеки.

Результати цього дослідження допоможуть вдосконалити практику використання систем аналізу логів. Це дозволить не тільки підвищити швидкість виявлення та усунення проблем, але й покращити загальний рівень безпеки та стабільність роботи інформаційних систем. Швидке реагування на інциденти мінімізує потенційні збитки та простої.

У 2025 році очікується подальший розвиток відкритих рішень для управління логами та моніторингу. OpenObserve побудований так, щоб бути налаштовуваним, дозволяючи адаптувати його для безшовної інтеграції в існуюче ІТ-середовище. Його потужні можливості інтеграції забезпечують гнучкість у впровадженні[20].

Крім того, аналіз поточного стану та визначення перспектив розвитку ІТ у цій галузі сприятиме підвищенню якості ІТ-сервісів в цілому. Розуміння того, куди рухаються технології моніторингу логів, дасть можливість організаціям планувати свої інвестиції та стратегії, що, в свою чергу, призведе до зменшення часу простою критично важливих систем та підвищення їхньої надійності. Це безпосередньо впливає на безперервність бізнес-процесів та фінансову стійкість підприємств.

Розглянемо існуючі рішення, такі як Kibana та OpenSearch, які є лідерами в цій галузі.

1. Kibana[1]

Kibana - це потужна веб-платформа візуалізації та дослідження даних, яка є частиною Elastic Stack (ELK Stack), що включає Elasticsearch, Logstash та Kibana. Вона дозволяє користувачам інтерактивно досліджувати, візуалізувати та ділитися даними, що зберігаються в Elasticsearch.

Переваги:

- Візуалізація даних: Широкий спектр візуалізацій, включаючи гістограми, лінійні графіки, кругові діаграми, карти, таблиці та багато іншого.
- Інтерактивні дашборди: Можливість створювати кастомізовані дашборди для агрегованого перегляду даних з різних джерел.
- Пошук та фільтрація: Потужні можливості пошуку та фільтрації даних за допомогою мови запитів Lucene.
- Машинне навчання (Machine Learning): Дозволяє виявляти аномалії та тренди в даних.

Недоліки:

- Вимоги до ресурсів: Може бути ресурсоємною, особливо при великих обсягах даних.
- Крива навчання: Початківцям може знадобитися час для освоєння всіх можливостей.
- Ліцензування: Деякі розширені функції Elastic Stack (включаючи Kibana) вимагають комерційної ліцензії.

2. OpenSearch[29]

OpenSearch - це розподілена пошукова система та аналітичний двигун, а також пов'язаний з ним набір інструментів візуалізації (OpenSearch Dashboards), який є форком Elasticsearch та Kibana. OpenSearch був створений Amazon Web Services (AWS) у відповідь на зміни в ліцензуванні Elastic. Він є відкритим та безкоштовним, під ліцензією Apache 2.0, що робить його привабливою альтернативою для тих, хто шукає повністю відкрите рішення..

Переваги

- Пошук та аналіз: Ефективний пошук та агрегація даних у розподіленому середовищі.
- Моніторинг та оповіщення: Можливості для створення сповіщень на основі даних.
- Машинне навчання (Machine Learning): Інструменти для виявлення аномалій та прогнозування.
- SQL та PPL (Pipe Processing Language): Можливість виконання запитів до даних за допомогою SQL та спеціальної мови PPL.
- Продуктивність: Оптимізований для високої продуктивності та масштабованості.
- Відкритий вихідний код: Повністю відкритий та безкоштовний під ліцензією Apache 2.0.

Недоліки:

- Відносно нова екосистема: Хоча активно розвивається, екосистема може бути меншою порівняно з Elastic Stack.
- Можливо, менше готових інтеграцій: Хоча сумісність висока, деякі специфічні інтеграції можуть бути доступніші для Elastic Stack.
- Залежність від AWS: Хоча є відкритим проектом, AWS є основним контриб'ютором.

Таблиця 1.1 – Порівняння існуючих ІС

Характеристика	Kibana	OpenSearch Dashboards	Система даного проекту
Тип ліцензії	Частково платна (SSPL)	Повністю безкоштовна (Apache)	Повністю безкоштовна

Характеристика	Kibana	OpenSearch Dashboards	Система даного проекту
Інтеграція з Elasticsearch	Глибока	Повна	Може інтегруватися частково
Підтримка алертів	Так (частково платно)	Так (безкоштовно)	Так (безкоштовно)
Машинне навчання	Так (тільки платна версія)	Обмежена	Ні
Гнучкість інтерфейсу	Висока	Висока	Середня
Стрімінг логів у реальному часі	Через Logstash/Beats	Через Logstash/Beats	Через WebSocket
Легкість розгортання	Складна (багато компонентів)	Складна	Легка (1-2 сервіси)
Потреба в ресурсах	Висока	Висока	Низька
Масштабованість	Висока	Висока	Обмежена
Підтримка користувачів	Є рольова модель	Є рольова модель	Є рольова модель
Плагіни / розширення	Так (офіційні та спільноти)	Так (менше плагінів)	Ні або вручну
Пошук за логами	Lucene DSL	Lucene / SQL	Простий фільтр або Regex

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Результати порівняльного дослідження наявних рішень дозволяють сформулювати наступні основні висновки.

Створена нами інформаційна система показує високу конкурентність відносно комерційних продуктів завдяки легкості впровадження, адаптивним можливостям налаштування та здатності підлаштовуватися під специфічні вимоги. Система показує відмінні результати в критично важливих сферах: продуктивність процесингу інформації, комфортність потокового аналізу, оперативність відгуку на інциденти та можливості з'єднання з зовнішніми сервісами через WebSocket API.

Kibana вражає багатим арсеналом інструментів, потужними засобами візуалізації та тісною взаємодією з Elasticsearch, що робить її ідеальним вибором для великих підприємств. Проте, складність налаштування та частково комерційна ліцензійна політика можуть створювати бар'єри для малих колективів або початкових проєктів.

OpenSearch Dashboards виділяється як потужна безплатна open source платформа з підтримкою передових можливостей log-аналізу, SQL-запитів та механізмами сповіщень. Активна підтримка від спільноти розробників та Amazon Web Services дає змогу проєктам обходити ліцензійні обмеження Elastic Stack.

Базуючись на отриманих результатах, ми визначили пріоритетні напрямки розвитку нашого рішення:

- інтеграція машинного навчання;
- розробка інтуїтивного конструктора панелей управління;
- додавання базових аналітичних функцій;
- покращення масштабованості та оптимізація для роботи з великими датасетами.

Підсумовуючи, зазначимо, що запропоноване в даному проекті рішення представляє собою ефективний та адаптивний інструмент для компаній, які потребують простого у впровадженні продукту з потенціалом для подальшого налаштування відповідно до корпоративних вимог.

РОЗДІЛ 2

РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Аналіз і специфікація вимог до інформаційної системи

Процес аналізу та формування вимог до системи, яка забезпечує моніторинг і обробку лог-файлів у реальному часі, має на меті створення чіткого розуміння очікувань користувачів, цілей бізнесу та необхідного функціоналу для їх досягнення.

Вимоги збираються шляхом проведення інтерв'ю та опитувань основних зацікавлених осіб системних адміністраторів, розробників, спеціалістів із безпеки, а також через аналіз існуючих рішень для логування й моніторингу. Особливу увагу приділяють визначенню ключових сценаріїв використання системи, зокрема для виявлення аномалій, аналізу продуктивності та аудиту безпекових подій.

На етапі аналізу вимоги класифікуються на функціональні та нефункціональні. Функціональні вимоги охоплюють можливості збору логів з різних джерел, обробки та нормалізації даних, пошуку, фільтрації, візуалізації у вигляді дашбордів, а також налаштування сповіщень. Нефункціональні стосуються продуктивності системи (наприклад, обробка великих обсягів даних), безпеки (захист інформації), масштабованості, зручності у використанні та відповідності нормативним стандартам. Також проводиться перевірка вимог на повноту, несуперечність і реалістичність, виявляються потенційні конфлікти й розставляються пріоритети з урахуванням важливості для бізнесу.

Нефункціональні вимоги включають обмеження на затримку обробки (наприклад, не більше N секунд), заходи безпеки (автентифікація, шифрування), обробку великих обсягів даних.

Для ефективного вирішення завдань з ведення логів і моніторингу важливо враховувати кілька ключових аспектів. Передусім слід чітко визначити мету логування, щоб уникнути зайвого інформаційного шуму та оптимізувати обсяг зібраних даних [21]. Логи мають бути структурованими — із використанням формату JSON, пар ключ–значення, а також містити чіткі часові мітки, ідентифікатори користувачів і IP-адреси, що спрощує подальший аналіз[22].

Фінальний етап передбачає погодження вимог із усіма зацікавленими сторонами, що гарантує їх відповідність очікуванням і підтверджує офіційне ухвалення. У разі потреби забезпечується управління змінами, аналіз їхнього впливу на існуючу архітектуру та план розробки, а також належна комунікація щодо змін зі всіма учасниками процесу.

2.1.1 Бізнес-вимоги до системи

Бізнес-вимоги — це специфікації, які окреслюють те, що потрібно бізнесу від інформаційної системи чи рішення. Вони часто містять опис функцій і функцій, а також будь-які технічні обмеження. Бізнес-вимоги дають чітке розуміння того, що необхідно для задоволення очікувань клієнтів, а також скільки часу та грошей потрібно виділити на проект. Основне питання на яке відповідають бізнес вимоги це чому організації, вам чи клієнту потрібна система про яку говориться.[23]

В контексті системи визначення бізнес-вимог є критично важливим, оскільки дозволяє чітко визначити цілі й очікування від системи, а також узгодити бачення всіх зацікавлених сторін. Це забезпечує розуміння того, які

функції є справді необхідними для досягнення бізнес-результатів, такі як своєчасне виявлення збоїв, підвищення безпеки чи зменшення часу простою. Завдяки сформульованим вимогам можна ефективно пріоритезувати функціонал, уникнути непотрібних витрат на переробки та забезпечити оцінку ефективності вже впровадженого рішення. У результаті, бізнес-вимоги стають основою для прийняття обґрунтованих рішень під час розробки та впровадження системи.

Система повинна забезпечувати оперативне виявлення аномалій, збоїв і загроз безпеці шляхом автоматизованого збору, обробки та візуалізації логів з різних джерел, з можливістю налаштування оповіщень і гнучкої фільтрації для прийняття своєчасних управлінських рішень.

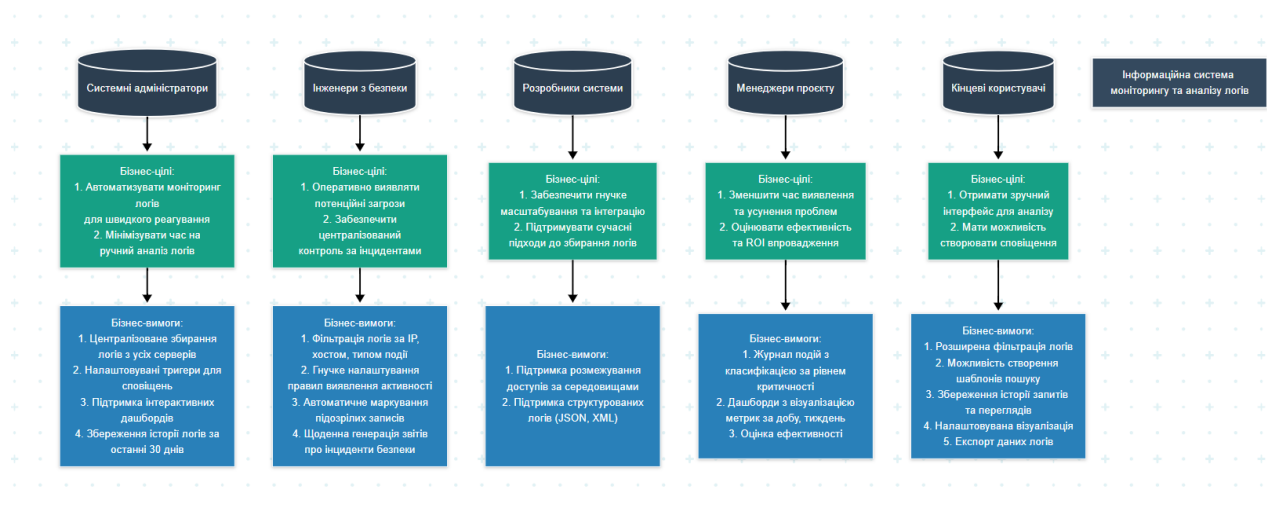


Рисунок 2.1 - Діаграма бізнес-вимог

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.1.2. Функціональні вимоги до системи

Функціональні вимоги - це формулювання того, як система повинна поводитися. Він визначає, що повинна робити система, щоб задовольнити потреби або очікування користувача. Функціональні вимоги можна розглядати як функції, які виявляє користувач. Вони відрізняються від нефункціональних вимог, які визначають, як система повинна працювати всередині (наприклад,

продуктивність, безпека тощо). Вони формують технічний фундамент, завдяки якому можна вчасно виявити збої, вивчити причини інцидентів, забезпечити аудит, а також отримати аналітику в режимі реального часу. Нижче розглянемо ключові функціональні вимоги, які мають бути реалізовані в подібній системі.[24]

Однією з основ системи є централізоване збирання логів. Уявімо компанію, яка має десятки серверів, розташованих у різних дата-центрах. Якщо логи залишаються на кожному сервері окремо, процес пошуку проблеми може затягнутися на години або навіть дні. Це значно пришвидшує діагностику проблем і полегшує роботу команди підтримки.

Наступною важливою вимогою є зберігання історії логів. Неможливо ефективно аналізувати інциденти, що сталися кілька тижнів тому, якщо логи не збережені. Саме тому важливо зберігати історію принаймні за останні 30 днів. Це дає змогу проводити ретроспективний аналіз, підтверджувати або спростовувати гіпотези, а також відповідати на запити внутрішнього або зовнішнього аудиту.

Третя вимога - це можливість фільтрації логів. Без ефективної фільтрації аналітик буде змушений переглядати тисячі рядків вручну, що є непрактичним і неефективним. Наявність гнучкої фільтрації дозволяє миттєво звужувати пошук до потрібних подій і працювати з ними максимально ефективно.

Ще один елемент, який значно підвищує ефективність роботи з логами, - це створення шаблонів запитів. Коли інженери або аналітики регулярно перевіряють однакові умови або події, їм не потрібно щоразу вручну формулювати запити. Вони можуть використовувати заздалегідь підготовлені шаблони, які спрощують роботу і дозволяють стандартизувати підходи до моніторингу.

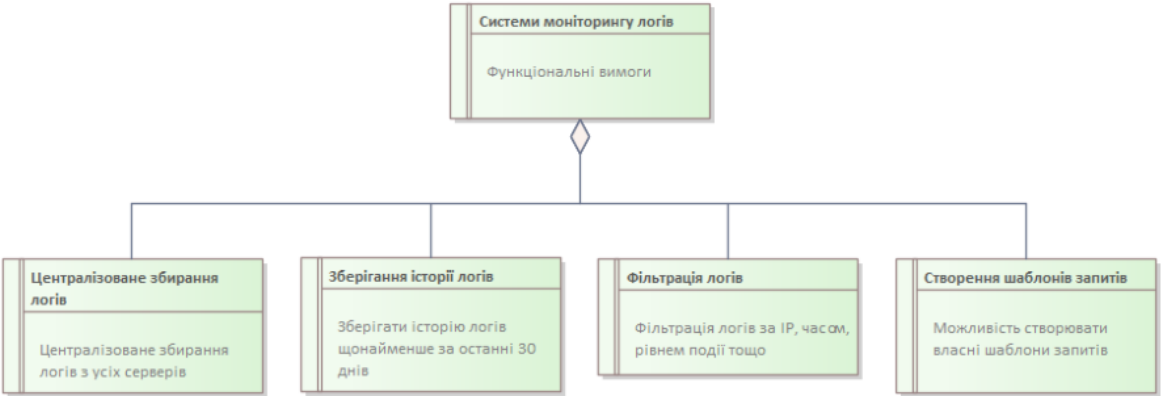


Рисунок 2.2 - Діаграма функціональних вимог

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

☒ Системи моніторингу логів	Functional	Mandatory	High	High
Функціональні вимоги				
☒ Створення шаблонів запитів	Functional	Proposed	High	Medium
Можливість створювати власні шаблони запитів				
☒ Централізоване збирання логів	Functional	Approved	High	High
Централізоване збирання логів з усіх серверів				
☒ Зберігання історії логів	Functional	Approved	High	Medium
Зберігати історію логів щонайменше за останні 30 днів				
☒ Фільтрація логів	Functional	Implemented	Medium	High
Фільтрація логів за IP, часом, рівнем події тощо				

Рисунок 2.3 - Специфікація функціональних вимог

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.1.3. Нефункціональні вимоги до системи

Нефункціональні вимоги це обмеження, накладені на систему, які визначають її атрибути якості. Зазвичай вони позначаються такими прикметниками, як безпека, продуктивність і масштабованість. Нefункціональні

вимоги важливі, оскільки вони допомагають забезпечити відповідність системи потребам користувача.[25]

Нефункціональні вимоги є критичними для розробки системи, оскільки вони визначають її ефективність, надійність та здатність підтримувати високий рівень продуктивності навіть при великих обсягах даних або змінюваних умовах експлуатації. Вони дозволяють зрозуміти, як система повинна вести себе в умовах реального використання, а також забезпечують високий рівень користувацького досвіду.

Масштабованість є одним з основних нефункціональних вимог, оскільки система повинна здатись до розширення без перерви в роботі сервісу. Масштабованість забезпечує можливість адаптації системи до зростаючих потреб без необхідності її переробки чи зупинки, що є критично важливим для застосувань, що функціонують у реальному часі.

Швидка відповідь на запит є ще однією важливою вимогою. Відповідь на запит повинна здійснюватися протягом двох секунд, оскільки в умовах реального часу, де потрібно оперативно реагувати на події та аналізувати великі обсяги даних, будь-які затримки можуть призвести до неприпустимих наслідків.

Відновлення після збоїв також є важливим елементом, оскільки система повинна мати надійний механізм для резервного зберігання логів. Це включає забезпечення надійного резервного копіювання та відновлення логів, що дозволяє зберігати цілісність даних навіть в умовах збоїв



Рисунок 2.4 – Діаграма нефункціональних вимог

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

☒ Системи моніторингу логів	Functional	Mandatory	Medium	Medium
Нефункціональні вимоги				
☒ Масштабованість	Functional	Approved	High	Medium
Можливість розширення логіки без зупинки сервісу				
☒ Швидка відповідь на запит	Functional	Implemented	High	Medium
Відповідь на запит до 2 секунд				
☒ Відновлення після збоїв	Functional	Validated	High	Low
Наявність механізму для резервного зберігання логів				

Рисунок 2.5 – Специфікація нефункціональних вимог

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.2 Постановка задачі та алгоритм розв’язання задачі

Основна задача проєкту полягає у створенні комплексної інформаційної системи моніторингу та аналізу лог-файлів в режимі реального часу, яка забезпечує автоматизоване збирання, обробку, аналіз та візуалізацію системних подій і логів з різних джерел інформаційної інфраструктури.

Забезпечення оперативного виявлення аномалій, помилок, порушень безпеки та продуктивності в ІТ-системах організації є основною метою проєкту. Система повинна надавати можливість своєчасного реагування на критичні події, попереджувати про потенційні проблеми та сприяти підтримці стабільної роботи інформаційної інфраструктури.

Запропонована система повинна забезпечити автоматизоване збирання логів з різних джерел, таких як сервери, програмні додатки, мережеве обладнання. Отримані логи повинні бути уніфіковані, нормалізовані та збережені у централізованому сховищі для подальшого аналізу. Обробка та аналіз логів повинні виконуватись у режимі реального часу, що дозволить виявляти аномалії, шаблони, підозрілі дії та інші події, які можуть свідчити про несправності або загрози. Користувач повинен мати змогу працювати з логами через інтерфейс, що дозволяє перегляд, фільтрацію, пошук та аналіз. У випадку виявлення інцидентів система має оперативно надсилати сповіщення відповідальним особам за допомогою інтегрованої системи нотифікацій.

У довгостроковій перспективі очікується зменшення часу реагування на збої, зниження кількості інцидентів, пов'язаних із людським фактором, підвищення рівня інформаційної безпеки, оптимізація обслуговування інфраструктури та зменшення витрат на технічну підтримку. Також система стане основою для впровадження елементів штучного інтелекту у подальшому.

Проєктування програмного забезпечення передбачає використання сучасних технологій потокової обробки даних, таких як Apache Kafka або Apache Flink, побудову масштабованої та відмовостійкої архітектури, забезпечення високої продуктивності з мінімальною затримкою, інтеграцію з існуючими системами моніторингу, а також впровадження механізмів захисту даних і розмежування прав доступу.

Періодичність розв'язання задачі та видачі інформації повинна бути безперервною: збирання логів здійснюється постійно, їх збереження — у режимі реального часу за допомогою систем потокового запису. Обробка та аналіз даних

також виконуються миттєво після їх надходження. Формування звітів відбувається за запитом користувача або за встановленим графіком — щодня, щотижня чи щомісяця. У разі виявлення інциденту сповіщення повинно надходити негайно.

Автоматизоване розв’язання задачі може бути припинене у випадках критичних збоїв програмного чи апаратного забезпечення, втрати зв’язку з джерелами логів або сховищем, виявлення помилок, що призводять до некоректного аналізу даних, а також при виникненні загроз безпеці, які потребують ручного втручання, наприклад спроби несанкціонованого доступу чи кіберінциденти.

Під час розв’язання задачі здійснюється керування такими об’єктами, як лог-файли з різних джерел (системні, прикладні, мережеві), метадані (джерело, час, рівень події, категорія), правила обробки та фільтрації, події безпеки, що потребують реакції, а також користувачі системи та їх права доступу.

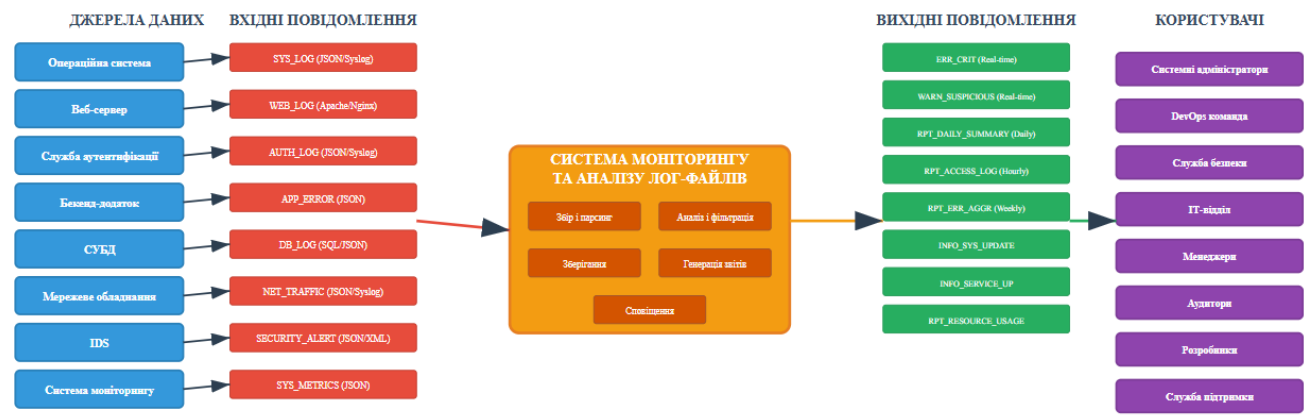


Рисунок 2.6 – Інформаційна модель задачі

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.2.1. Вхідна інформація

Система оперує з широким спектром повідомлень, які надходять з різних джерел, таких як операційна система, веб-сервери, бази даних, системи безпеки та мережеві пристрої. Усі повідомлення мають стандартизовану форму подання,

найчастіше у форматах JSON, Syslog або текстових рядках, що дозволяє їх уніфіковано обробляти. Повідомлення надходять із різною частотою, але здебільшого в режимі реального часу або з мінімальними затримками, що є критично важливим для своєчасного реагування на потенційні інциденти. Завдяки використанню унікальних ідентифікаторів кожного типу повідомлення забезпечується їх ефективна класифікація та фільтрація. Це створює надійну базу для оперативного аналізу подій, виявлення помилок, загроз безпеці та аномальної активності, що в підсумку сприяє підвищенню стабільності та безпеки функціонування всієї IT-інфраструктури.

Таблиця 2.1 – Перелік вхідних повідомлень

Назва вхідного повідомлення	Ідентифікатор	Форма подання	Термін і частота надходження	Джерело
Системний лог	SYS_LOG	Текстовий рядок (JSON або Syslog)	В режимі реального часу, постійно	Операційна система сервера
Лог веб-сервера	WEB_LOG	Текстовий рядок (формат Apache/Nginx)	В режимі реального часу, кожен HTTP-запит	Веб-сервер (Apache/Nginx)
Лог авторизації користувача	AUTH_LOG	JSON-об'єкт або Syslog рядок	При кожній спробі входу в систему	Служба аутентифікації (SSO, PAM)
Повідомлення про	APP_ERROR	JSON із деталями помилки	Відразу після	Бекенд-додаток

помилку додатка			виникнення помилки	
Журнал роботи бази даних	DB_LO G	Текстов ий рядок (SQL або JSON)	При кожному запиті/помилц і/події	СУБД (PostgreSQL, MySQL тощо)
Події мережевого трафіку	NET_TR AFFIC	JSON або Syslog	Постійн о, залежно від навантаження	Маршру тизатори, брандмауери
Логи безпеки/IDS	SECURI TY_ALERT	JSON або XML	У момент виявлення інциденту	Система виявлення вторгнень (IDS)
Події моніторингу ресурсів системи	SYS_M ETRICS	JSON- об'єкти з метриками	Кожні 10–60 секунд	Сервіс моніторингу (Prometheus тощо)

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.2.2. Вихідна інформація

Система охоплює широкий спектр інформаційних подій, які автоматично генеруються на основі стану та поведінки системи. Повідомлення умовно поділяються на три основні групи: аварійні, регулярні звітні, а також інформаційні/системні.

Аварійні повідомлення, формуються та надсилаються негайно після виявлення інциденту. Вони мають найвищий пріоритет і передаються у реальному часі через кілька каналів - веб-інтерфейс, email, месенджерію. Основними користувачами цих повідомлень є системні адміністратори, DevOps-

фахівці та служба безпеки, оскільки саме вони повинні оперативно реагувати на загрози або несправності.

Регулярні звітні повідомлення включають щоденні зведення звіти про доступ до системи. Вони генеруються за розкладом і надсилаються в заданий час - зазвичай у ранкові години, щоб користувачі мали свіжу інформацію на початку робочого дня. Ці повідомлення подаються у зручних форматах (PDF, CSV) і розраховані на ширше коло користувачів менеджерів, аудиторів, аналітиків та системних адміністраторів.

Інформаційні повідомлення відносяться до змін у самій системі, зокрема до оновлень або відновлення сервісів після збоїв. Вони також мають високу оперативність, але носять радше сповіщувальний, а не критичний характер.

Таблиця 2.2 – Перелік вихідних повідомлень

Назва вихідного повідомлення	Ідентифікатор	Форма подання	Періодичність видачі	Термін і час видачі	Користувачі інформації
Повідомлення про критичну помилку	ERR_CRIT	Текстовий файл, Email, Web-інтерфейс	Негайно при виявленні	У реальному часі	Системні адміністратори, DevOps
Попередження про підозрілу активність	WARN_SUSPICIOUS	Веб-інтерфейс	Негайно при виявленні	У реальному часі	Служба безпеки, IT-відділ

Щодо ний звіт активності	RPT_ DAILY_SUMMARY	PDF, Email	Щодо нно	08:00 щодня	Мене джери, аудитори
Звіт про доступ до системи	RPT_ ACCESS_LOG	CSV, Web- інтерфейс	Щого дини	На початку кожної години	Систе мні адміністрат ори
Агрег ований звіт про помилки	RPT_ ERR_AGG R	Excel, Email	Щоти жнево	Поне ділок, 09:00	Аналі тики, розробники
Інфор мація про оновлення системи	INFO_ SYS_UPDATE	Email , Web- інтерфейс	Після кожного оновлення	Упро довж 15 хв після оновлення	Кори стувачі системи, ІТ-відділ
Повід омлення про відновленн я сервісу	INFO_ SERVICE _UP	Email	Негай но	У реальному часі	DevO ps, служба підтримки
Звіт про використан ня ресурсів	RPT_ RESOURCE_USAGE	Графі к (Web), PDF	Щодо нно	07:00 щодня	Систе мні адміністрат ори, DevOps

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.2.3. Алгоритм розв'язання задачі

У системі моніторингу лог-файлів важливо виявляти аномалії, оцінювати навантаження, частоту помилок, та загальну активність системи в режимі реального часу. Основні математичні показники:

1. Інтенсивність логів (log rate):

$$\lambda = N / \Delta t$$

де:

λ - інтенсивність логів (кількість записів за секунду або хвилину),

N - кількість записів логів за період часу Δt ,

Δt - часовий інтервал.

2. Середній час до інциденту (Mean Time To Incident):

$$MTTI = 1 / \lambda_e$$

де:

λ_e - інтенсивність появи інцидентів типу "помилка", "збій".

3. Частка критичних помилок:

$$F_c = (E_c / E_t) \times 100\%$$

де:

F_c - частка критичних помилок,

E_c - кількість критичних помилок,

E_t - загальна кількість усіх помилок.

4. Оцінка аномалії (Anomaly Score):

$$A_s = |X_i - \mu| / \sigma$$

де:

X_i - поточне значення метрики,

μ - середнє значення за історичний період,

σ - стандартне відхилення.

Якщо $A_s > 3$ - вважається аномалією.

5. Рейтинг джерел логів за активністю:

$$R_s = (L_s / L_t) \times 100\%$$

де:

R_s - відсоток логів від джерела s ,

L_s - кількість логів від джерела s ,

L_t - загальна кількість логів.

6. Середній час на відновлення (Mean Time To Repair):

$$MTTR = \sum T_{\text{repair}} / N_{\text{incidents}}$$

де:

T_{repair} - час на відновлення системи після кожного інциденту,

$N_{\text{incidents}}$ - кількість інцидентів.

Система функціонує, починаючи зі збору даних, де спеціальні агенти на серверах постійно надсилають логі в систему. Ці дані потім записуються у сховище даних, як-от Elasticsearch або ClickHouse.

Наступним кроком є попередня обробка, під час якої система парсить логі, щоб витягнути з них найважливіші параметри: часову позначку, рівень важливості назву сервісу та саме повідомлення. На цьому етапі також відбувається фільтрація зайвого шуму та нормалізація форматів повідомлень для одноманітності.

Після цього система переходить до аналізу в реальному часі. Вона обчислює інтенсивність подій, частоту помилок та рівень аномалій. Завдяки цьому можна виявити пікові навантаження та аномалії, використовуючи статистичні методи або моделі машинного навчання.

Результати цього аналізу потім переходять до етапу візуалізації, де ключові метрики відображаються на графіках та дашбордах. Це включає показ частоти помилок, перелік найактивніших сервісів та інформацію про інциденти за останню годину.

Важливим аспектом системи є реагування на події. Коли певні пороги перевищуються, наприклад, значення аномалії перевищує 3, або відсоток помилок зростає понад 20%, система автоматично надсилає сповіщення через email.

2.3 Моделювання інформаційної системи

2.3.1. Моделювання поведінки системи

Моделювання поведінки інформаційної системи дозволяє розробникам і аналітикам чітко уявити функціональність системи, зрозуміти логіку її роботи та передбачити можливі сценарії взаємодії користувачів із системою. У випадку системи моніторингу та аналізу лог-файлів в режимі реального часу поведінкове моделювання є критично важливим через складну архітектуру, потребу в обробці великої кількості даних і високі вимоги до часу реагування.

Для цього використовуються UML-діаграми, зокрема діаграми прецедентів (Use Case), діаграми діяльності, діаграми послідовності та діаграми станів. Вони дозволяють детально описати можливості користувачів, дії системи у відповідь на зовнішні події, а також сценарії роботи в різних умовах.

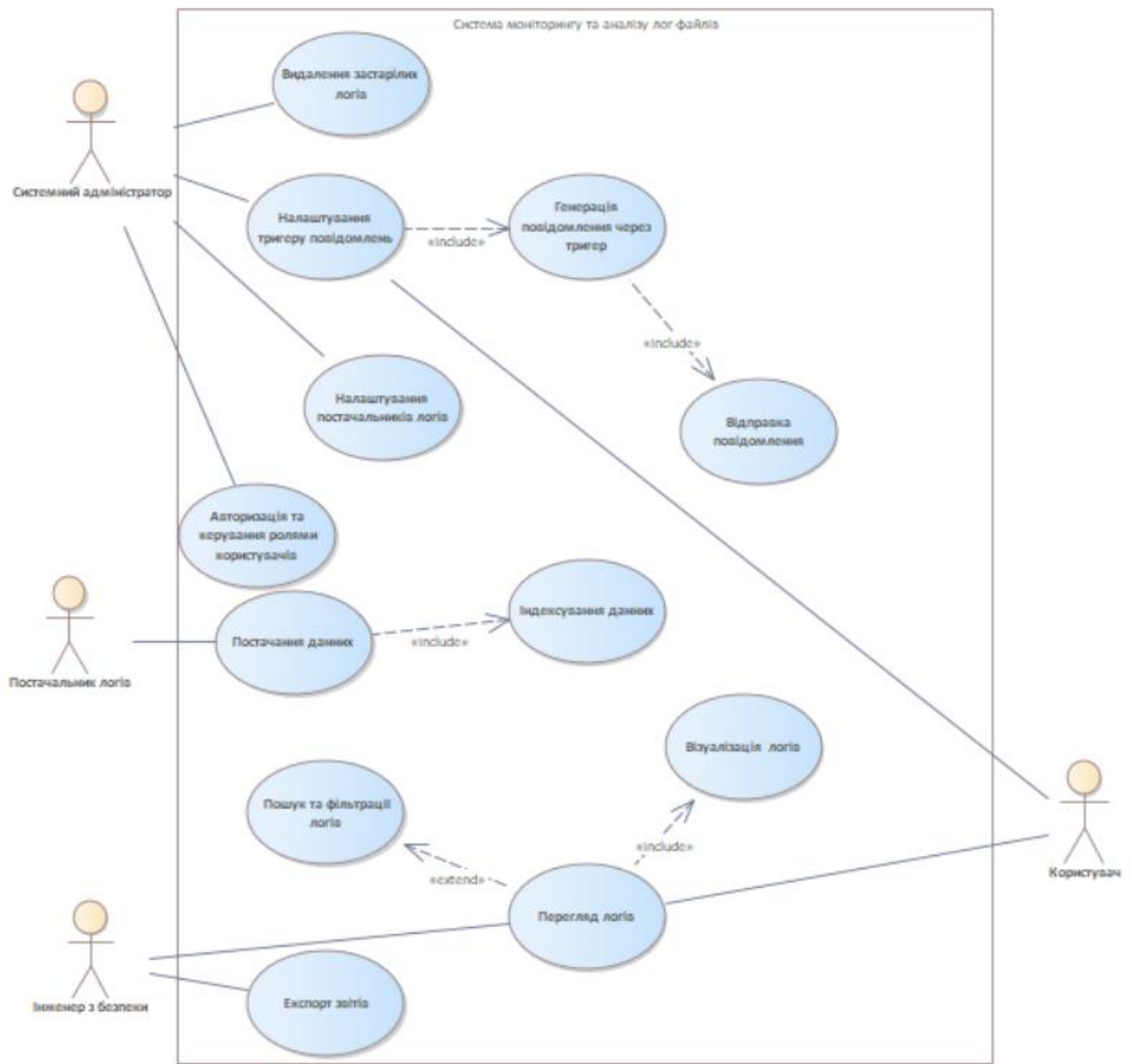


Рисунок 2.7 – Діаграма прецедентів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Виходячи з діаграми вище, до основних акторів системи належать: Системний адміністратор, який має найвищий рівень доступу та відповідальний за загальне налаштування системи, управління користувачами, джерелами логів та конфігурацію сповіщень, Постачальник логів - технічний компонент або зовнішній сервіс, що надсилає лог-файли у систему в реальному часі, Інженер з безпеки, який використовує систему для аналізу інцидентів безпеки, експортує звіти, фіксує підозрілі події, та Користувач, що має обмежений доступ до функціоналу - може переглядати, шукати, фільтрувати та візуалізувати логи.

Для Системного адміністратора передбачені такі прецеденти, як видалення застарілих логів для очищення системи від непотрібних даних, налаштування тригерів повідомлень для визначення умов автоматичного сповіщення про інциденти, генерація повідомлення через тригер як автоматична реакція на визначені події, відправка повідомлень відповідальним користувачам, налаштування постачальників логів та авторизація і керування ролями користувачів.

Для Постачальника логів основними функціями є постачання даних через надсилання лог-файлів у систему з різних джерел та індексація даних для включення логів у пошуковий індекс після надходження.

Для Інженера з безпеки ключовим прецедентом є експорт звітів про події безпеки для подальшого аналізу або архівації.

Для Користувача доступні такі функції, як перегляд логів, пошук та фільтрація логів за допомогою запитів та фільтрів, а також візуалізація логів у вигляді графіків, діаграм або дашбордів.

Target +	Use Cases::Авторизація та керу	Use Cases::Видалення застаріл	Use Cases::Відправка повідомл	Use Cases::Візуалізація логів	Use Cases::Генерація повідомл	Use Cases::Експорт звітів	Use Cases::Індексування даних	Use Cases::Налаштування пост	Use Cases::Налаштування триг	Use Cases::Перегляд логів	Use Cases::Постачання даних	Use Cases::Пошук та фільтрації
+ Source												
Requirements::Зберігання ...				↑				↑				↑
Requirements::Системи мо...		↑				↑						
Requirements::Створення ...				↑						↑		
Requirements::Фільтрація ...										↑		↑
Requirements::Централізо...							↑	↑			↑	

Рисунок 2.8 – Матриця відповідності вимог і прецедентів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Далі наведено приклади поведінки системи

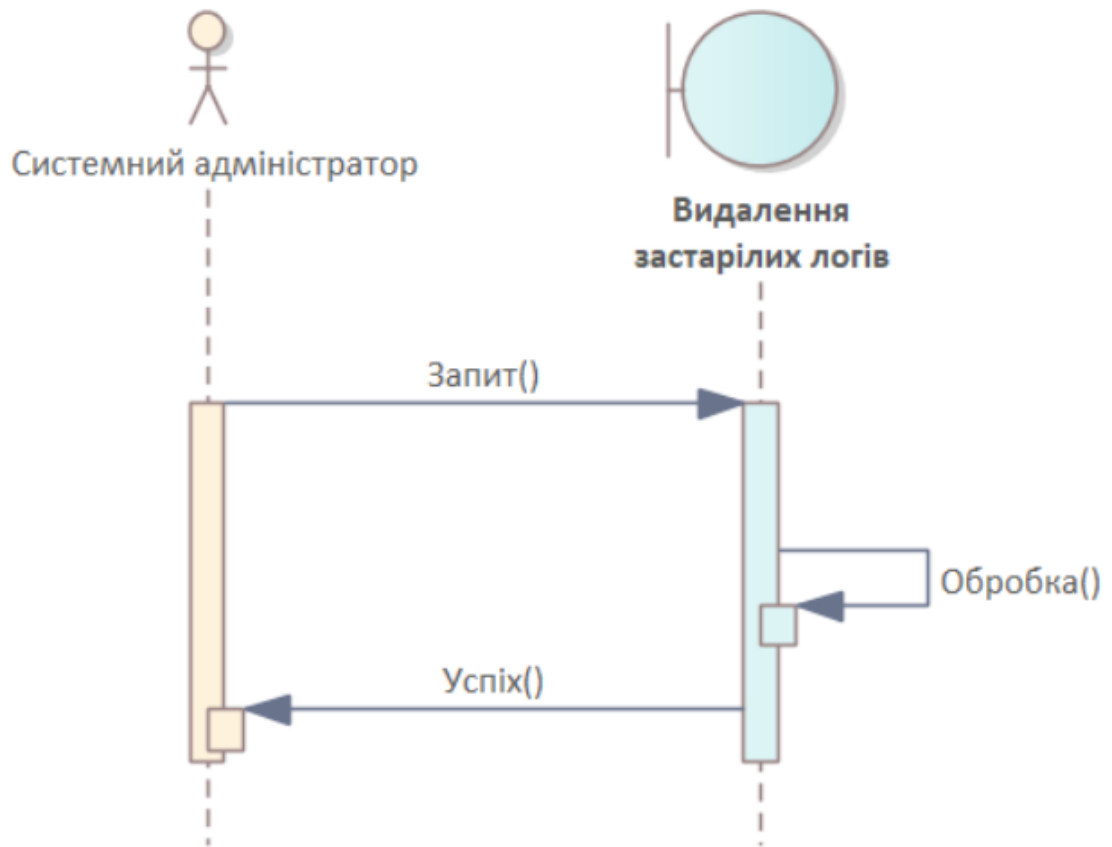


Рисунок 2.9 – Діаграма послідовності Видалення застарілих логів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Ця діаграма відображає показує механізм видалення застарілих логів, який є важливою частиною системи, оскільки дозволяє оптимізувати використання ресурсів та підтримувати продуктивність системи на належному рівні. Адміністратор має змогу ініціювати цей процес, отримуючи підтвердження про успішне виконання.

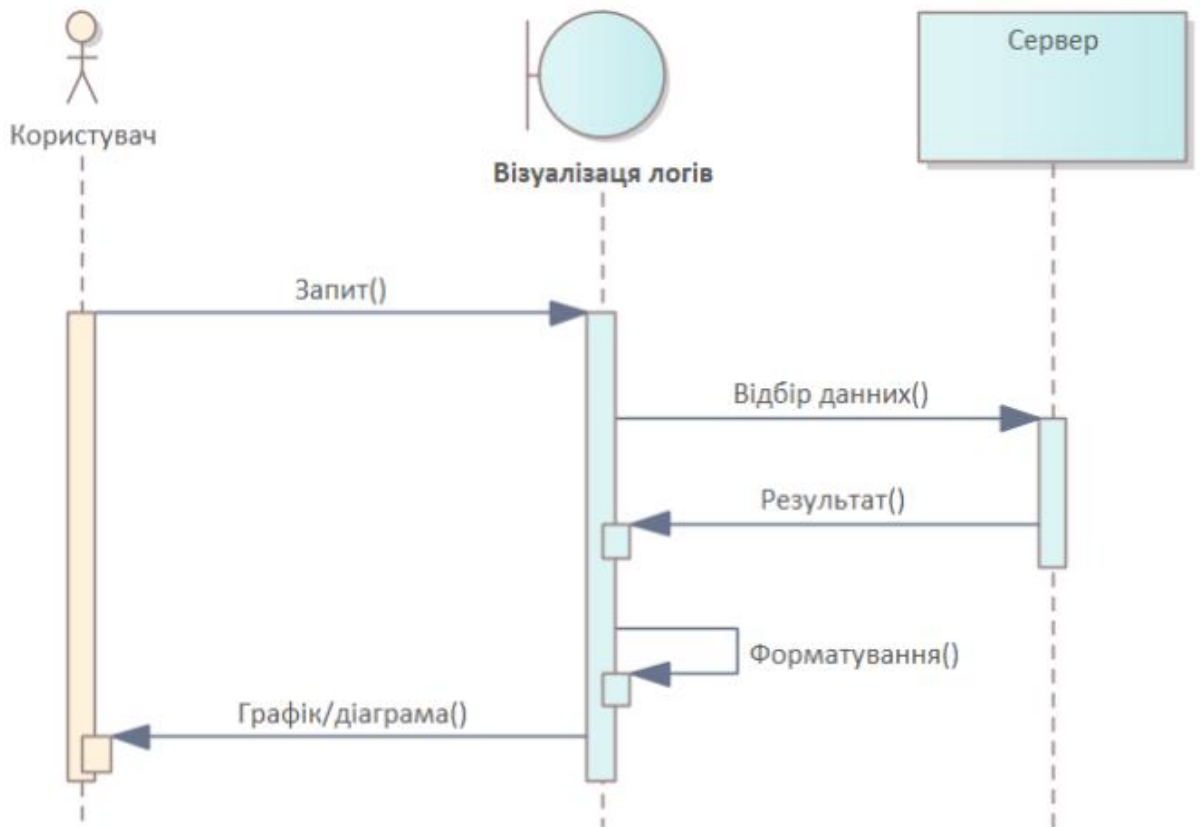


Рисунок 2.10 – Діаграма послідовності Візуалізація логів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

На цій діаграмі користувач ініціює взаємодію, надсилаючи запит до компонента візуалізації логів. Цей центральний компонент відповідає за обробку та відображення даних. Після отримання запиту він звертається до сервера, який зберігає лог-файли та надає до них доступ. Сервер опрацьовує запит і повертає результати пошуку. Отримані дані компонент з візуалізації форматує та на основі них генерує графіки або діаграми. У результаті користувач отримує візуальне представлення аналізу логів.

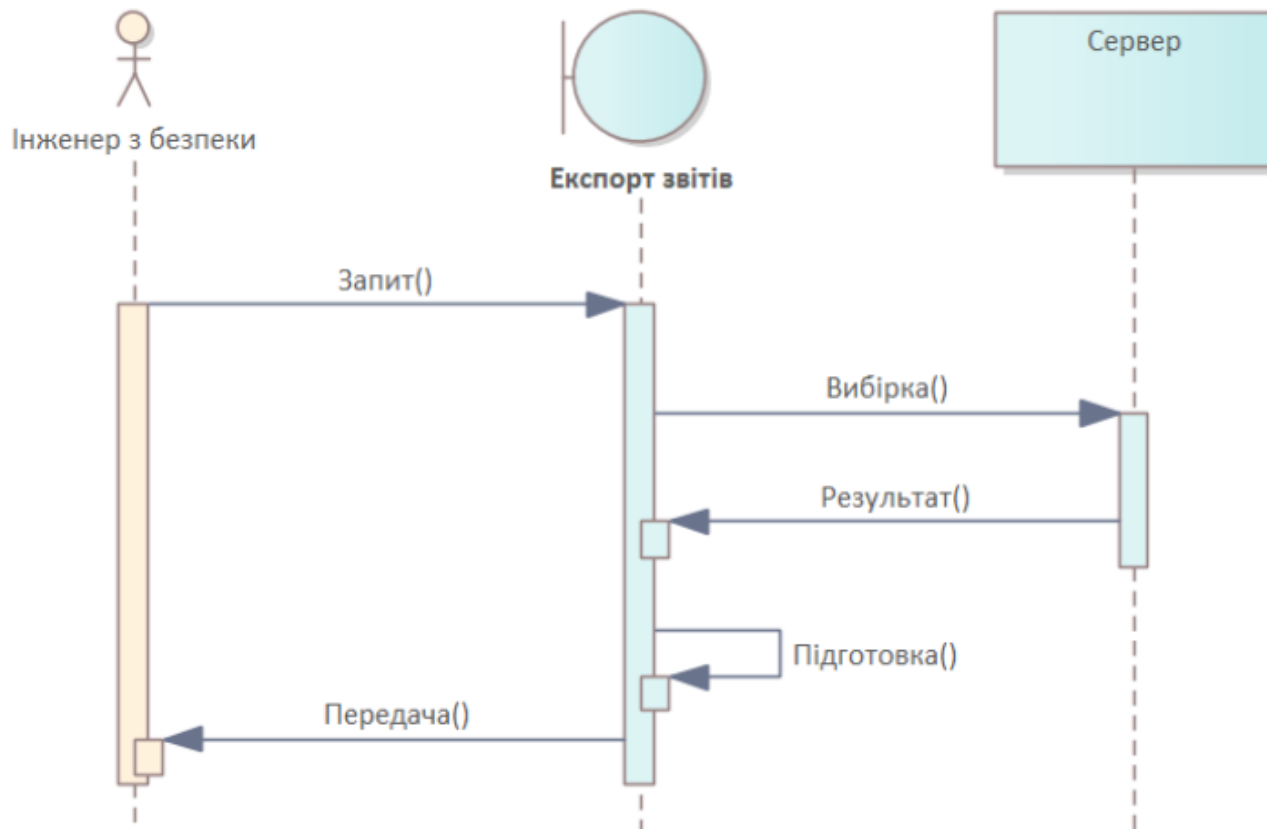


Рисунок 2.11 – Діаграма послідовності експорт звітів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Інженер з безпеки, який є користувачем системи, ініціює процес, відправляючи запит що є частиною інтерфейсу або сервісу. У відповідь на цей запит модуль звертається до сервера, який зберігає та обробляє дані. Сервер повертає результати вибірки, після чого виконує підготовку даних, що може включати форматування, фільтрацію та інші операції. У результаті підготовлені дані передаються інженеру з безпеки.

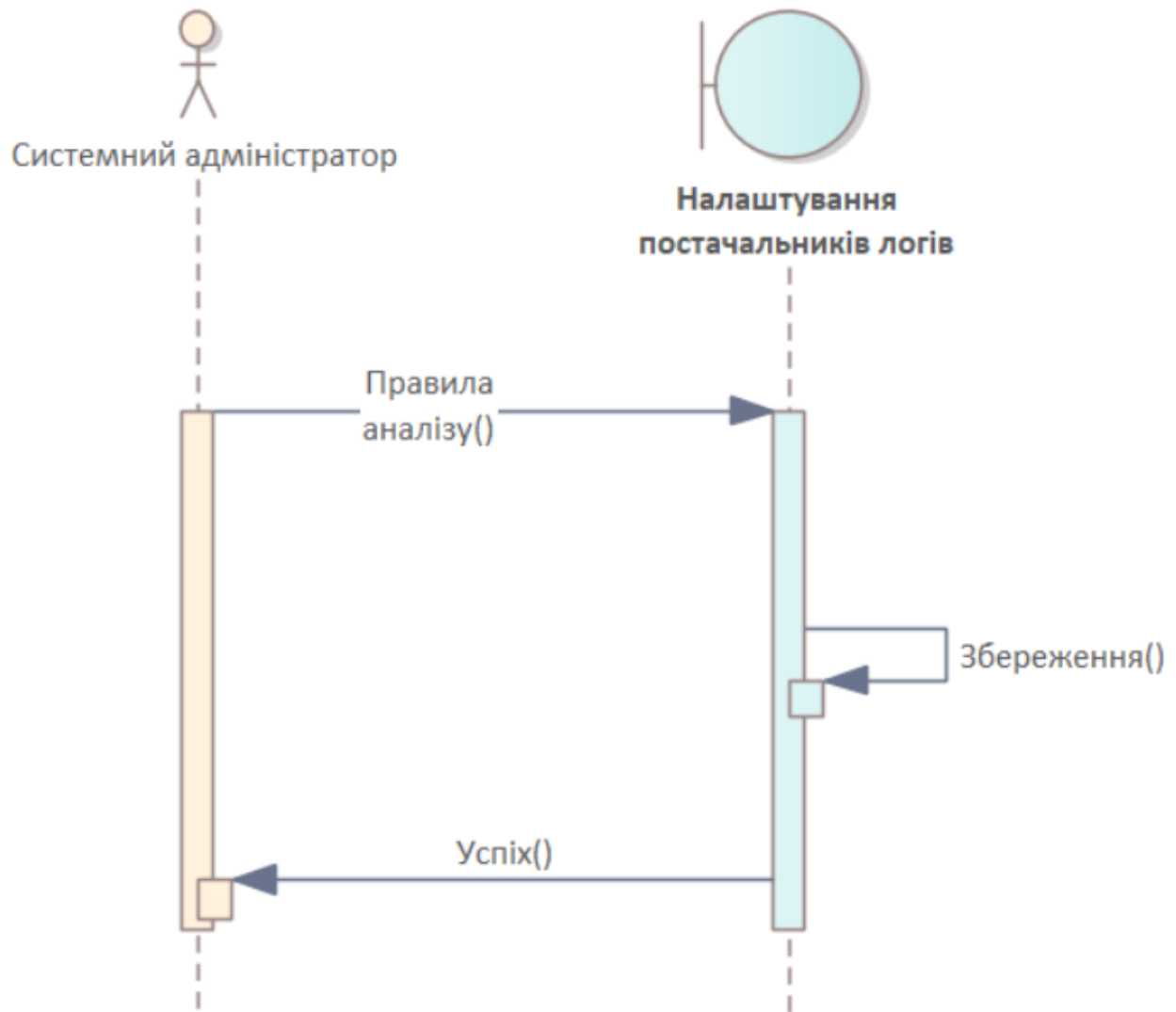


Рисунок 2.12 – Діаграма послідовності Налаштування постачальників логів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Діаграма представляє базову архітектуру систем, де системний адміністратор встановлює правила аналізу, які зберігаються в підсистемі налаштування постачальників логів. Ця послідовність дозволяє конфігурувати систему для моніторингу та аналізу логів у режимі реального часу.

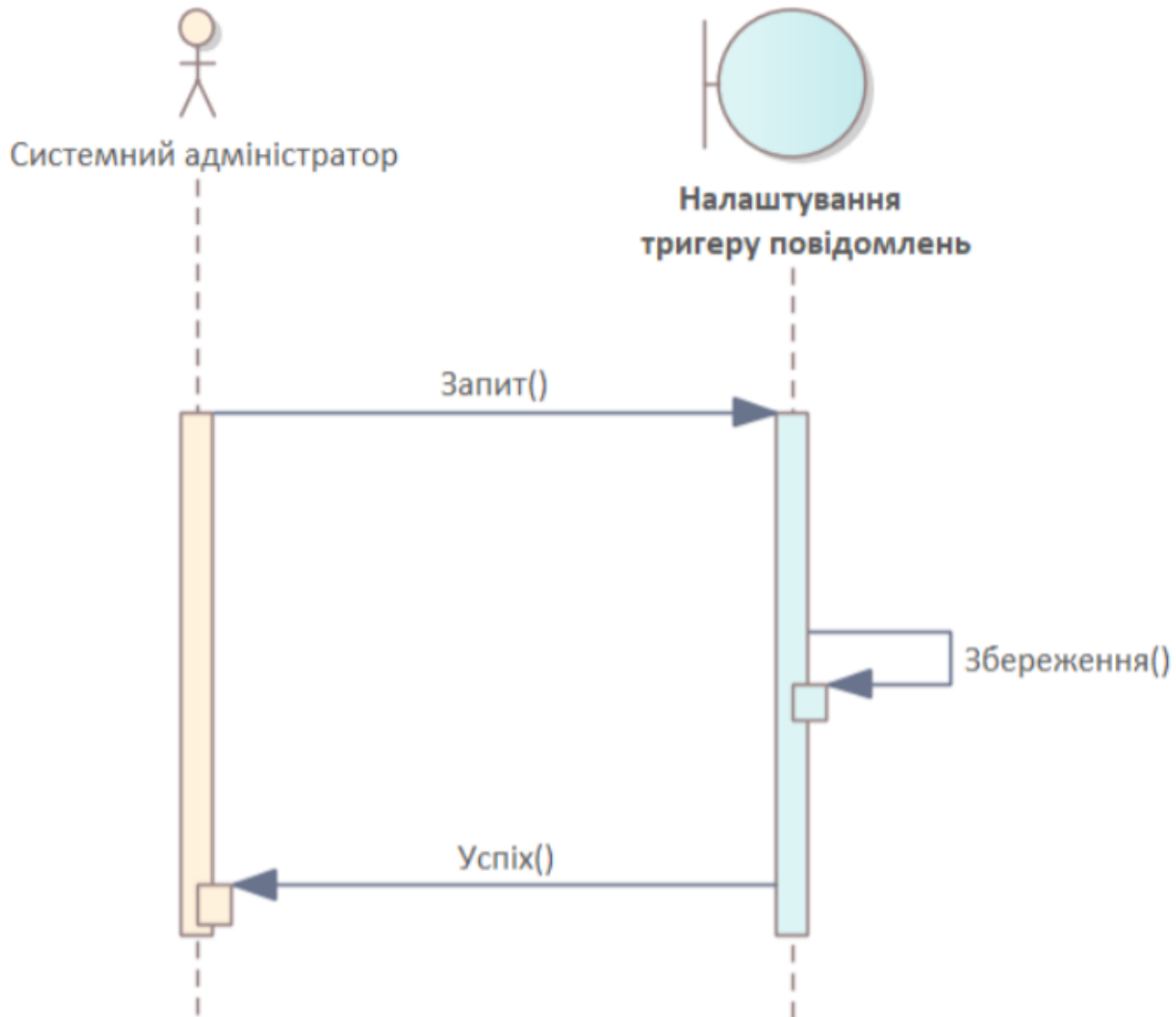


Рисунок 2.13 – Діаграма послідовності налаштування триггеру повідомлень

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Діаграма відображає процес налаштування системи моніторингу логів, де адміністратор встановлює тригери (правила) для виявлення певних подій у лог-файлах. Ця послідовність є частиною більшої системи моніторингу та аналізу логів у реальному часі, де після налаштування тригерів система зможе автоматично відстежувати події у логах та надсилати повідомлення при виявленні заданих умов.

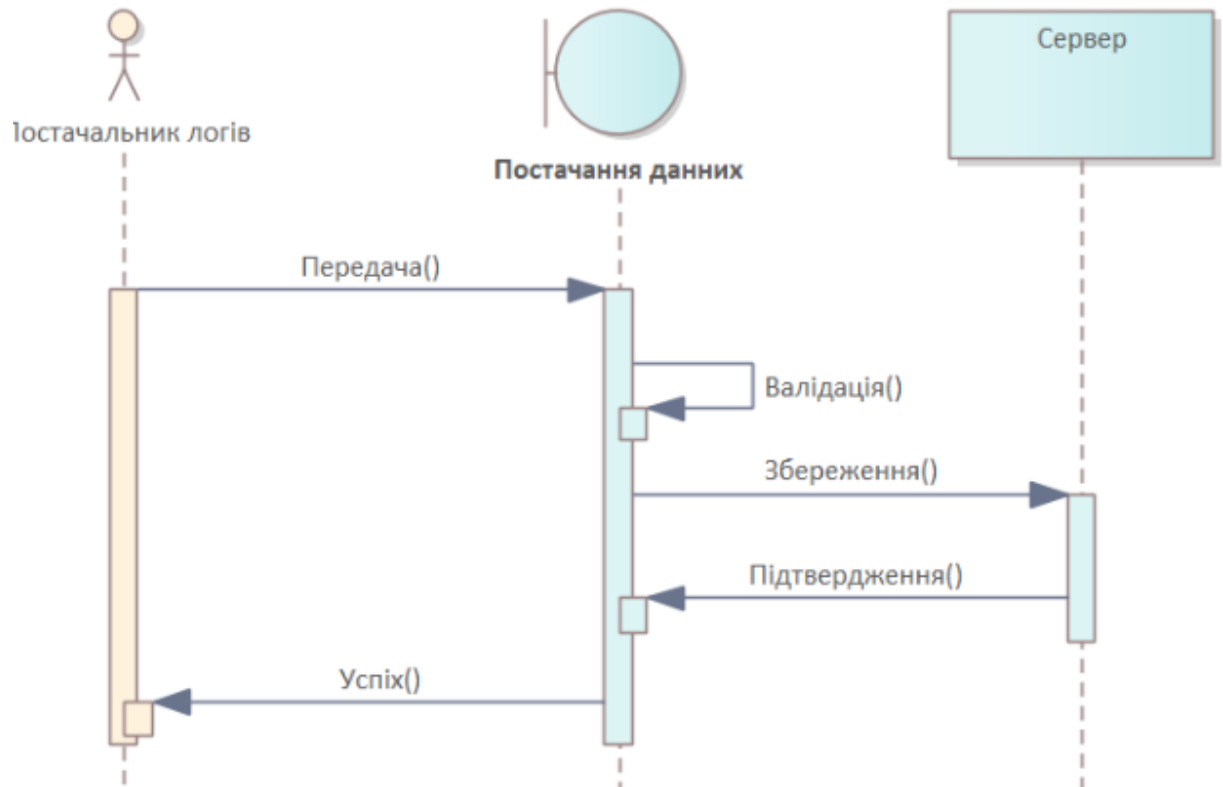


Рисунок 2.14 – Діаграма послідовності Постачання даних

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

У цьому діаграмі постачальник логів виступає джерелом лог-файлів, компонент постачання даних відповідає за їх обробку та аналіз, а сервер за зберігання. Усе починається з того, що постачальник логів ініціює процес. Отримані логи проходять перевірку компонентом постачання даних. Якщо валідація успішна, дані передаються на сервер, де зберігаються. Сервер, у свою чергу, підтверджує успішне збереження. Після цього компонент постачання даних надсилає постачальнику логів повідомлення про успішне завершення процесу.

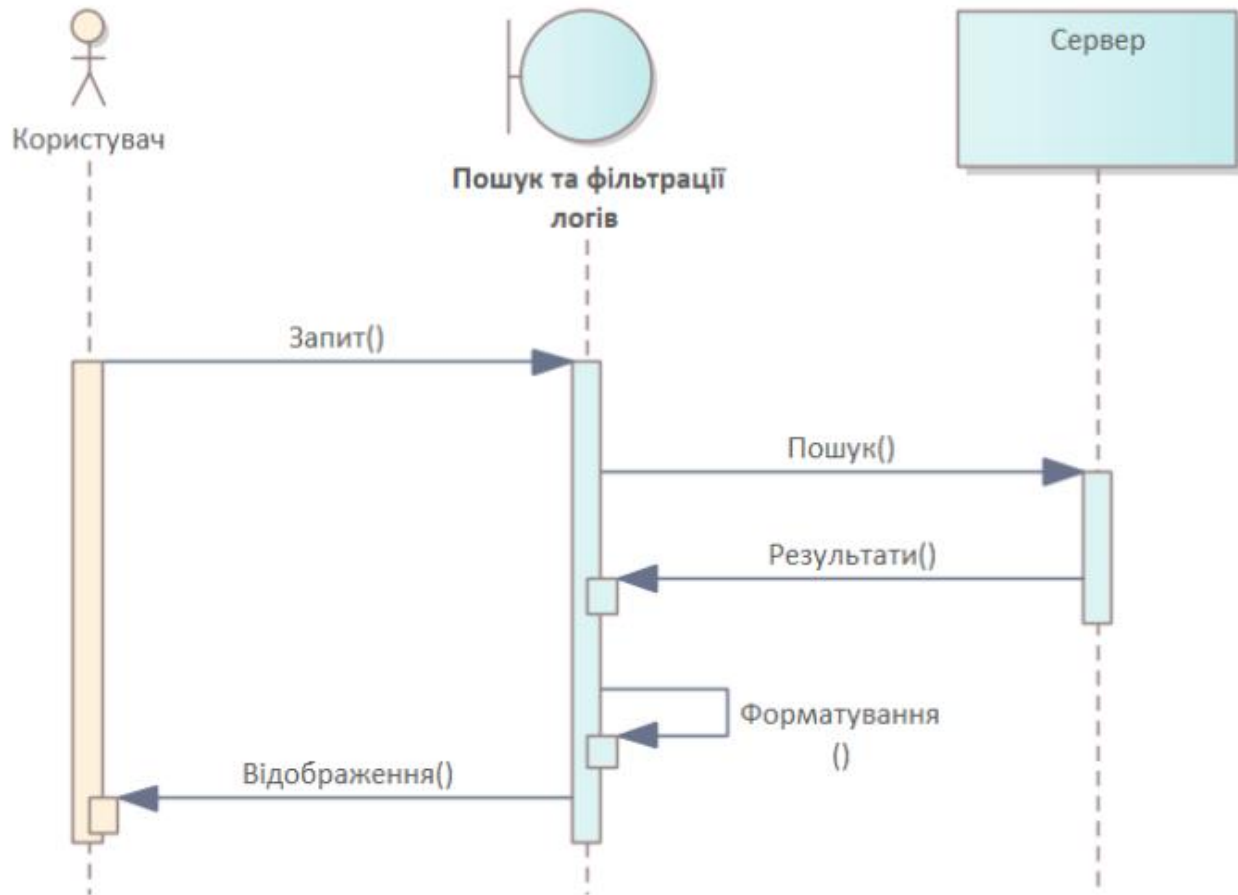


Рисунок 2.15 – Діаграма послідовності Пошук та фільтрація логів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Ця діаграма послідовності ілюструє процес пошуку та фільтрації логів у системі моніторингу та аналізу лог-файлів у реальному часі, відображаючи взаємодію між користувачем, компонентом пошуку та фільтрації логів і сервером. У цьому процесі користувач ініціює запит на пошук логів, який обробляється компонентом пошуку та фільтрації. Цей компонент, у свою чергу, взаємодіє із сервером, формує результати та надає їх користувачу у зручному форматі. Сервер відповідає за виконання пошуку логів і передачу відповідних даних назад до компонента пошуку.

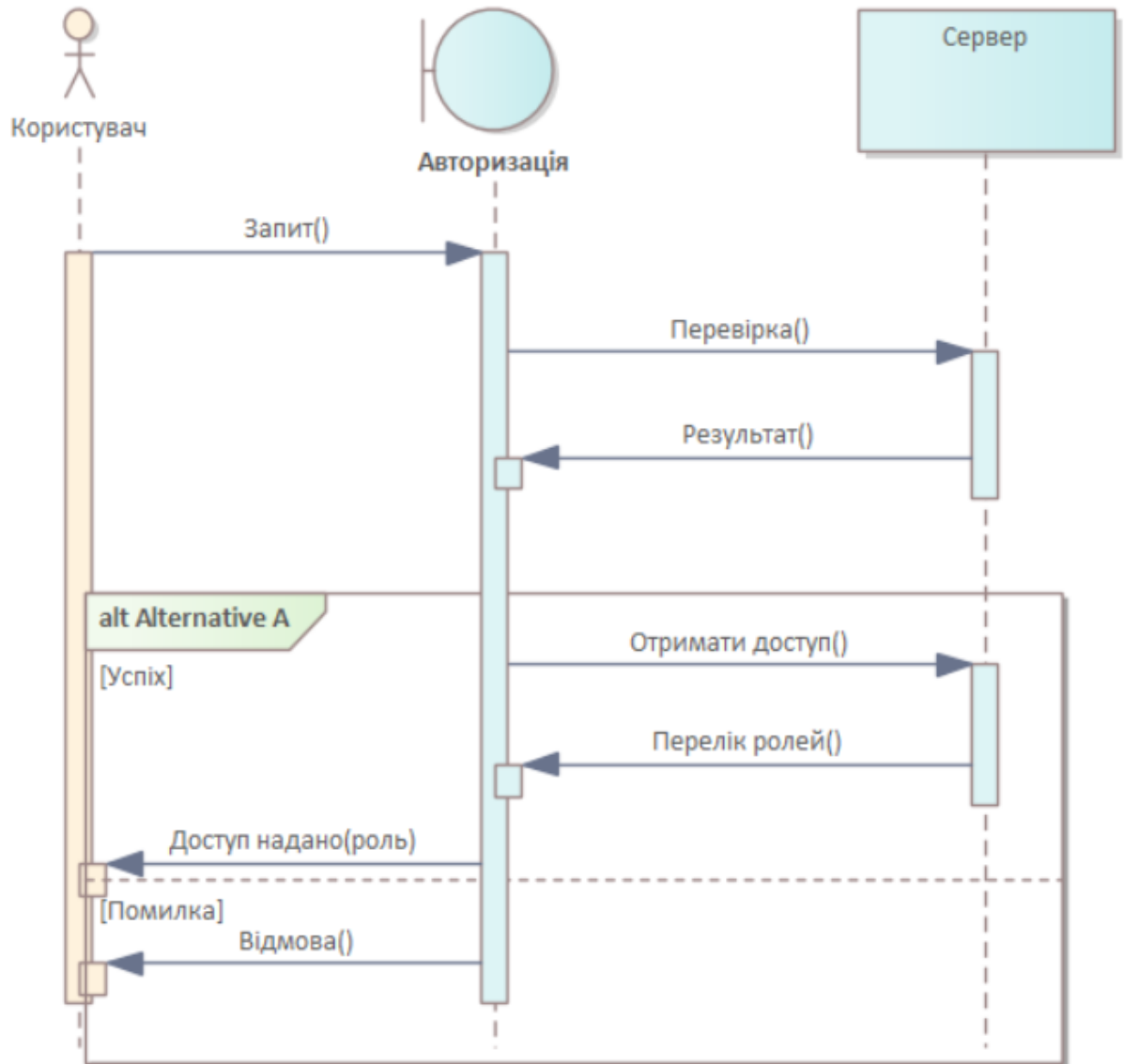


Рисунок 2.16 – Діаграма послідовності авторизація

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.3.2. Моделювання структури системи

Прийнято рішення використовувати архітурний патерн DDD (Domain Driven Design), оскільки він надає ефективний підхід до розробки програмного забезпечення для складних доменів із великою кількістю бізнес-правил. У контексті системи моніторингу та аналізу лог-файлів у реальному часі, DDD

дозволяє зосередитись на основних аспектах домену: обробці подій, агрегації логів, виявленні аномалій, генерації сповіщень та інтеграції з іншими сервісами.

DDD сприяє структуризації системи через доменні сутності (наприклад, LogEntry, AlertRule, Incident), сервіси та value-об'єкти, що забезпечує прозорість, контроль над логікою та гнучкість у розширенні функціональності. Такий підхід є особливо важливим у системах, де обробка подій відбувається в реальному часі, і необхідна висока надійність та масштабованість.

Однією з ключових переваг DDD є тісна співпраця з експертами домену. У сфері обробки логів це дозволяє узгодити розуміння таких понять, як «аномалія», «інцидент», «правило тригера» тощо, між технічними і бізнес-учасниками проекту. Створення спільної мови мінімізує непорозуміння, покращує якість системи і пришвидшує адаптацію до нових вимог (наприклад, зміна форматів логів чи політик безпеки).

Система моніторингу часто інтегрується з багатьма зовнішніми джерелами: веб-серверами, базами даних, хмарними платформами. DDD дозволяє виокремити такі інтеграції в bounded context, зберігаючи чистоту ядра бізнес-домену. Це дозволяє ізолювати специфіку окремих джерел логів і забезпечити адаптивність системи при масштабуванні або заміні зовнішніх сервісів.

Ще одна сильна сторона DDD - можливість чітко моделювати доменні правила, наприклад: коли слід створити інцидент, за яких умов тригер активується, як обробляти дублікати повідомлень. Це критично для системи, яка автоматично реагує на події в реальному часі.

DDD також добре поєднується з підходами на кшталт Event Sourcing або CQRS, що є особливо корисними для систем моніторингу. Вони дозволяють фіксувати всі зміни у вигляді подій, забезпечуючи трасування, аудит та аналітику. Це значно спрощує виявлення проблем, відстеження історії подій і створення систем звітності.

- **LogItem (Запис логу)** — це окрема подія або повідомлення, яке фіксується системою. Наприклад: помилка на сервері, повідомлення про підключення користувача, попередження про нестачу пам'яті тощо. Лог містить рівень важливості (level), текст повідомлення, джерело та час створення.

- **LogSource (Джерело логу)** — описує, звідки саме надійшов лог. Це може бути певний сервер, програма чи мікросервіс. Завдяки цьому можна відслідковувати, які системи генерують найбільше повідомлень.

Дашборди та фільтри

- **Dashboard (Дашборд)** — це візуальна панель користувача, яка показує важливу інформацію з логів у зручному вигляді: графіки, таблиці, віджети.

- **Filter (Фільтр)** — набір критеріїв, за якими можна відібрати лише потрібні записи логів. Наприклад, всі помилки за останню годину з певного сервера.

Тригери та сповіщення

- **Trigger (Тригер)** — умова, за якої система автоматично виконує певну дію. Наприклад, якщо з'явилося повідомлення з рівнем "error" у логах, система може спрацювати тригер.

- **Notification (Сповіщення)** — повідомлення, яке надсилається користувачу або системі, коли спрацював тригер. Містить текст, отримувача та час надсилання. Це дозволяє швидко реагувати на проблеми без необхідності постійного перегляду логів вручну.

Звіти

- **Report (Звіт)** — підсумковий документ, який генерується на основі логів, фільтрів або запитів. Наприклад, звіт про всі помилки за

тиждень. Звіти можна експортувати у різних форматах — PDF, CSV, JSON тощо.

Користувач заходить у систему, проходить авторизацію. Система перевіряє його роль та дозволи. Через інтерфейс користувач може:

- Переглядати лог-файли за допомогою фільтрів.
- Налаштовувати дашборд, який виводить потрібні йому дані в реальному часі.
- Створювати тригери, які автоматично реагують на появу певних логів.
- Отримувати сповіщення, коли щось пішло не так.
- Формувати звіти, які зручно зберігати або передавати колегам.

Діаграми визначення блоків забезпечують структурований підхід до аналізу та проектування системи, сприяючи виявленню ключових компонентів, їх функціональності та взаємодії. Такий підхід дозволяє візуалізувати функціональні блоки системи моніторингу логів, включаючи модулі збору, обробки та аналізу лог-файлів у реальному часі. Наприклад, це можуть бути окремі блоки для агрегації логів з різних джерел, фільтрації даних, виявлення аномалій, створення сповіщень, а також інтерфейси для візуалізації результатів аналізу та керування системою.

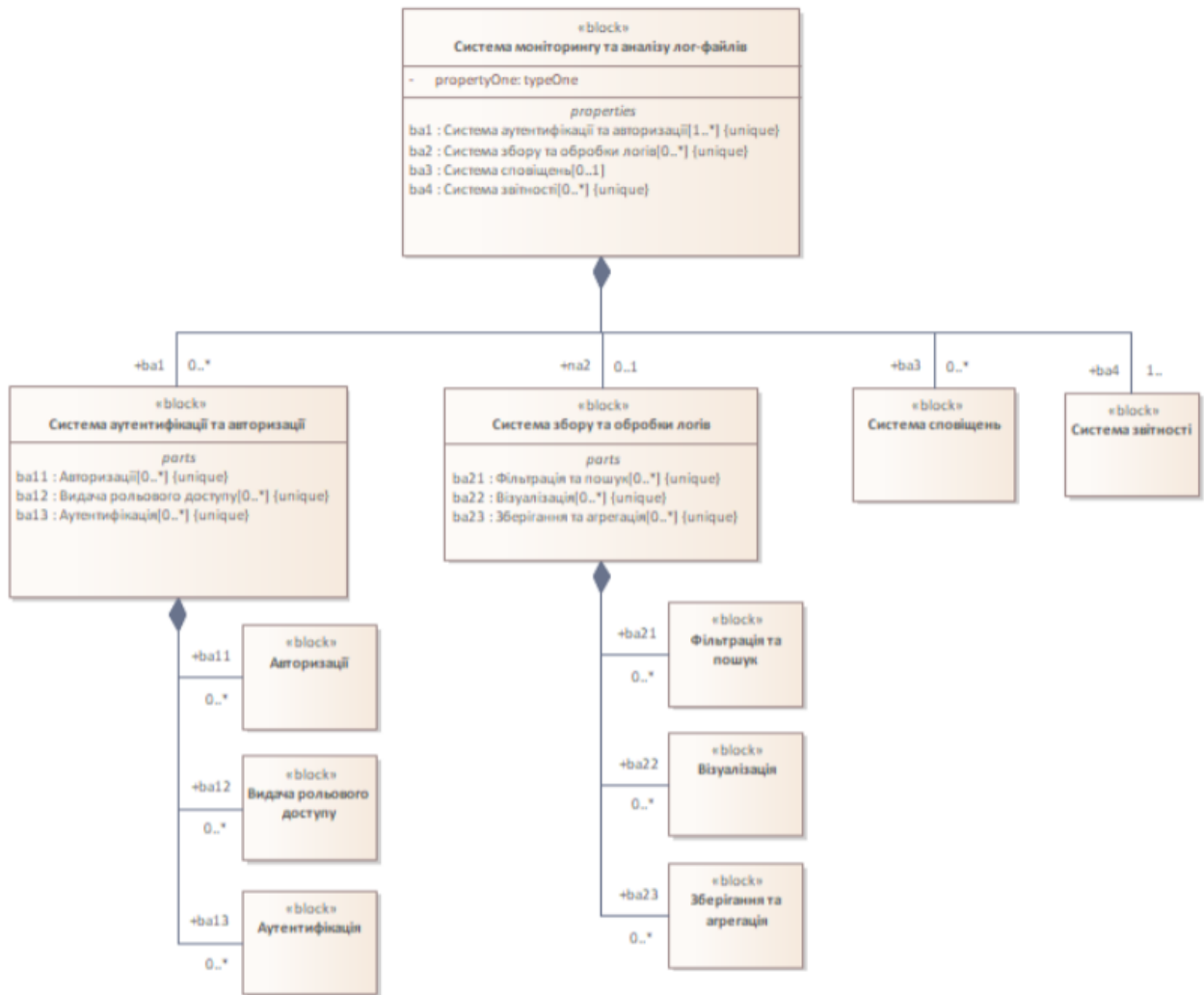


Рисунок 2.18 – Діаграма визначення блоків для подання внутрішньої структури системи (BDD)

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

На рисунку 2.19 представлено діаграму процесу обробки лог-файлів. Процес починається з блоку LogInput, який відповідає за надходження не оброблених логів у систему. Ці логи передаються далі до модуля збору даних, що агрегує інформацію з різних джерел. Після агрегації дані спрямовуються до блоку первинної обробки, де вони фільтруються, нормалізуються та очищуються від зайвої інформації. Далі оброблені логи надходять до аналітичного модуля, який застосовує до них відповідні алгоритми. Це може включати виявлення аномалій, виведення кореляцій або генерацію алертів. Результати аналізу зберігаються і передаються до фінального блоку AlertOutput, який забезпечує

вивід алертів або аналітичних даних для подальшого реагування, наприклад, надсилання повідомлень адміністраторам.

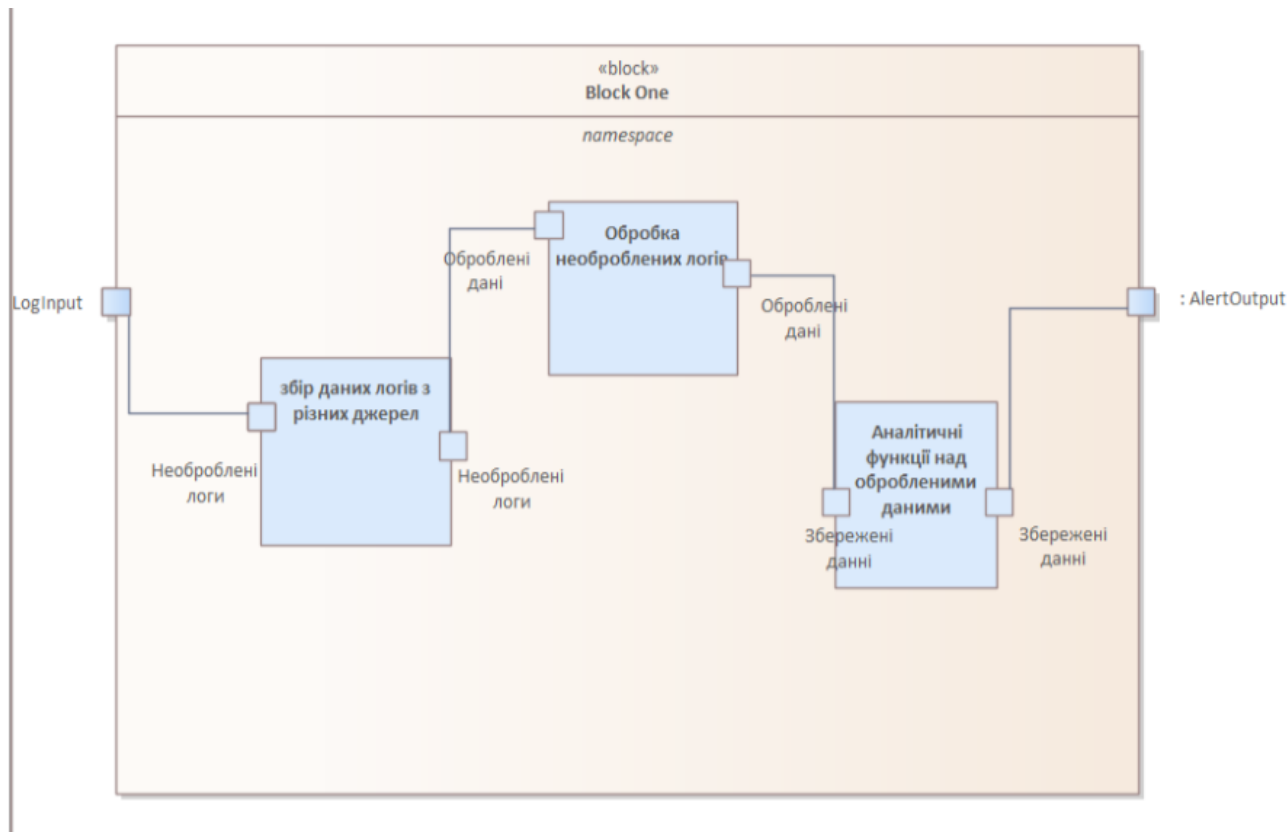


Рисунок 2.19 – Діаграма процесу обробки логів (IBD)

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

2.3.3. Моделювання користувацького інтерфейсу

На рисунку 2.20 зображено інтерфейс сторінки дашборду, яка призначена для перегляду логів.

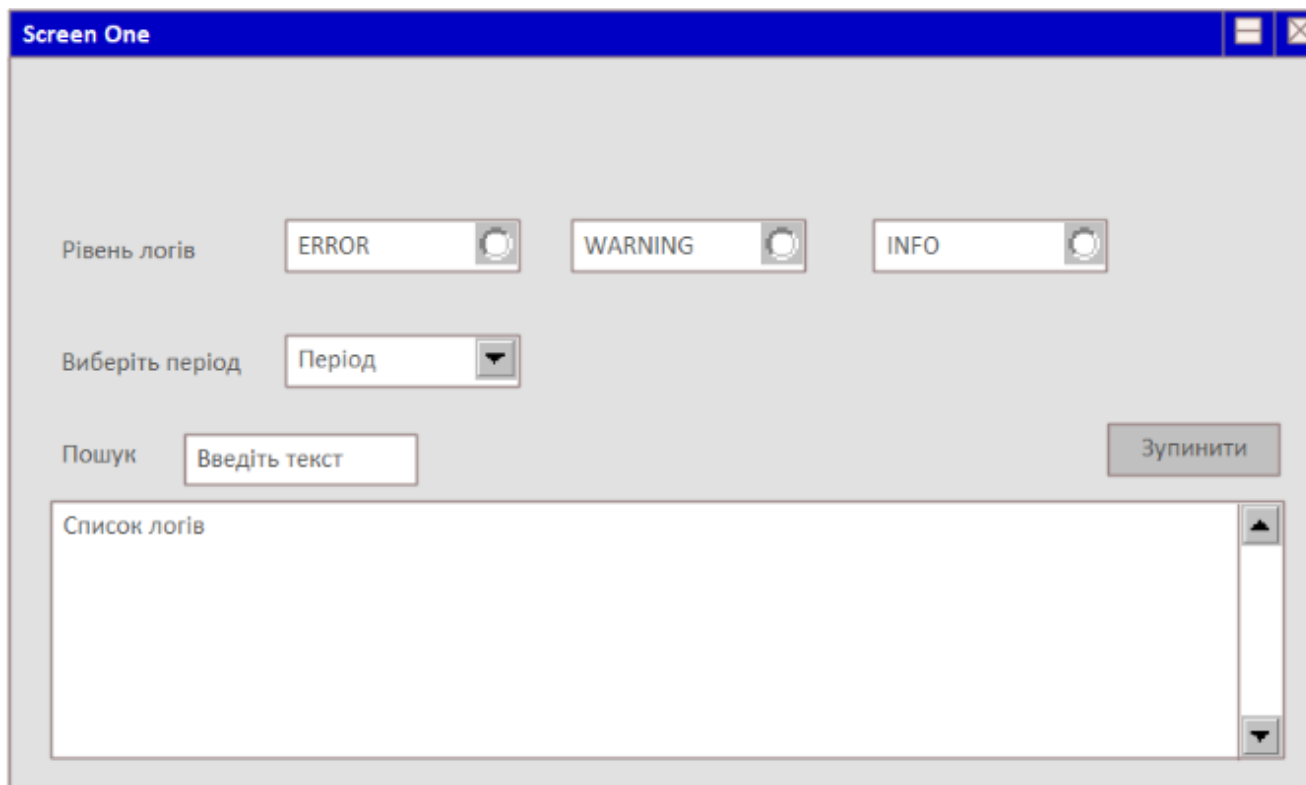


Рисунок 2.20 – Інтерфейс сторінки дашборду

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

На рисунку 2.21 зображено інтерфейс сторінки генерації звіту, яка призначена для налаштування параметрів і генерації звіту.

The screenshot shows a web application window titled "Screen One". Inside, there is a form titled "Параметри звіту" (Report Parameters). The form contains several input fields and checkboxes. On the left, under "Період" (Period), there is a dropdown menu with "Дата" (Date) selected. Below it, under "Рівні логів" (Log Levels), there are three checkboxes: "INFO", "WARNING", and "ERROR", all of which are currently unchecked. On the right, under "Формат звіту" (Report Format), there are two radio buttons: "CSV" and "PDF", both of which are currently unchecked. A "Згенерувати" (Generate) button is located at the bottom right of the "Параметри звіту" section. Below this section, there is a box labeled "Останній звіт" (Last Report) which is currently empty. A "Завантажити" (Download) button is located at the bottom right of the "Останній звіт" box.

Рисунок 2.21 – Інтерфейс сторінки генерації звіту

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

На рисунку 2.22 зображено інтерфейс сторінки створення тригера для повідомлень, вона призначена для налаштування тригерів сповіщень, які реагують на події в системі.

Screen One

Створення тригера

Назва

Введіть назву тригера

Умови

Рівень ERROR

Канали сповіщень

Email admin@example.com

Telegram @ihorbourak

Зберегти Скасувати

Рисунок 2.22 – Інтерфейс сторінки створення тригера

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

РОЗДІЛ 3

ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

3.1 Інформаційне забезпечення

Структура інформаційного забезпечення включає декілька ключових компонентів. Центральне сховище даних містить первинні лог-записи з різних джерел, включаючи веб-сервери, системи безпеки, мережеве обладнання та додатки. Метадані описують структуру та походження кожного лог-запису, включаючи часові мітки, рівні важливості, категорії подій та джерела інформації. Індeksi та агрегації забезпечують швидкий доступ до даних для аналітичних запитів і формування звітів.

Додатково система містить конфігураційні дані, які визначають правила парсингу різних форматів логів, налаштування алертів та моніторингу, а також параметри збереження та архівації даних. Історичні дані зберігаються для трендового аналізу та порівняння з поточними показниками системи.

Інформаційне забезпечення охоплює:

- Джерела даних: лог-файли з серверів, мережевого обладнання, додатків (web, backend), баз даних.
- Формати даних: JSON, plaintext, syslog-формат.
- Обробка в реальному часі: дані агрегуються і фільтруються за допомогою потокових сервісів.
- Зберігання: історичні та поточні логи зберігаються в БД.
- Інтерфейс доступу: панель керування (dashboard) для візуалізації метрик, запитів, фільтрації логів.
- Аналітика: застосування правил для виявлення підозрілої активності, алертинг.

На основі цього PostgreSQL виглядає найкращим варіантом для створення надійної та масштабованої системи. PostgreSQL представляє собою потужну об'єктно-реляційну систему управління базами даних з відкритим вихідним кодом, яка ідеально підходить для системи моніторингу лог-файлів. Це рішення забезпечує високу продуктивність обробки транзакцій, підтримує складні запити та аналітичні операції, а також надає широкі можливості для роботи з неструктурованими даними.

Архітектура PostgreSQL базується на процесній моделі, де кожне з'єднання обслуговується окремим процесом, що забезпечує стабільність та ізоляцію. Система підтримує ACID-властивості транзакцій та багаторівневе блокування.

Для лог-файлів критично важливими є GIN-індекси для повнотекстового пошуку та BRIN-індекси для великих таблиць з часовими даними, які забезпечують швидкий доступ до інформації при мінімальних витратах дискового простору.

PostgreSQL надає потужні можливості для роботи з JSON-документами через нативну підтримку типів JSON та JSONB, що особливо актуально для сучасних структурованих логів. Оператори та функції для роботи з JSON дозволяють ефективно індексувати та запитувати вкладені структури даних без необхідності попередньої нормалізації.

Продуктивність при роботі з великими обсягами даних є критичним фактором. PostgreSQL демонструє відмінні показники при обробці мільйонів записів завдяки ефективним алгоритмам планування запитів, підтримці паралельного виконання операцій та оптимізованим структурам зберігання.

Таблиця 3.1 – Опис таблиці log_sources

Назва атрибута	Формат	Обмеження на право звертання	Первинний (вторинний) ключ	Вивідимість значень	Індексне поле
----------------	--------	------------------------------	----------------------------	---------------------	---------------

id	Ціле	Тільки для читання / адміністратор	Первинний	Так	Так (PK)
name	Текст	Читання/запис — адміністратор	Унікальне	Так	Так
description	Текст	Читання/запис — адміністратор	—	Так	Ні

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Таблиця 3.2 – Опис таблиці log_levels

Назва атрибута	Формат	Обмеження на право звертання	Первинний (вторинний) ключ	Виводимість значень	Індиксне поле
id	Ціле	Тільки для читання / адміністратор	Первинний	Так	Так (PK)
level	Текст	Читання/запис — адміністратор	Унікальне	Так	Так

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Таблиця 3.3 – Опис таблиці log_entries

Назва атрибута	Формат	Обмеження на право звертання	Первинний (вторинний) ключ	Виводимість значень	Індиксне поле
id	Ціле (BIG)	Тільки для читання	Первинний	Так	Так (PK)
source_id	Ціле	Читання/запис	Вторинний (FK)	Так	Так
level_id	Ціле	Читання/запис	Вторинний (FK)	Так	Так
message	Текст	Читання/запис	—	Так	Ні
timestamp	Дата/час	Читання/запис	—	Так	Так
content	JSON В	Читання/запис	—	Так	Так (GIN)
created_at	Дата/час	Тільки для читання	—	Так	Ні

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Таблиця 3.4 – Опис таблиці alerts

Назва атрибута	Формат	Обмеження на право звертання	Первинний (вторинний) ключ	Виводимість значень	Індиксне поле
----------------	--------	------------------------------	----------------------------	---------------------	---------------

id	Ціле	Тільки и для читання	Перв инний	Так	Так (PK)
name	Текст	Читан ня/запис	—	Так	Так
condition	JSON В	Читан ня/запис адміністрат ором	—	Так	Так
notification	Текст	Читан ня/запис	—	Так	Ні
created_by	Текст	Тільки и для читання	—	Так	Так
is_active	Булев е	Читан ня/запис	—	Так	Так
created_at	Дата/ час	Тільки и для читання	—	Так	Ні

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Таблиця 3.5 – Опис таблиці alert_events

Назва атрибута	Формат	Обмеження на право звертання	Первинний (вторинний) ключ	Виводимість значень	Індексне поле
id	Ціле	Тільки и для читання	Перв инний	Так	Так (PK)

alert_id	Ціле	Читання	Вторинний (FK)	Так	Так
triggered_at	Дата/час	Тільки для читання	—	Так	Так
log_entry_id	Ціле (BIG)	Читання	Вторинний (FK)	Так	Так
message	Текст	Тільки для читання	—	Так	Ні

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Таблиця 3.6 – Опис таблиці reports

Назва атрибута	Формат	Обмеження на право звертання	Первинний (вторинний) ключ	Виводимість значень	Індикаторне поле
id	Ціле	Тільки для читання	Первинний	Так	Так (PK)
name	Рядок (255)	Читання/Запис (admin/user)	—	Так	Ні
description	Текст	Читання/Запис (admin)	—	Ні	Ні
report_type	Рядок (50)	Тільки для читання	Вторинний	Так	Так

config	JSON	Запис (admin)	—	Так	Ні
sched ule	JSON	Запис (admin)	—	Ні	Так (GIN)
forma t	Рядок (20)	Тільки и для читання	—	Так (за замовчуван ням: pdf)	Ні
recipi ents	Маси в тексті	Запис (admin)	—	Ні	Ні
create d_by	Рядок (255)	Тільки и для читання	Втор инний	Так	Так
is_acti ve	Логіч не	Запис (admin)	—	Так (default: true)	Так
create d_at	Дата+ Час	Тільки и для читання	—	Так	Ні
updat ed_at	Дата+ Час	Тільки и для читання	—	Так	Ні

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Таблиця 3.7 – Опис таблиці reports_executions

Назв а атрибута	Форм а т	Обме ження на	Перв инний	Виво димість значень	Індек сне поле
--------------------	-------------	------------------	---------------	----------------------------	-------------------

		право звертання	(вторинни й) ключ		
id	Ціле	Тільки и для читання	Перв инний	Так	Так (PK)
report _id	Ціле	Тільки и для читання	Втор инний (FK → reports.id)	Так	Так
status	Рядок (50)	Тільки и для читання	Втор инний	Так (default: pending)	Так
file_p ath	Текст	Тільки и для читання	—	Ні	Ні
file_si ze	Ціле	Тільки и для читання	—	Ні	Ні
execut ion_time	Ціле	Тільки и для читання	—	Ні	Ні
error_ message	Текст	Тільки и для читання	—	Ні	Ні
param eters	JSON	Тільки и для читання	—	Ні	Ні

executed_at	Дата+ Час	Тільки для читання	Вторинний	Так	Так
completed_at	Дата+ Час	Тільки для читання	—	Ні	Ні

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

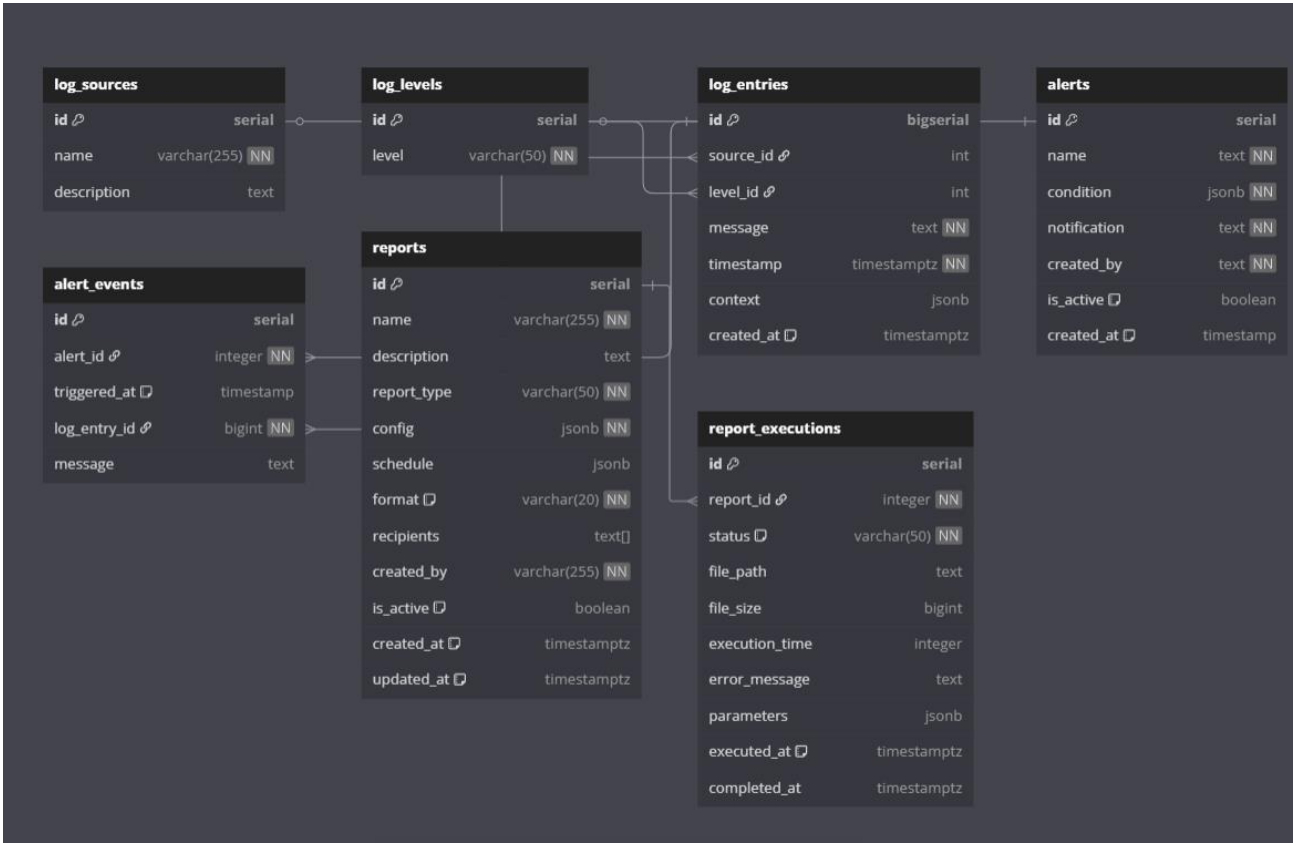


Рис.3.1. – Інфологічна модель ІС

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

3.2 Технічне забезпечення

Основою системи є серверна частина, яка виконує функції збору, обробки, зберігання логів та взаємодії з користувачем. Веб-сервер приймає запити від клієнтів через браузер, обробляє їх та передає на сервер додатків.

Сервер додатків реалізує бізнес-логіку системи: обробляє запити користувачів, виконує авторизацію, формує відповіді на основі даних з бази або системи логів.

Окремий компонент системи — це платформа збору логів. Логи, що генеруються на різних серверах (наприклад, веб-сервери, бази даних, додатки), передаються за допомогою агентів збору на центральний сервер, який агрегує, обробляє та передає їх у сховище логів. Обробка логів передбачає їх нормалізацію, фільтрацію та перетворення у формат, зручний для пошуку та аналізу.

Для ефективного зберігання та пошуку логів використовується реляційна база даних. Вона дозволяє індексувати логи в реальному часі та швидко здійснювати пошук навіть за великими обсягами даних

Для обробки даних у режимі реального часу між модулями використовується брокер повідомлень. Він дозволяє передавати повідомлення між компонентами з мінімальною затримкою, підтримуючи реакцію на події практично миттєво.

Користувачі взаємодіють із системою через веб-інтерфейс у браузері. Клієнтська частина реалізована на сучасному JavaScript-фреймворку і забезпечує зручний інтерфейс для пошуку, фільтрації та візуалізації логів. Оновлення даних здійснюється через WebSocket-з'єднання або регулярні запити до API, що дозволяє отримувати інформацію в реальному часі.

Схема взаємодії технічних засобів:

1. **Користувач** через браузер надсилає запит до **веб-сервера**. Запит може містити, наприклад, фільтр для пошуку логів, час, ключові слова або ідентифікатор користувача.

2. **Веб-сервер** приймає запит і передає його на **сервер додатків**, який відповідає за обробку логіки. Він перевіряє права доступу, обробляє запит і формує звернення до відповідного джерела даних.

3. Якщо потрібна інформація про логи сервер додатків звертається до **системи зберігання логів**, наприклад до **Elasticsearch**, де зберігаються та індексуються всі події. Якщо запит стосується налаштувань або облікових даних виконується звернення до **реляційної бази даних**.

4. Паралельно агенти збору логів, встановлені на серверах інфраструктури, постійно передають нові записи логів у систему збору звідки вони потрапляють у Elasticsearch або іншу систему зберігання.

5. Отримані дані повертаються сервером додатків на веб-сервер, а той клієнту. У випадку використання WebSocket-з'єднання, користувач отримує оновлення логів у реальному часі без потреби оновлювати сторінку.

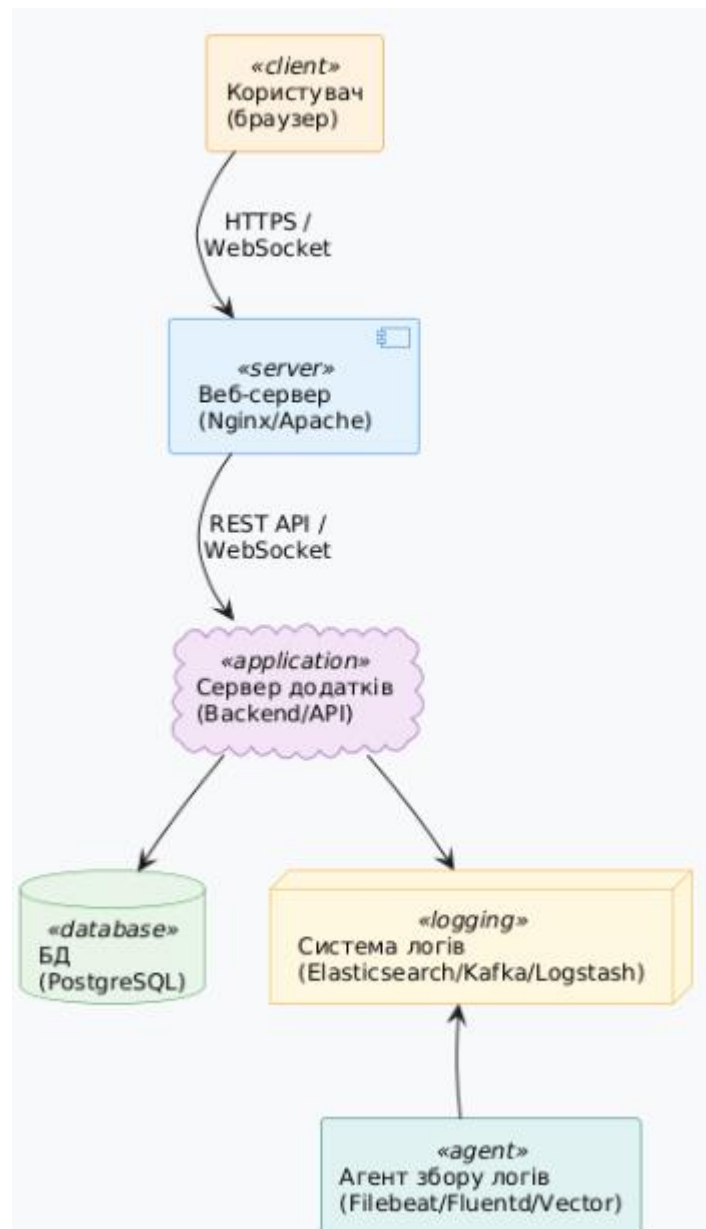


Рис.3.2 – Загальна схема автоматизації

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

3.3 Програмне забезпечення

Для розробки системи було обрано середовище розробки WebStorm. Це рішення зумовлене низкою важливих переваг, які забезпечують ефективну та якісну роботу над проєктом.

WebStorm є потужним середовищем, яке повністю підтримує JavaScript, TypeScript та React, що є основними технологіями розробки у цьому проєкті.

Крім того, воно дозволяє зручно працювати з HTML, CSS, що є необхідним для створення інтерфейсу користувача.

Особливо важливим аспектом є можливість інтеграції з базами даних. У контексті даного проєкту лог-файли зберігаються у структурованому вигляді в базі даних, тому можливість швидкого підключення до MySQL або PostgreSQL через ORM-рішення та виконання запитів є суттєвою перевагою.

Ще однією важливою причиною вибору сучасного IDE є підтримка локального запуску Next.js серверів і можливість налагодження. Це дає змогу покроково відстежувати виконання коду, перевіряти значення змінних і оперативно виявляти логіку помилок, що особливо важливо у системі, яка працює в режимі реального часу.

IDE також дозволяє зручно працювати з системою контролю версій Git, що дає змогу зберігати історію змін, створювати гілки для нових функціональностей і швидко повертатися до попередніх версій проєкту.

Технологією розробки на якій буде будуватись уся екосистема є Next.js. Next.js це сучасний React-фреймворк, який поєднує в собі переваги серверного та клієнтського рендерингу, забезпечуючи високу продуктивність та SEO-оптимізацію. Основними перевагами є широка підтримка спільноти та наявність великої кількості готових бібліотек у npm-екосистемі.

Next.js характеризується простотою інтеграції з різними базами даних через ORM-рішення як Prisma або TypeORM, що важливо для зберігання інформації про транзакції. Фреймворк також надає вбудовані API Routes, що дозволяє створювати backend-функціональність безпосередньо в рамках проєкту.

Сучасна архітектура Next.js значно покращує продуктивність завдяки автоматичному розділенню коду (code splitting), статичній генерації сторінок (SSG) та серверному рендерингу (SSR), що дозволяє обробляти великі обсяги запитів. Вбудована підтримка HTTPS та сучасних веб-стандартів забезпечує захищену передачу даних.

Підтримка серверних компонентів (Server Components) та потокового рендерингу дозволяє ефективно обробляти множинні лог-файли одночасно, що підвищує пропускну здатність системи.

Після вибору основної технології розробки, слід визначитись з не менш важливою частиною нашої ІС додатковими бібліотеками та інструментами. Next.js надає повноцінну екосистему для побудови сучасних веб-додатків корпоративного рівня. Його модульна архітектура дозволяє використовувати лише ті бібліотеки, які необхідні для конкретного проекту, оптимізуючи продуктивність системи.

Для системи особливо цінними є такі можливості Next.js як вбудована система безпеки, включаючи захист від CSRF-атак, XSS-вразливостей. Middleware дозволяє реалізувати додаткові рівні захисту та валідації запитів. Фреймворк також забезпечує відмінний рівень роботи з базами даних через Prisma ORM, що спрощує роботу з різними СУБД та полегшує міграцію даних.

Next.js надає розвинуту систему автентифікації через NextAuth.js та авторизації. Вбудована підтримка форм з валідацією даних через бібліотеки як React Hook Form дозволяє забезпечити коректність введення інформації користувачами. Компонентна структура React сприяє розробці за принципами SOLID, що підвищує якість коду та його підтримуваність у довгостроковій перспективі.

Next.js має добре документовані API та чіткі конвенції розробки, що спрощує інтеграцію з зовнішніми сервісами. Фреймворк також надає потужні засоби профілювання та відлагодження через React DevTools та Next.js аналітику, які допомагають виявляти та усувати проблеми продуктивності.

Архітектура системи на базі Next.js буде побудована за принципами повнофункціонального фреймворку (Full-Stack). Next.js має вбудовані механізми для реалізації API через API Routes, що дозволяє створювати RESTful та GraphQL сервіси для обробки платіжних запитів.

Для оптимізації продуктивності варто використовувати кешування (Next.js має вбудоване кешування), а для горизонтального масштабування використовувати Redis для сесій та кешування.

Для зберігання інформації про транзакції, користувачів, платіжні методи та інші важливі дані обрана реляційна база даних PostgreSQL. Вона відзначається високою стабільністю, надійністю та підтримкою складних транзакцій, що критично важливо для фінансових систем. PostgreSQL має потужні механізми забезпечення цілісності даних (через зовнішні ключі, тригери, обмеження) та підтримує ACID-властивості, що дозволяє забезпечити правильність обробки платіжних операцій навіть у випадках збоїв системи.

Серед переваг PostgreSQL також варто виділити підтримку розширень (наприклад, pgcrypto для шифрування), широкі можливості оптимізації запитів, індексування, а також зручний механізм реплікації, що дозволяє масштабувати систему в разі зростання навантаження.

Для взаємодії Next.js з базою даних обрана ORM-бібліотека Prisma, яка є сучасним стандартом для TypeScript/JavaScript проєктів. Prisma забезпечує високий рівень type-safety та дозволяє працювати з типізованими об'єктами замість ручного створення запитів. Це значно спрощує розробку, покращує читабельність та підтримуваність коду, а також дозволяє ефективно реалізувати шаблони проектування, такі як Repository.

Prisma підтримує міграції бази даних, які є важливим інструментом для керування схемою бази даних у великих проєктах, де змінюється структура таблиць з часом. Завдяки можливості кешування результатів запитів (наприклад, через Redis), Prisma також дозволяє зменшити навантаження на базу даних при повторному доступі до однакових даних.

Для управління асинхронним обміном інформацією між системою та зовнішніми API буде використано систему черг на базі BullMQ з RabbitMQ як транспортом. Це дозволить обробляти велику кількість запитів без блокування

основного потоку обробки запитів та забезпечить надійність обробки колбеків від платіжних систем.

Клієнтська частина системи моніторингу та аналізу лог-файлів в режимі реального часу буде розроблена як інтегральна частина Next.js додатку з використанням React компонентів. Основним завданням інтерфейсу є забезпечення зручного, швидкого та динамічного відображення великого обсягу даних, які постійно оновлюються. У цьому контексті React з Next.js демонструє свою ефективність, оскільки створений для побудови реактивних інтерфейсів з високим рівнем продуктивності.

React надає потужну систему реактивного зв'язування даних через хуки (`useState`, `useEffect`), що дозволяє автоматично оновлювати елементи інтерфейсу при зміні стану даних без необхідності ручного втручання. У системах, де дані надходять у режимі реального часу наприклад, лог-файли з серверів, які постійно доповнюються новими записами, це особливо важливо. Замість перезавантаження сторінки або ручного оновлення, інтерфейс автоматично відображає нові повідомлення, забезпечуючи користувача максимально актуальною інформацією.

Крім того, React базується на компонентній архітектурі, що дає змогу розбити інтерфейс на логічно ізольовані та багаторазово використовувані блоки. Такий підхід значно полегшує розробку та супровід системи, дозволяє легко реалізовувати нову функціональність або змінювати існуючу без ризику порушити роботу інших частин програми. Наприклад, таблиця логів, панель фільтрів, індикатор підключення чи графік активності можуть бути окремими компонентами з власною логікою та стилями, які взаємодіють між собою через чітко визначені інтерфейси.

Ще одним важливим аспектом є легка інтеграція React з WebSocket-протоколом, який використовується для організації обміну даними в режимі реального часу. Завдяки реактивності React, дані, що надходять через WebSocket,

можуть миттєво відображатися в DOM-дереві браузера без додаткових оновлень, що забезпечує плавність і безперервність роботи системи.

Next.js з React також має низький поріг входу і простий у засвоєнні синтаксис, що особливо важливо в контексті навчального проєкту. Використання JSX, зрозумілих хуків і гнучкої системи прив'язок робить процес розробки логічно послідовним і менш обтяжливим. Це дозволяє сконцентруватися на побудові логіки взаємодії з логами, системою фільтрації, візуалізацією та повідомленнями, не витрачаючи надмірно багато часу на вивчення інструменту.

Фреймворк має добре структуровану офіційну документацію, велику спільноту, а також підтримує інтеграцію з сучасними інструментами UI-дизайну та стилізації, такими як Tailwind CSS або Styled Components. Це надає широкі можливості для побудови привабливого, адаптивного й інтуїтивно зрозумілого інтерфейсу.

3.4 Результати реалізації інформаційної системи

У рамках виконання проєкту було розроблено систему моніторингу та аналізу лог-файлів в режимі реального часу.

Ця система надає комплексне рішення для агрегації логів у реальному часі, моніторингу, оповіщення та звітності з різних джерел. Вона дозволяє організаціям централізувати інфраструктуру логування, виявляти аномалії та отримувати практичні інсайти з логів додатків та систем.

Основними можливостями системи є:

- Централізований збір: Агрегує логи з різних джерел (додатки, сервери, сервіси, бази даних)
- Гнучке налаштування джерел: Кожне джерело можна унікально ідентифікувати та налаштувати з власним описом
- Часове відстеження: Точні часові мітки з урахуванням часових поясів для коректного порядку подій

- Рівні важливості: Класифікація записів за важливістю (DEBUG, INFO, WARN, ERROR)
- Структуровані метадані: Зберігання додаткової контекстної інформації у форматі JSON для розширеного фільтрування та аналізу
- Оповіщення за умовами: Налаштування правил сповіщення з використанням JSON-умов
- Моніторинг у реальному часі: Безперервне порівняння вхідних логів з умовами оповіщення
- Гнучкі сповіщення: Налаштування повідомлень і механізмів доставки
- Атрибуція користувачів: Відстеження авторів оповіщень і спільне керування ними
- Багатоформатна генерація: Створення звітів у різних форматах (PDF, CSV, JSON тощо)
- Конфігурація на основі шаблонів: Використання JSON для налаштування параметрів та форматування звітів
- Багатоканальна доставка: Автоматична розсилка звітів кільком зацікавленим сторонам
- Моніторинг виконання: Відстеження статусу генерації, продуктивності та обробки помилок

В майбутньому я бачу такі шляхи розвитку системи:

- ✓ Інтеграція з системами штучного інтелекту: Використання ML/AI для автоматичного виявлення аномалій та прогнозування інцидентів
- ✓ Візуалізація в реальному часі: Побудова інтерактивних дашбордів для моніторингу ключових подій та показників
- ✓ Розширена аналітика: Підтримка складних запитів для аналізу поведінкових шаблонів і кореляції подій

- ✓ Глибша інтеграція з DevOps-інструментами: Автоматичні реакції на події, інтеграція з CI/CD пайплайнами
- ✓ Розгортання за допомогою сучасних хмарних рішень: Деплой системи на сучасні хмарні рішення (AWS, DigitalOcean)

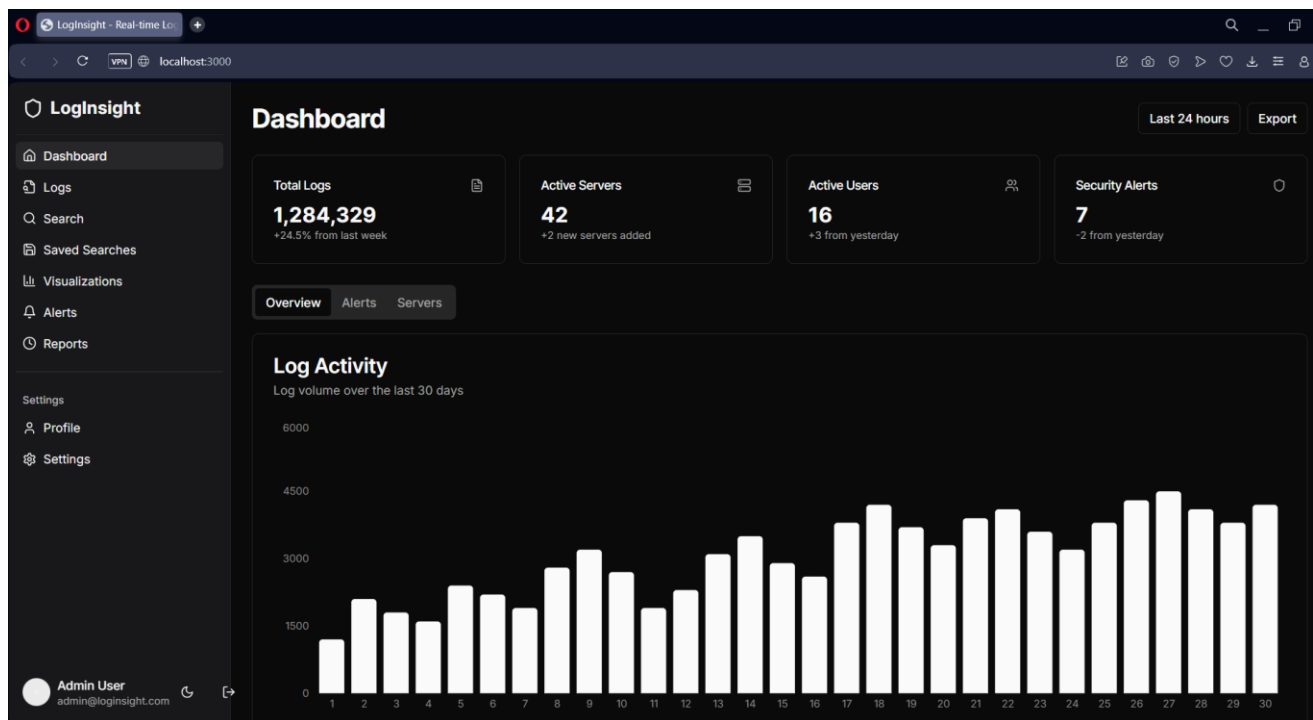


Рис.3.3 – Сторінка дашборду

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

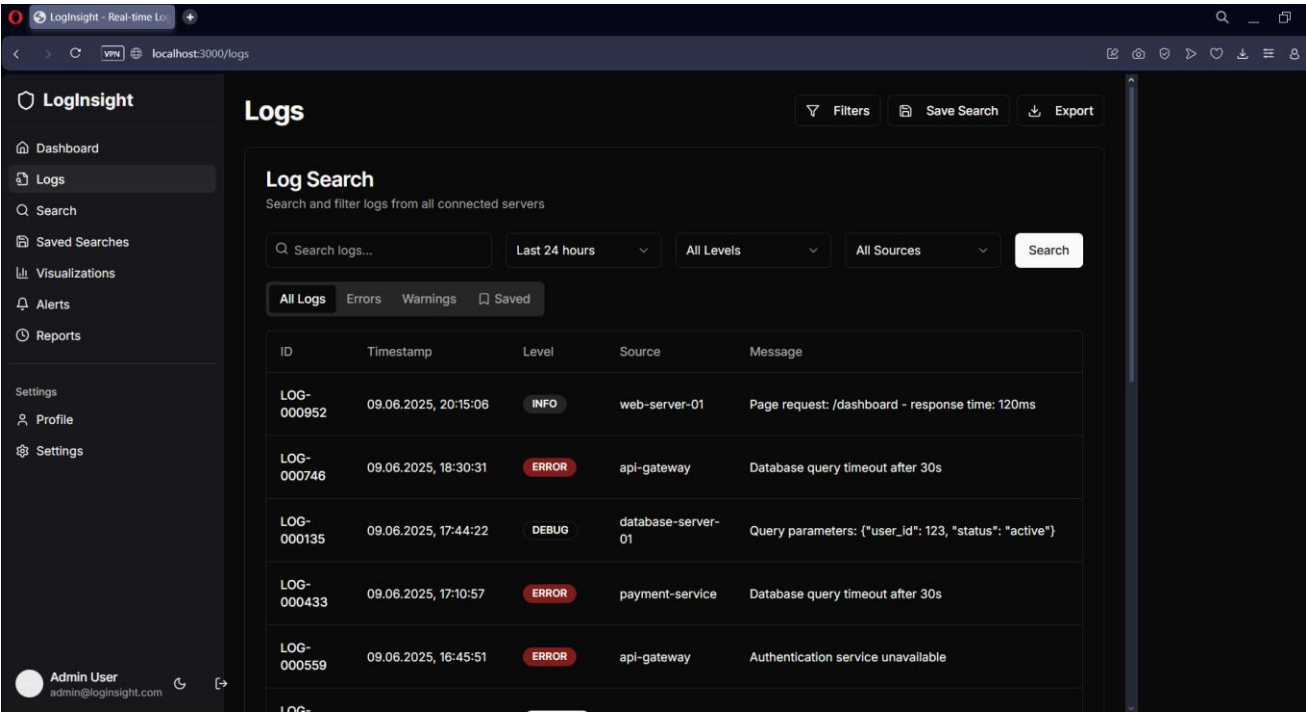


Рис.3.4 – Сторінка лог-файлів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

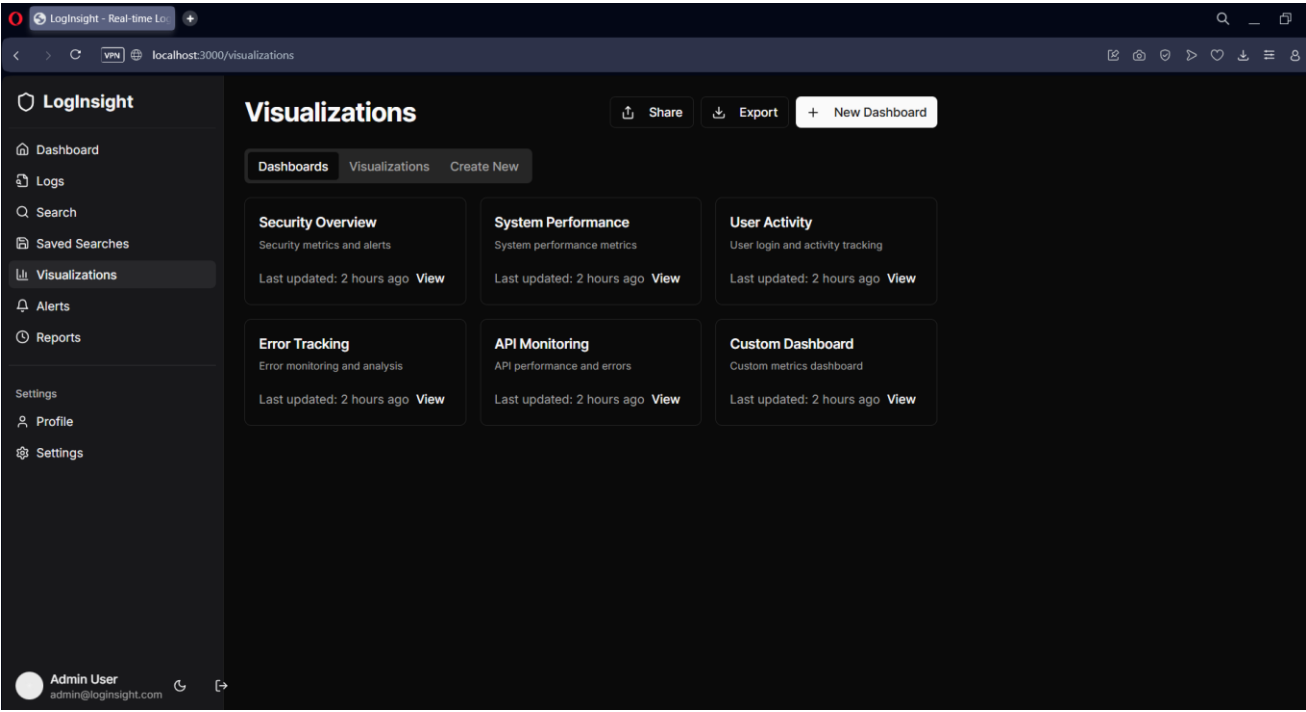


Рис.3.5 – Сторінка налаштувань візуалізації дашборду

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

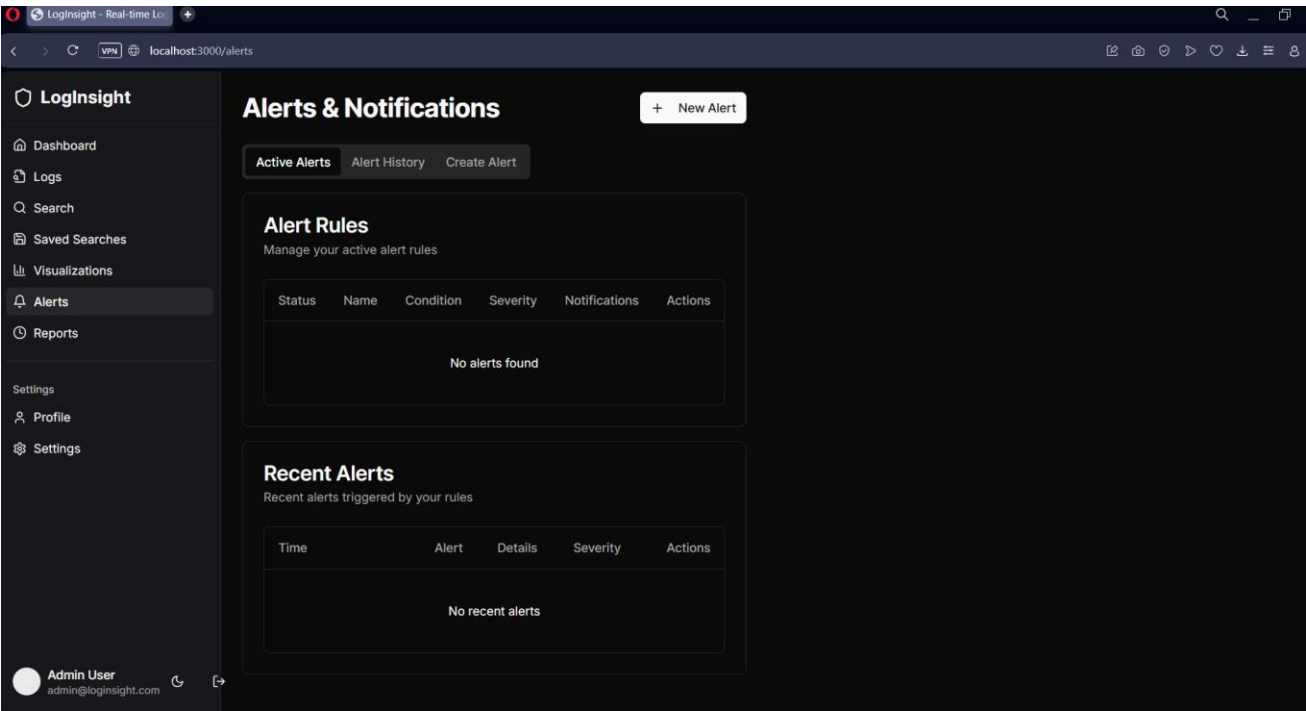


Рис.3.6 – Сторінка з повідомленнями

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

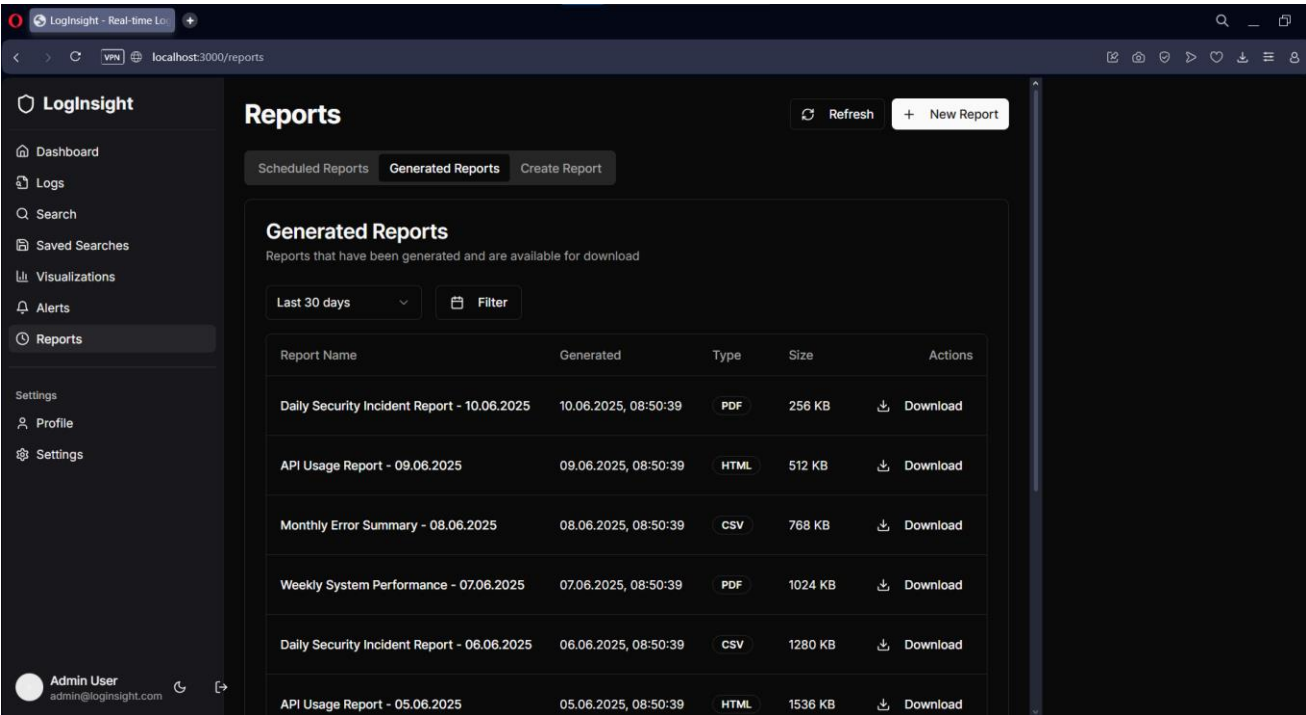


Рис.3.7 – Сторніка зі звітами

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

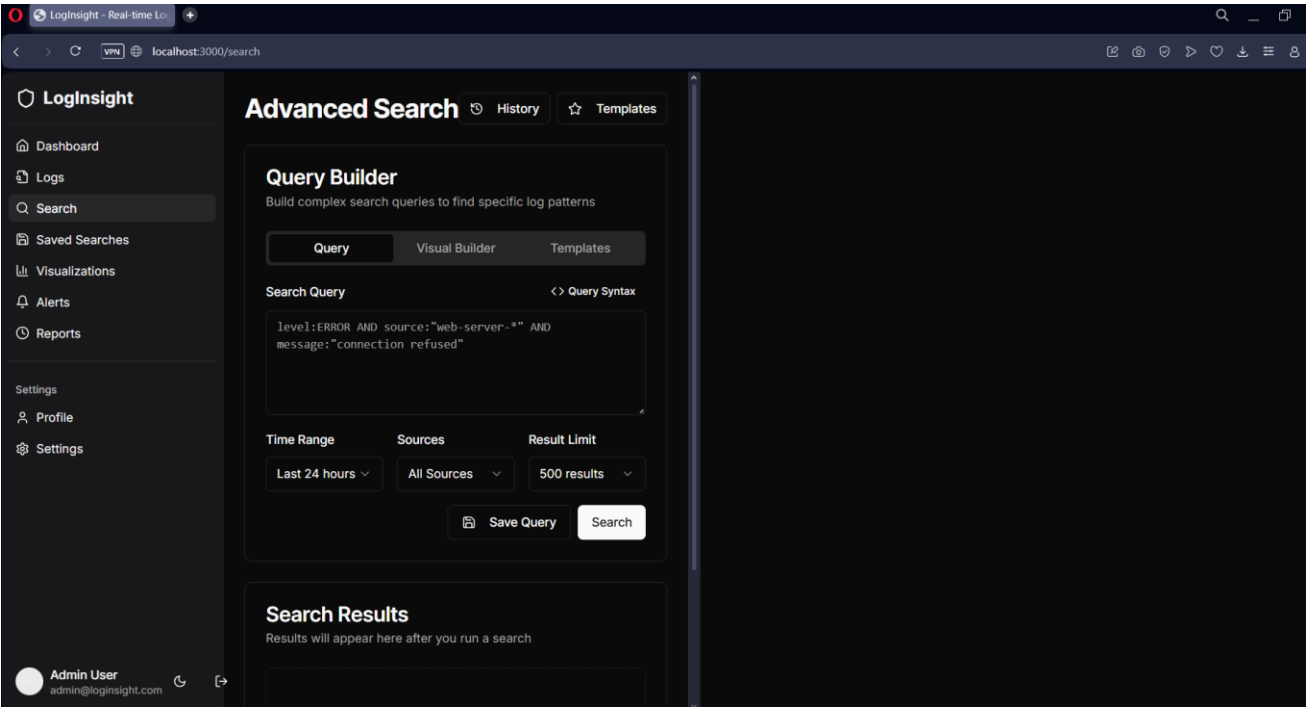


Рис.3.8 – Сторінка пошуку

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

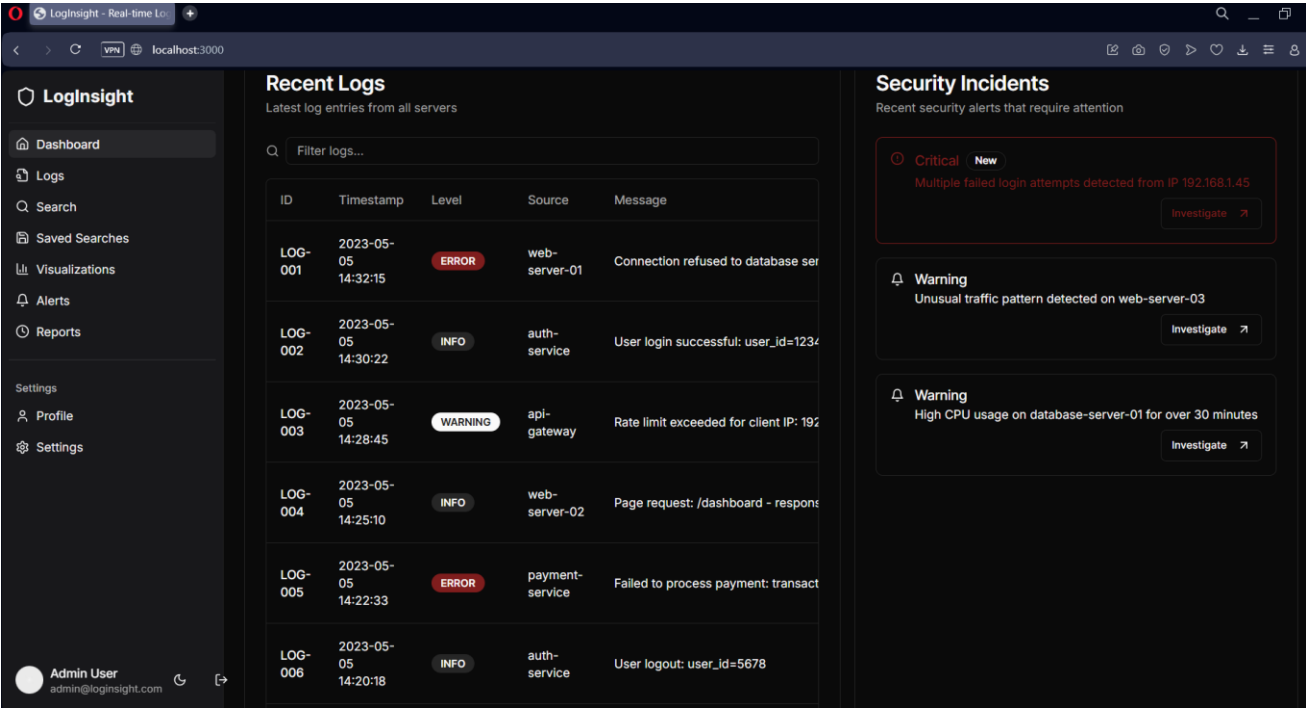


Рис.3.9 – Приклад лог-файлів на сторінці дашборду

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

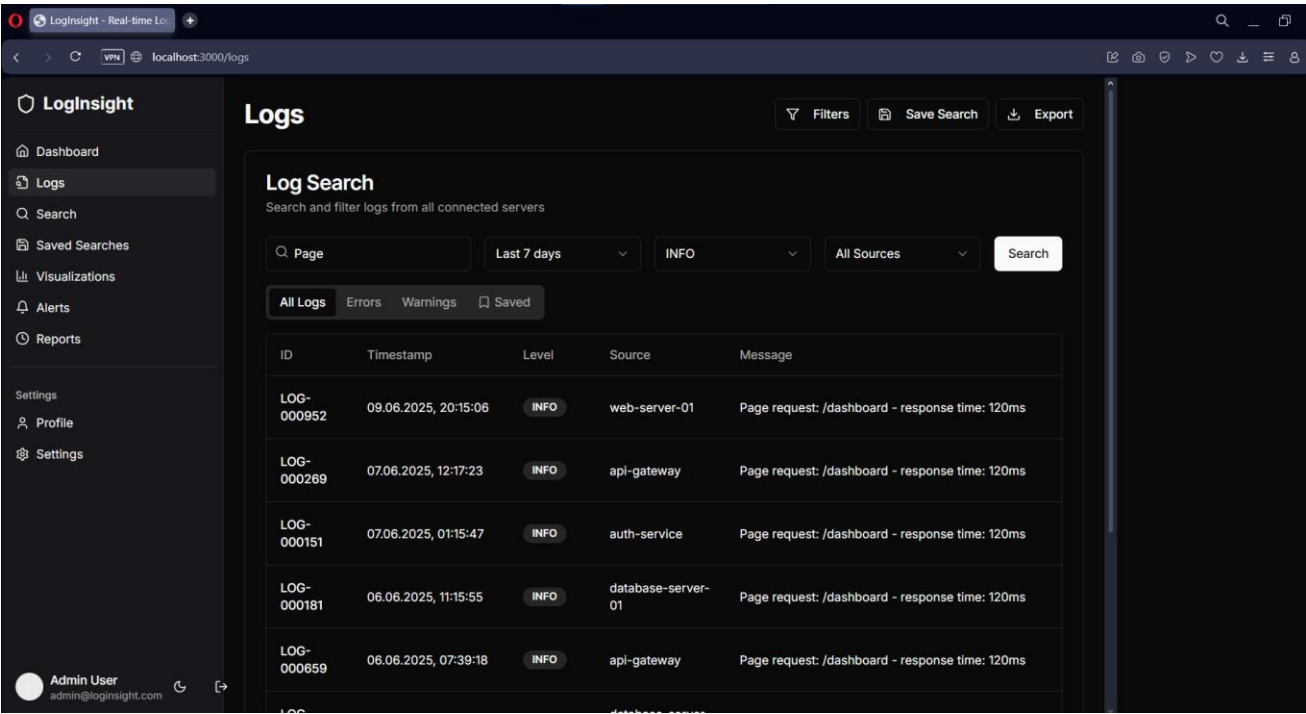


Рис.3.10 – Приклад пошуку та фільтрації логів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

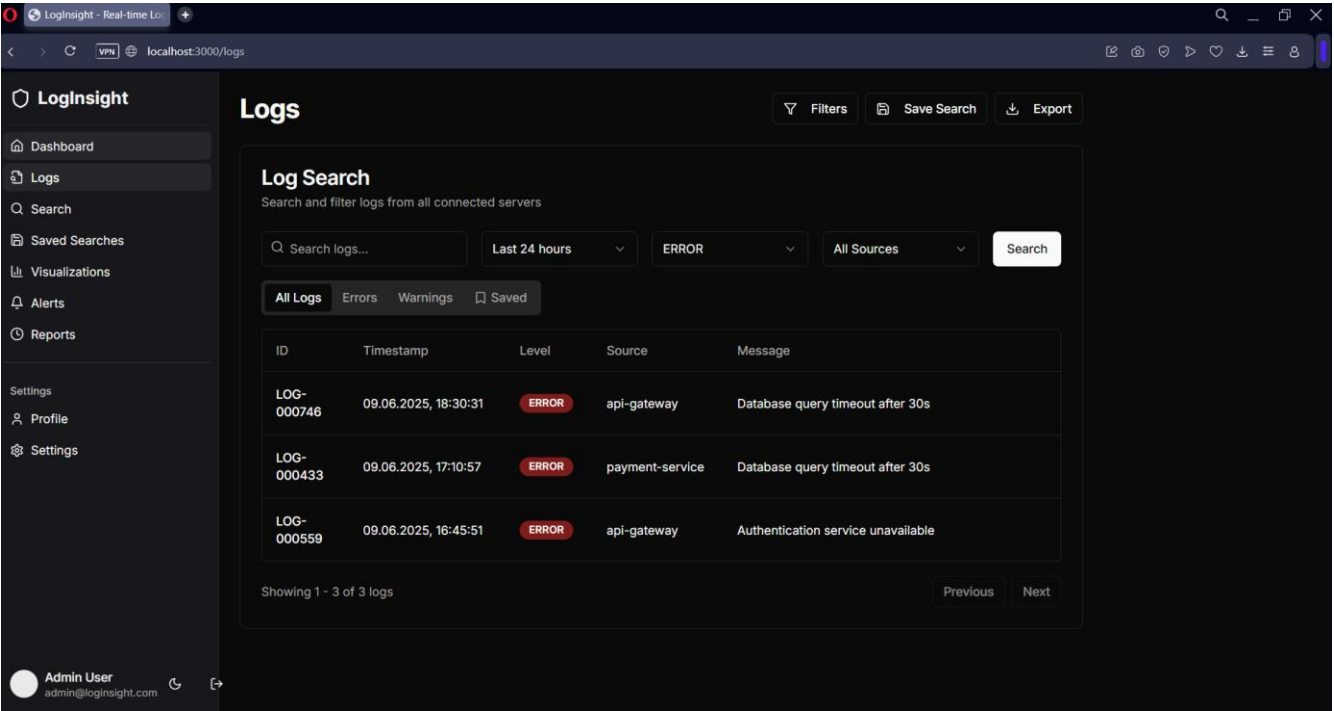


Рис.3.11 – Приклада пошуку та фільтрації логів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

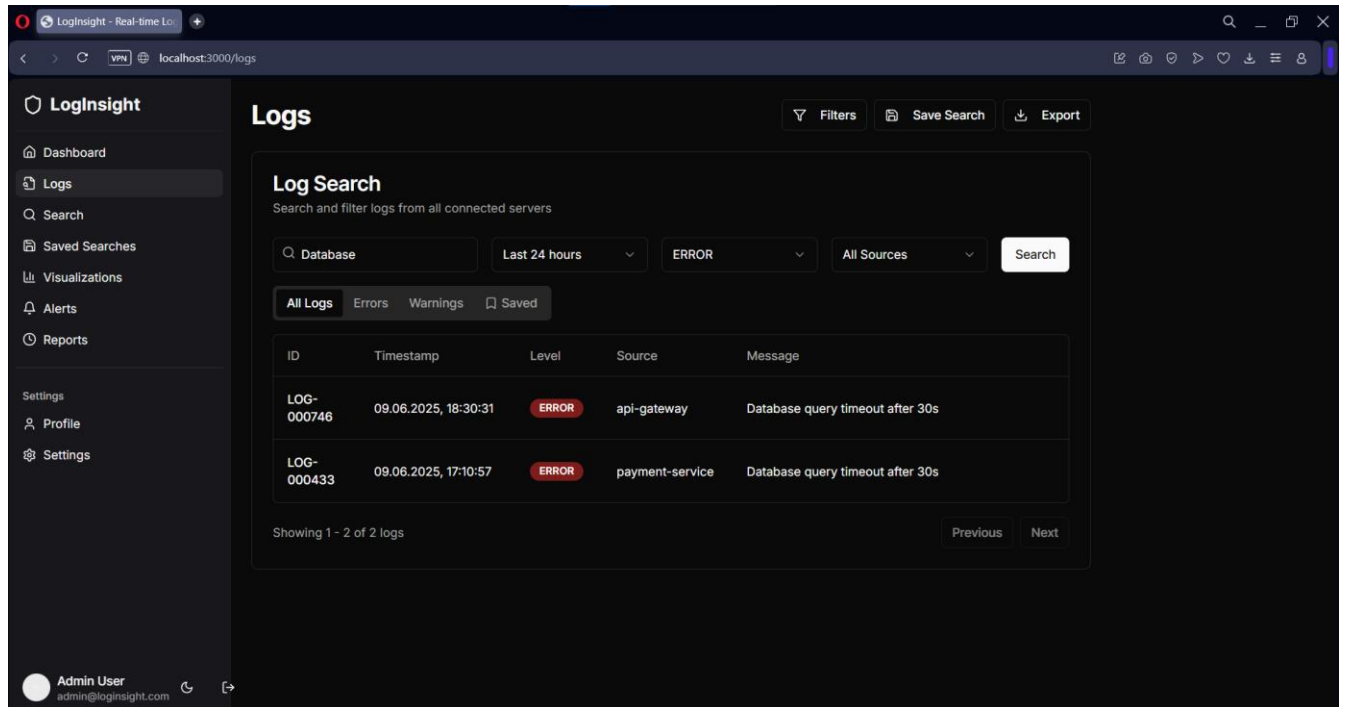


Рис.3.12 – Приклад пошуку та фільтрації логів

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

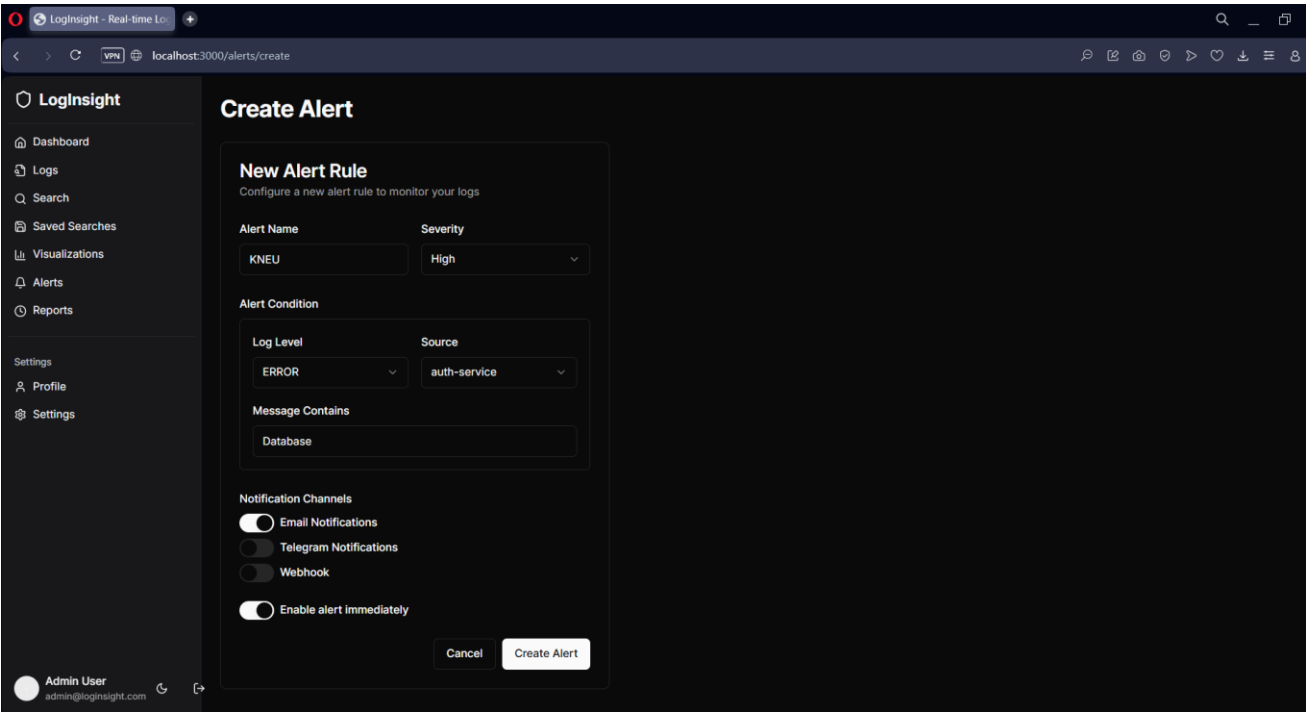


Рис.3.13 – Сторінка створення триггеру

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

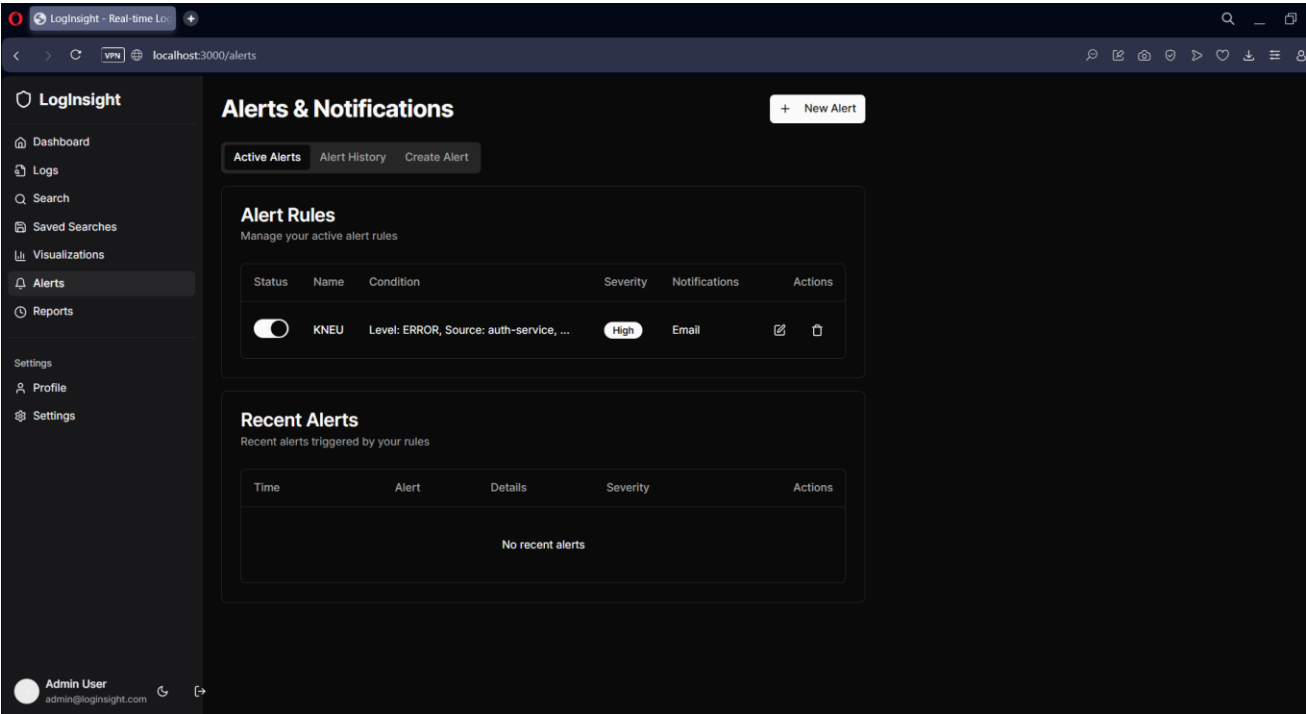


Рис.3.14 – Приклад створеного триггеру

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

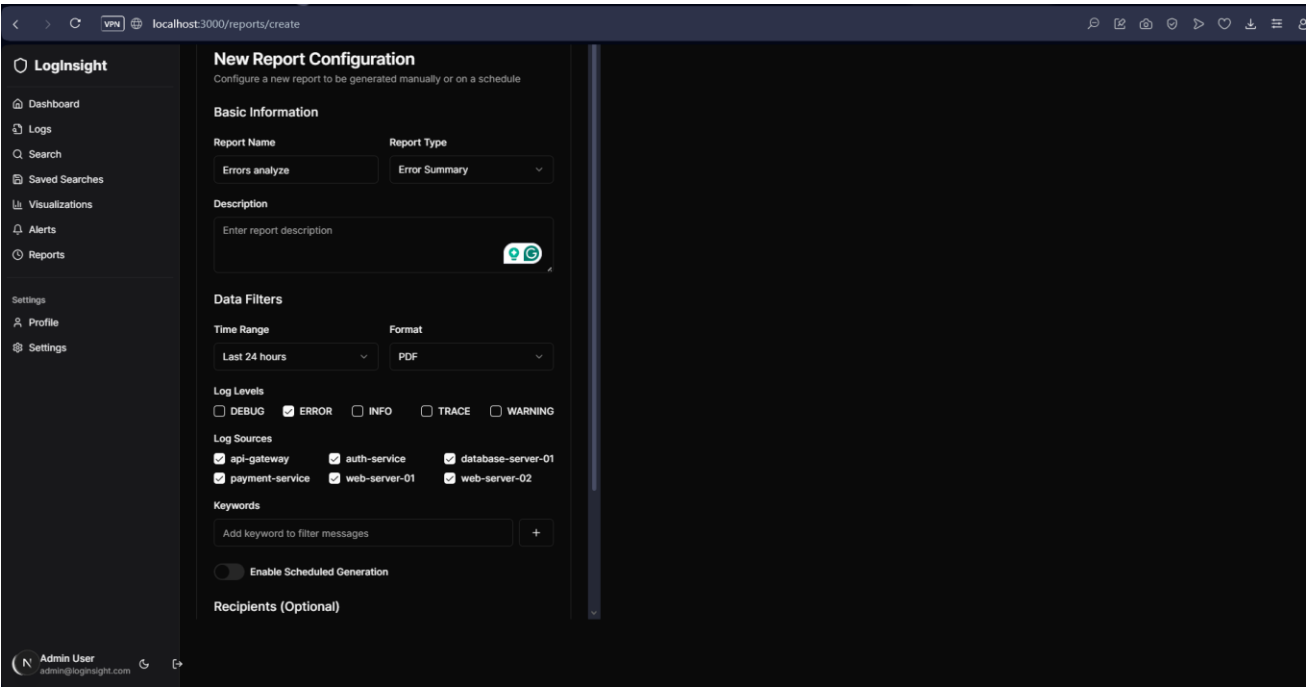


Рис.3.15 – Сторінка створення звіту

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

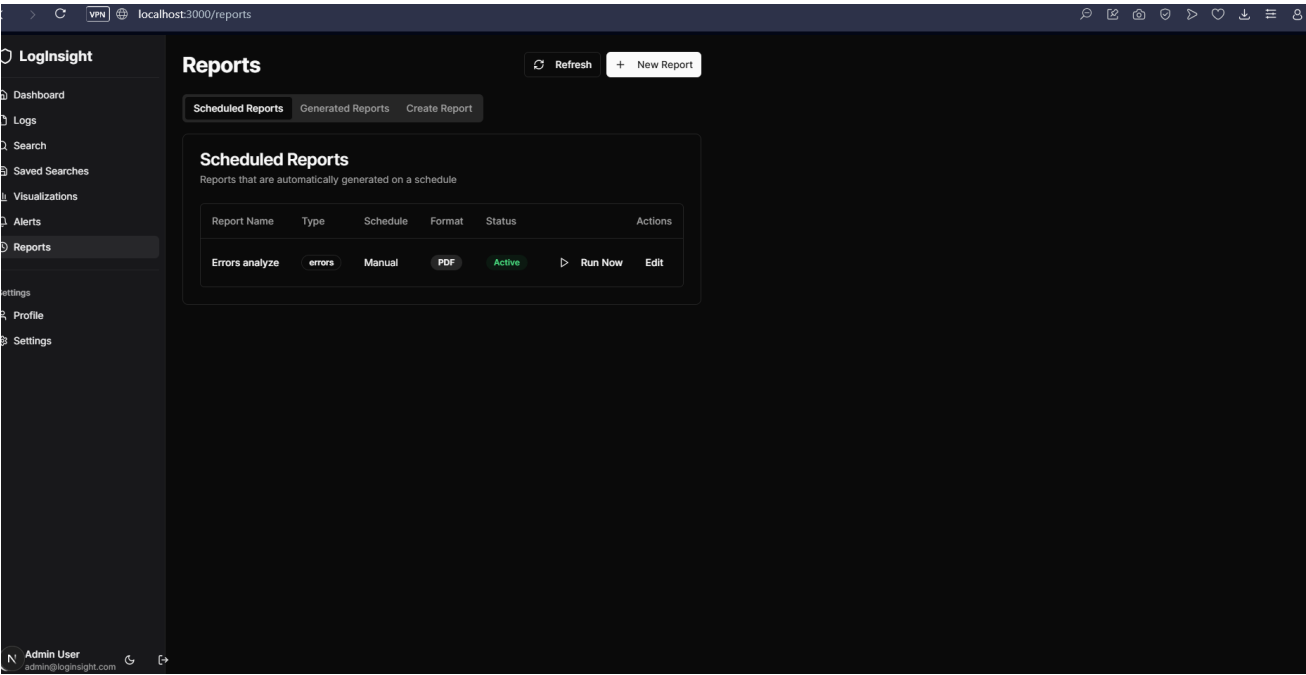


Рис.3.16 – Приклад звіту готово до генерації

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

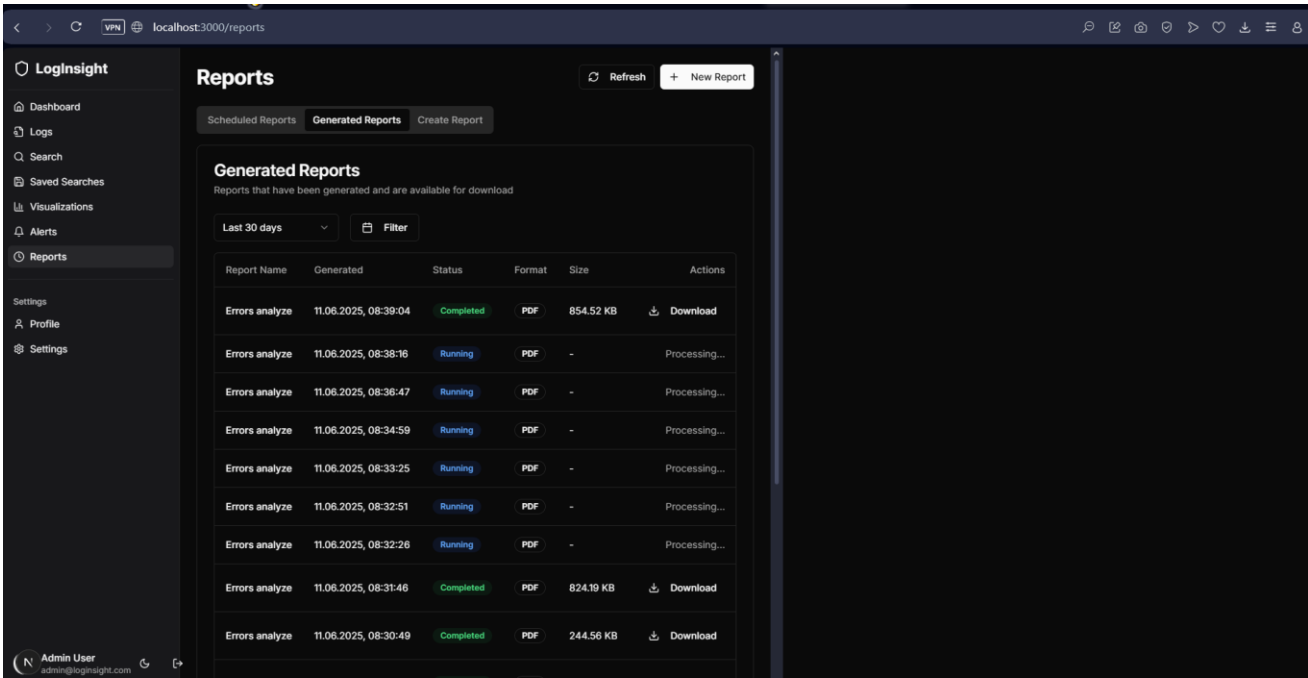


Рис.3.17 – Згенеровані звіти

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

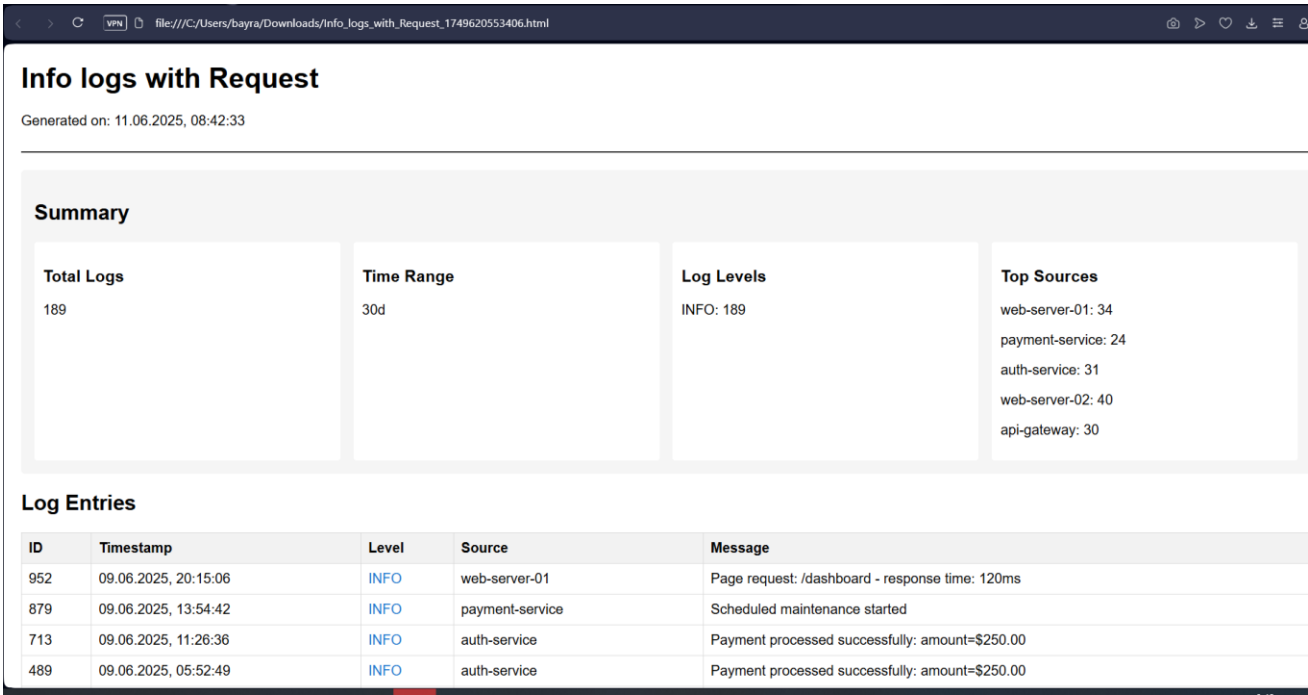


Рис.3.18 – Приклад згенерованого звіту

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

РОЗДІЛ 4

ТЕСТУВАННЯ І ВЕРИФІКАЦІЯ ПРОЕКТНИХ РІШЕНЬ

4.1. Розробка тест-кейсів

Тест-кейс — це набір вхідних значень, попередніх умов виконання, очікуваних результатів і післяумов, розроблених для конкретної мети або умови тестування.[26] Наприклад, для виконання певного програмного шляху або для перевірки відповідності певній вимозі. Вони:

- дозволяють зрозуміти покриття вимог системи;
- допомагають не забути, що саме протестували;
- систематизують всю подальшу.

Модульні тест-кейси призначені для перевірки окремих компонентів системи, таких як парсинг логів або фільтрація даних. Такі кейси повинні бути ізольованими й виконуватися швидко, дозволяючи запускати сотні тестів за секунди. У межах проєкту модульні тести використовуються для перевірки алгоритмів аналізу логів на коректність обробки різних форматів, а також для валідації функцій фільтрації та класифікації записів.

Інтеграційні тест-кейси орієнтовані на перевірку взаємодії між модулями системи, наприклад, процесу збору даних, їх обробки та подальшої візуалізації. Вони базуються на методі «чорного ящика» та тестують поведінку системи через доступні інтерфейси компонентів. Такі кейси застосовуються для перевірки передачі даних між модулем збору логів та сховищем, а також для контролю коректної роботи механізму сповіщень у разі виявлення аномалій.

Системні тест-кейси виконуються для оцінки роботи всієї системи в умовах реального середовища. Вони включають перевірку показників продуктивності та стабільності функціонування. Це означає тестування здатності

обробляти великі обсяги даних у реальному часі та перевірку відмовостійкості системи за умови високого навантаження.

Приймальні тест-кейси застосовуються для валідації відповідності системи встановленим бізнес-вимогам. Вони часто охоплюють сценарії, що імітують реальне використання системи кінцевими користувачами. Такі кейси дозволяють перевірити зручність інтерфейсу для аналітиків, а також протестувати формування звітів у заданих форматах та часових проміжках.

Тест-кейси сценаріїв представляють собою високорівневі описи дій користувача або бізнес-процесів, які підлягають перевірці в системі. Основна увага приділяється тому, що саме необхідно тестувати, без заглиблення в технічні деталі реалізації. Такі сценарії описують наскрізні процеси або функціональні блоки, наприклад, аналіз нового лог-файлу або реакцію системи на виявлені аномалії. Наприклад, тести, що перевіряють відправлення сповіщення при виявленні ключового слова «ERROR», або ж ситуацію, коли система не реагує на незначні попередження.

Негативні тест-кейси спрямовані на перевірку поведінки системи у разі некоректних, неочікуваних або граничних вхідних даних чи дій з боку користувача. Їхнє призначення - забезпечити стабільність та безпечність роботи системи навіть у разі помилкових умов. Це можуть бути ситуації, коли здійснюється завантаження файлу з некоректним форматом надходить порожній лог-файл, система отримує надто великий обсяг даних, що перевищує встановлений ліміт, або надходить запит без необхідних параметрів таких як ідентифікатор джерела логів.

Комплексне тестування системи моніторингу та аналізу лог-файлів в режимі реального часу включає:

- Авторизація - перевірка процесу входу користувача в систему та ідентифікації його облікового запису.

- Генерація звіту - тестування функціональності створення звітів на основі даних логування.
- Експорт звіту - перевірка можливості експортування згенерованих звітів у зовнішні формати.
- Налаштування дашборду - тестування функцій персоналізації інтерфейсу моніторингу для відображення потрібних метрик.
- Налаштування тригера сповіщень - перевірка можливості створення умов для автоматичних сповіщень при певних подіях.
- Отримання дозволів користувача - тестування отримання інформації про права доступу для конкретного користувача.
- Отримання ролей користувача - перевірка функціональності перегляду призначених ролей для користувача.
- Перегляд користувача - тестування функцій перегляду інформації про обліковий запис користувача.
- Перегляд логів певного часового проміжку - перевірка фільтрації логів за обраним періодом часу.
- Призначення ролей користувачу - тестування функціональності зміни ролей доступу для облікових записів.
- Спрацювання тригера - перевірка коректності роботи механізму тригерів при настанні відповідних умов.
- Фільтрація логів за рівнем - тестування можливості відфільтрувати записи логів за рівнями важливості.

Детальна тестова документація наведена у додатку А.

4.2. Трасування вимог до системи

Трасування вимог забезпечує зв'язок між вимогами та відстеження джерел вимог. Трасування є фундаментальною основою проведення аналізу впливу (impact analysis) при зміні вимог, допомагаючи передбачати ефект від внесення таких змін. Трасування передбачає спрямований зв'язок (подається у вигляді складного спрямованого ациклічного графа) між вимогами, тобто залежності.

Вимоги (B) аолодіють зворотною залежністю (тобто вторинні) по відношенню до вимог (A) і зацікавлених осіб, які є джерелом або утворюють причину появи аналізованих вимог (B). І, навпаки, вимоги (A) трасуються безпосередньо до тих вимог (B) і елементів дизайну (наприклад, моделі або, в загальному випадку, коду, запитів на зміни і т.п.), які породжуються або задовольняють вимогам (A).[27]

Завдяки трасуванню, розробники можуть легко відслідкувати, як зміна вимоги, наприклад, щодо нових правил фільтрації логів, вплине на вже існуючі модулі. Це дозволяє швидко оцінити масштаби змін та адаптувати систему без порушення суміжної логіки. Для тестувальників це означає, що кожна вимога буде мати прив'язаний до неї тест, який перевіряє її реалізацію. У свою чергу, менеджери можуть впевнено звітувати про ступінь реалізації проєкту, бачачи, які вимоги вже виконані, а які ще потребують розробки чи тестування.

Корисність трасування також проявляється в етапах супроводу системи: коли через кілька місяців потрібно внести зміни або провести аудит, саме трасування допоможе швидко знайти відповідні ділянки коду, документації або тестів, пов'язані з конкретною бізнес-вимогою.

Для якісного трасування необхідно мати чітко сформульовані вимоги, що не допускають подвійного трактування. Важливо, щоб кожна вимога мала унікальний ідентифікатор, що дозволяє посилатися на неї на будь-якому етапі розробки.

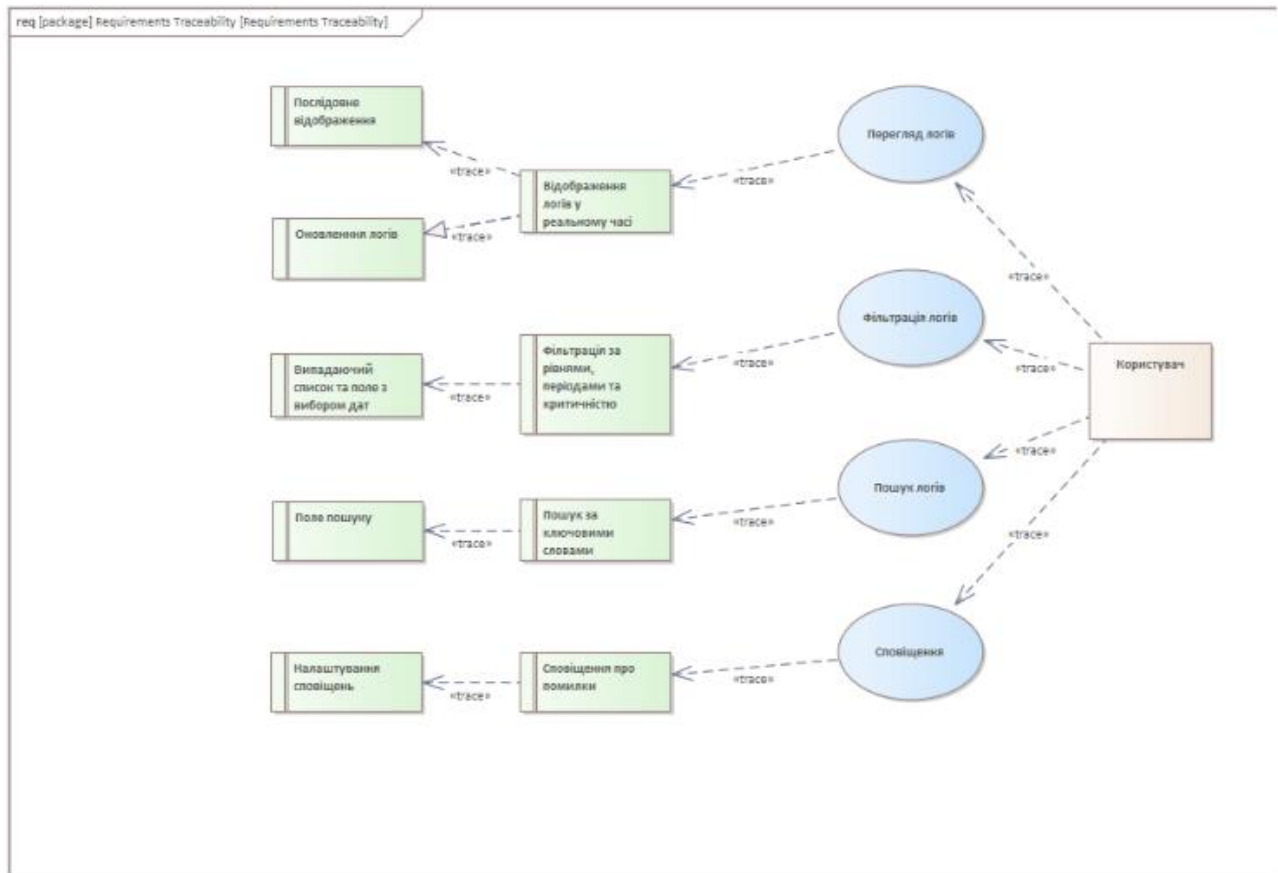


Рисунок 4.1 – Діаграма трасування системи

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

Target +	Requirements Traceability::Вина	Requirements Traceability::Відом	Requirements Traceability::Корн	Requirements Traceability::Нала	Requirements Traceability::Оно	Requirements Traceability::Пере	Requirements Traceability::Пол	Requirements Traceability::Посл	Requirements Traceability::Пов	Requirements Traceability::Пов	Requirements Traceability::Снов	Requirements Traceability::Снов	Requirements Traceability::Філь	Requirements Traceability::Філь
+ Source						↑				↑	↑			↑
Requirements Traceability::...														
Requirements Traceability::...														
Requirements Traceability::...		↑												
Requirements Traceability::...														
Requirements Traceability::...							↑							
Requirements Traceability::...								↑						
Requirements Traceability::...											↑			
Requirements Traceability::...				↑										
Requirements Traceability::...	↑													
Requirements Traceability::...													↑	

Рисунок 4.2 – Матриця взаємозв’язків

Джерело: сформовано (розроблено) автором на основі виконаного дослідження

ВИСНОВОК

У межах даного бакалаврського проєкту було розроблено систему моніторингу та аналізу лог-файлів в режимі реального часу, яка дозволяє виявляти критичні події, аналізувати поведінку системи та своєчасно реагувати на потенційні загрози. Під час реалізації проєкту здійснено комплексне дослідження предметної області, визначено ключові вимоги до функціональності системи та її технічних характеристик.

На етапі аналізу вивчено існуючі рішення для обробки логів(такі як ELK Stack). Їх аналіз допоміг визначити ефективні підходи до збору, фільтрації та візуалізації лог-файлів, а також сформувати технічне бачення проєктованої системи. Було обрано відповідну архітектуру, яка поєднує високу продуктивність, масштабованість і зручність у використанні.

Реалізацію виконано із застосуванням сучасного технологічного стеку, Next.JS, PostgreSQL та Nginx. Для забезпечення ефективної та швидкої розробки викростувався потужний текстовий редактор WebStorm та Postman як засіб для тестування. Інтерфейс було спроектовано з урахуванням потреб користувачів, що забезпечує інтуїтивну взаємодію з даними та оперативний доступ до критичних подій.

Під час розробки здійснено проектування бази даних, реалізацію бекенд-та фронтенд-частини, а також тестування основного функціоналу системи. Особливу увагу приділено питанням безпеки, стабільності та швидкодії під час обробки великої кількості логів у реальному часі.

Система демонструє переваги, що забезпечують її ефективність. Вона здатна аналізувати логи в реальному часі та забезпечує високу швидкість обробки запитів. Крім того, система пропонує можливість гнучкого масштабування відповідно до потреб користувачів та містить надійний механізм сповіщення про інциденти.

Щодо подальшого розвитку, існує три основні напрями вдосконалення. По-перше, доцільно додати підтримку машинного навчання для автоматичного виявлення аномалій у системі. По-друге, варто реалізувати рольову модель доступу, яка дозволить гнучко налаштовувати права користувачів залежно від їхніх функцій. Нарешті, важливим кроком буде розгортання системи на сучасних хмарних рішеннях для покращення її доступності та масштабованості.

Проект дозволив продемонструвати набуті знання з аналізу та обробки даних, роботи з потоковими сервісами, веб-технологіями та системами візуалізації. Результатом є працездатна система, яку можна надалі масштабувати, розширювати функціональність та інтегрувати з іншими інструментами інформаційної безпеки. Таким чином, поставлені цілі досягнуто, а розроблена система має потенціал для практичного використання у реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The ELK Stack: Elasticsearch, Logstash, and Kibana. Elastic. URL: <https://www.elastic.co/what-is/elk-stack>
2. Splunk Documentation. Splunk Inc. URL: <https://docs.splunk.com>
3. Graylog – Log Management Tool. Graylog Inc. URL: <https://www.graylog.org>
4. Datadog – Cloud Monitoring as a Service. Datadog Inc. URL: <https://www.datadoghq.com>
5. Sumo Logic – Continuous Intelligence Platform. Sumo Logic Inc. URL: <https://www.sumologic.com>
6. Real-Time Monitoring System Solutions: The Ultimate 2024 Guide. nTask. URL: <https://www.ntaskmanager.com/blog/real-time-monitoring-system-solutions-ultimate-2024-guide/>
7. Best Log Monitoring Tools. DNSstuff. URL: <https://www.dnsstuff.com/log-monitoring-tools>
8. What is a Log File? CrowdStrike. URL: <https://www.crowdstrike.com/en-us/cybersecurity-101/next-gen-siem/log-file/>
9. Best Log Monitoring Tools Comparison. Better Stack. URL: <https://betterstack.com/community/comparisons/log-monitoring-tools/>
10. Open Source Log Management Tools. SigNoz. URL: <https://signoz.io/blog/open-source-log-management/>
11. What is Log Monitoring? Middleware. URL: <https://middleware.io/blog/what-is-log-monitoring/>
12. Machine Learning for Log Analysis: A Survey. Science Direct. URL: <https://www.sciencedirect.com/science/article/pii/S2666827023000233>
13. What is Real-Time Log Monitoring? Mezmo. URL: <https://www.mezmo.com/learn-log-management/what-is-real-time-log-monitoring>
14. Benefits of Real-Time Analytics. Redpanda. URL: <https://www.redpanda.com/blog/benefits-of-real-time-analytics>
15. SIEM Log Management: What It Is and Why It's Vital for Cybersecurity. Bitlyft. URL: <https://www.bitlyft.com/resources/siem-log-management-what-it-is-and-why-its-vital-for-cybersecurity>
16. Log Analysis Tools and Best Practices. Uptrace. URL: <https://uptrace.dev/blog/log-analysis-tools.html>
17. A comprehensive survey on machine learning techniques for log analysis in cybersecurity. IET Research. URL: <https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/2024/8821891>
18. Which SIEM solutions offer the most advanced machine learning capabilities? LinkedIn. URL: <https://www.linkedin.com/advice/3/which-siem-solutions-offer-most-advanced-machine-ohrrf>

19. AI Anomaly Detection: Types, Methods, and Use Cases. TechMagic. URL: <https://www.techmagic.co/blog/ai-anomaly-detection>
20. Що таке ELK Stack? FreeHost. URL: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-elk-stack/>
21. Open Source Log Management Solutions. OpenObserve. URL: <https://openobserve.ai/resources/open-source-log-management-solutions>
22. Logging Best Practices. Better Stack. URL: <https://betterstack.com/community/guides/logging/logging-best-practices/>
23. Critical Log Management Best Practices. Exabeam. URL: <https://www.exabeam.com/explainers/log-management/7-critical-log-management-best-practices/>
24. Enterprise Architect User Guide. Sparx Systems. URL: <https://sparxsystems.com/bin/EAUserGuide.pdf>
25. 10 найкращих шаблонів документів для бізнес-вимог. Visure Solutions. URL: <https://visuresolutions.com/uk/вимоги-з-посібником-Word-Excel/найкращі-шаблони-документів-для-бізнес-вимог/>
26. Що таке функціональні вимоги. Visure Solutions. URL: <https://visuresolutions.com/uk/блог/функціональні-вимоги/>
27. Що таке нефункціональні вимоги. Visure Solutions. URL: <https://visuresolutions.com/uk/блог/нефункціональні-вимоги/>
28. Аналіз інформаційних систем. StudFiles. URL: <https://studfile.net/preview/5740844/page:11/>
29. What is OpenSearch. AWS. URL: <https://aws.amazon.com/what-is/opensearch/>
30. JavaScript Documentation. MDN. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/What_is_JavaScript
31. Next.js Documentation. Vercel. URL: <https://nextjs.org>
32. PostgreSQL Documentation. PostgreSQL Global development group. URL: <https://www.postgresql.org>

ДОДАТКИ

Додаток А

Тестова документація, підготовлена засобами Enterprise Architect

TESTING

Standard System test. Авторизація користувача

Description

1. Відкрити сторінку логіну
2. Ввести коректні дані в усі необхідні поля: ім'я, пароль
3. Натиснути кнопку "Увійти"

Input

- Ім'я користувача: existinguser
- Пароль: CorrectPassword123!

Acceptance Criteria

- Система повинна автентифікувати користувача з правильними обліковими даними

Results

- Користувач успішно увійшов до системи і переадресований на сторінку дашборду
Система створює сесію для користувача

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.1 – Тест кейс авторизація

Standard System test. Генерація звіту

Description

1. Перейти до сторінки перегляду логів
2. Вибрати логи для включення у звіт
3. Натиснути кнопку "Створити звіт"

Input

Вибрані логи

Acceptance Criteria

- Користувач повинен мати можливість вибирати логи для включення у звіт
- Система повинна генерувати звіт на основі вибраних логів

Results

- Система створює новий звіт з аналізом вибраних логів
- Звіт доступний для перегляду на сторінці звітів

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.2 – Тест кейс генерація звіту

 Standard System test. Експорт звіту

Description

1. Увійти в систему
2. Перейти до сторінки звітів
3. Вибрати звіт для експорту
4. Натиснути кнопку "Експортувати"

Input

ID звіту

Acceptance Criteria

Система повинна підтримувати експорт звітів

Results

- Система генерує файл звіту
- Користувач може завантажити згенерований файл

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.3 – Тест кейс експорт звіту

 Standard System test. Налаштування дашборту

Description

1. Увійти в систему
2. Перейти до налаштувань панелі моніторингу
3. Налаштувати відображення певних метрик та графіків
4. Зберегти налаштування
5. Перейти на панель моніторингу

Input

Налаштування користувача: {
 графік1: "Кількість помилок за годину",
 графік2: "Розподіл логів за джерелами",
 метрика1: "Загальна кількість критичних помилок"
 }

Acceptance Criteria

- Користувач повинен мати можливість налаштовувати власну панель моніторингу
- Налаштування повинні зберігатися між сесіями

Results

- Система зберігає налаштування користувача
- Панель моніторингу відображається згідно з користувацькими налаштуваннями
- При наступному вході користувач бачить свої попередні налаштування

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.4 – Тест кейс налаштування дашборду

 Standard System test. ~~Налаштування тригерів сповіщень~~

Description

1. ~~Увійти в систему як адміністратор~~
2. ~~Перейти до сторінки налаштувань сповіщень~~
3. ~~Створити новий тригер сповіщення~~
4. ~~Ввести умови тригера~~
5. ~~Зберегти налаштування~~

Input

- ~~Умова: level="ERROR"~~
- ~~ID тригера: 1~~
- ~~Активний: true~~
- ~~Критичність: Висока~~

Acceptance Criteria

- ~~Адміністратор повинен мати права на створення тригерів~~
- ~~Умова тригера повинна бути синтаксично коректною~~

Results

- ~~Новий тригер сповіщень створено та збережено в системі~~
- ~~Система почне відстежувати логи за вказаною умовою~~

~~[Last run Not Run at Invalid DateTime by and checked by .]~~

Рисунок А.5 – Тест кейс налаштування тригера сповіщень

 Standard System test. ~~Отримання дозволів користувача~~

Description

~~У профілі користувача натиснути "Переглянути дозволи"~~

Input

~~Користувач з роллю, яка має дозволи~~

Acceptance Criteria

~~Повертається перелік дозволених дій відповідно до ролі~~

Results

~~Відображається список дозволів~~

~~[Last run Not Run at Invalid DateTime by and checked by .]~~

Рисунок А.6 – Тест кейс отримання дозволів користувача

 Standard System test. ~~Отримання ролей користувача~~

Description

1. ~~Перейти до профілю користувача~~
2. ~~Відкрити вкладку "Ролі"~~

Input

~~Існуючий користувач з ID~~

Acceptance Criteria

~~Список ролей, що призначені користувачу, коректно відображається~~

Results

~~Ролі завантажено з бази й показано у вигляді таблиці~~

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.7 – Тест кейс отримання ролей користувача

 Standard System test. ~~Перегляд користувачів~~

Description

1. ~~Перейти на сторінку управління користувачами~~
2. ~~Натиснути "Показати всіх користувачів"~~

Input

~~ID користувача~~

Acceptance Criteria

- ~~Виводиться список користувачів~~
- ~~Інформація містить поля: id, name, role~~

Results

~~Користувачі відображені у вигляді таблиці~~

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.8 – Тест кейс перегляд користувача

 Standard System test. Перегляд логів певного часового проміжку

Description

1. Увійти в систему
2. Перейти до сторінки перегляду логів
3. Встановити часовий діапазон
4. Застосувати фільтр

Input

- Початкова дата: 2025-05-06 00:00:00
- Кінцева дата: 2025-05-07 23:59:59

Acceptance Criteria

Система повинна підтримувати фільтрацію логів за часовими проміжками

Results

- Система відображає лише ті логи, час створення яких входить у вказаний діапазон
- Логи поза вказаним діапазоном не відображаються

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.9 – Тест кейс перегляд логів певного часового проміжку

 Standard System test. Призначення ролей користувачеві

Description

1. Відкрити панель адміністратора
2. Вибрати користувача зі списку
3. Обрати роль зі списку (наприклад, "admin")
4. Натиснути "Призначити роль"

Input

Користувач існує в системі, роль — одна з дозволених: admin, viewer, editor

Acceptance Criteria

- Роль успішно призначена користувачеві
- У базі з'являється запис про нову роль у таблиці UserRole

Results

Користувач отримує права згідно з новою роллю, зміни збережено в системі

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.10 – Тест кейс призначення ролей користувачу

Standard System test. Тестування спрацювання тригера

Description

1. Налаштувати тригер для певного типу логів
2. Імітувати появу логу, що відповідає умові тригера
3. Перевірити створення сповіщення

Input

- Тригер: рівень логу = "ERROR"
- Тестовий лог: {id: 101, level: "ERROR", message: "Test error message", source: "TestApp", timestamp: поточний час}

Acceptance Criteria

- Система повинна перевіряти вхідні логи на відповідність умовам тригерів
- Для кожного спрацювання тригера має створюватися сповіщення

Results

- Система виявляє відповідність логу умові тригера
- Створюється нове сповіщення з відповідними даними
- Сповіщення надсилається користувачам згідно з налаштуваннями

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.11 – Тест кейс спрацювання тригера

Standard System test. Фільтрація логів за рівнем

Description

1. Увійти в систему
2. Перейти до сторінки перегляду логів
3. Вибрати опцію фільтрації за рівнем "ERROR"
4. Застосувати фільтр

Input

Фільтр: рівень = "ERROR"

Acceptance Criteria

Система повинна підтримувати фільтрацію логів за різними критеріями

Results

- Система відображає лише ті логи, рівень яких відповідає "ERROR"
- Інші логи приховані

[Last run Not Run at Invalid DateTime by and checked by .]

Рисунок А.12 – Тест кейс фільтрація логів за рівнем

Додаток Б

Приклади коду для системи

Приклад коду для роботи з повідомленнями

```

"use server"

import { db, eq } from "@lib/db"
import { alerts, alertEvents, logEntries, logLevels, logSources } from
"@lib/schema"
import { revalidatePath } from "next/cache"

export interface AlertCondition {
  level?: string
  source?: string
  message_contains?: string
  threshold?: number
  time_window?: number
}

export interface AlertData {
  name: string
  condition: AlertCondition
  notification: string
  createdBy: string
  isActive?: boolean
}

export async function createAlert(data: AlertData) {
  try {
    const result = await db
      .insert(alerts)
      .values({
        name: data.name,
        condition: data.condition,
        notification: data.notification,
        createdBy: data.createdBy,
        isActive: data.isActive ?? true,
      })
      .returning({ id: alerts.id })

    revalidatePath("/alerts")
    return result[0]
  } catch (error) {
    console.error("Error creating alert:", error)
    throw new Error("Failed to create alert")
  }
}

export async function getAlerts() {
  try {
    return await db.select().from(alerts).orderBy(alerts.createdAt)
  } catch (error) {
    console.error("Error fetching alerts:", error)
    throw new Error("Failed to fetch alerts")
  }
}

export async function getAlertById(id: number) {
  try {
    const result = await db.select().from(alerts).where(eq(alerts.id,
id)).limit(1)
    return result[0] || null
  } catch (error) {
    console.error(`Error fetching alert with ID ${id}:`, error)
    throw new Error("Failed to fetch alert")
  }
}

```

```

export async function updateAlert(id: number, data: Partial<AlertData>) {
  try {
    await db
      .update(alerts)
      .set({
        name: data.name,
        condition: data.condition,
        notification: data.notification,
        isActive: data.isActive,
      })
      .where(eq(alerts.id, id))

    revalidatePath("/alerts")
    return { success: true }
  } catch (error) {
    console.error(`Error updating alert with ID ${id}:`, error)
    throw new Error("Failed to update alert")
  }
}

export async function deleteAlert(id: number) {
  try {
    await db.delete(alerts).where(eq(alerts.id, id))
    revalidatePath("/alerts")
    return { success: true }
  } catch (error) {
    console.error(`Error deleting alert with ID ${id}:`, error)
    throw new Error("Failed to delete alert")
  }
}

export async function getAlertEvents(alertId?: number, limit = 20, offset = 0) {
  try {
    let query = db
      .select({
        id: alertEvents.id,
        triggeredAt: alertEvents.triggeredAt,
        message: alertEvents.message,
        alertId: alertEvents.alertId,
        alertName: alerts.name,
        logId: alertEvents.logEntryId,
        logMessage: logEntries.message,
        logLevel: logLevels.level,
        logSource: logSources.name,
      })
      .from(alertEvents)
      .leftJoin(alerts, eq(alertEvents.alertId, alerts.id))
      .leftJoin(logEntries, eq(alertEvents.logEntryId, logEntries.id))
      .leftJoin(logLevels, eq(logEntries.levelId, logLevels.id))
      .leftJoin(logSources, eq(logEntries.sourceId, logSources.id))
      .orderBy(alertEvents.triggeredAt)

    if (alertId) {
      query = query.where(eq(alertEvents.alertId, alertId))
    }

    return await query.limit(limit).offset(offset)
  } catch (error) {
    console.error("Error fetching alert events:", error)
    throw new Error("Failed to fetch alert events")
  }
}

```

Приклад коду для роботи з логами

```
"use server"

import { db, eq, and, desc, gte, lte, like } from "@lib/db"
import { logEntries, logLevels, logSources } from "@lib/schema"
import { sql } from "drizzle-orm"

export type LogLevel = "ERROR" | "WARNING" | "INFO" | "DEBUG" | "TRACE" | "all"
export type TimeRange = "15m" | "1h" | "24h" | "7d" | "30d" | "custom"

export interface LogSearchParams {
  query?: string
  level?: LogLevel
  source?: string
  timeRange: TimeRange
  startDate?: Date
  endDate?: Date
  limit?: number
  offset?: number
}

export async function getLogs({
  query,
  level,
  source,
  timeRange,
  startDate,
  endDate,
  limit = 20,
  offset = 0,
}: LogSearchParams) {
  try {
    let timeFilter = {}
    const now = new Date()

    if (timeRange === "custom" && startDate && endDate) {
      timeFilter = and(gte(logEntries.timestamp, startDate),
lte(logEntries.timestamp, endDate))
    } else {
      const minutes =
        timeRange === "15m"
          ? 15
          : timeRange === "1h"
            ? 60
            : timeRange === "24h"
              ? 1440
              : timeRange === "7d"
                ? 10080
                : timeRange === "30d"
                  ? 43200
                  : 43200 // Default to 30 days

      const pastDate = new Date(now.getTime() - minutes * 60 * 1000)
      timeFilter = gte(logEntries.timestamp, pastDate)
    }

    const query_conditions = []

    query_conditions.push(timeFilter)

    if (level && level !== "all") {
```

```

    const levelRecord = await
db.select().from(logLevels).where(eq(logLevels.level, level)).limit(1)
    if (levelRecord.length > 0) {
        query_conditions.push(eq(logEntries.levelId, levelRecord[0].id))
    }
}

    if (source && source !== "all") {
        const sourceRecord = await
db.select().from(logSources).where(eq(logSources.name, source)).limit(1)
        if (sourceRecord.length > 0) {
            query_conditions.push(eq(logEntries.sourceId, sourceRecord[0].id))
        }
    }

    if (query && query.trim() !== "") {
        query_conditions.push(like(logEntries.message, `%${query}%`))
    }

    const logs = await db
        .select({
            id: logEntries.id,
            message: logEntries.message,
            timestamp: logEntries.timestamp,
            context: logEntries.context,
            level: logLevels.level,
            source: logSources.name,
        })
        .from(logEntries)
        .leftJoin(logLevels, eq(logEntries.levelId, logLevels.id))
        .leftJoin(logSources, eq(logEntries.sourceId, logSources.id))
        .where(and(...query_conditions))
        .orderBy(desc(logEntries.timestamp))
        .limit(limit)
        .offset(offset)

    const countResult = await db
        .select({ count: sql<number>`count(*)` })
        .from(logEntries)
        .leftJoin(logLevels, eq(logEntries.levelId, logLevels.id))
        .leftJoin(logSources, eq(logEntries.sourceId, logSources.id))
        .where(and(...query_conditions))

    const total = countResult[0]?.count || 0

    return { logs, total }
} catch (error) {
    console.error("Error fetching logs:", error)
    throw new Error("Failed to fetch logs")
}
}

export async function getLogSources() {
    try {
        return await db.select().from(logSources).orderBy(logSources.name)
    } catch (error) {
        console.error("Error fetching log sources:", error)
        throw new Error("Failed to fetch log sources")
    }
}

export async function getLogLevels() {
    try {
        return await db.select().from(logLevels).orderBy(logLevels.level)
    }

```

```

    } catch (error) {
      console.error("Error fetching log levels:", error)
      throw new Error("Failed to fetch log levels")
    }
  }

export async function getLogById(id: number) {
  try {
    const logs = await db
      .select({
        id: logEntries.id,
        message: logEntries.message,
        timestamp: logEntries.timestamp,
        context: logEntries.context,
        level: logLevels.level,
        source: logSources.name,
      })
      .from(logEntries)
      .leftJoin(logLevels, eq(logEntries.levelId, logLevels.id))
      .leftJoin(logSources, eq(logEntries.sourceId, logSources.id))
      .where(eq(logEntries.id, id))
      .limit(1)

    return logs[0] || null
  } catch (error) {
    console.error(`Error fetching log with ID ${id}:`, error)
    throw new Error("Failed to fetch log")
  }
}

```

Приклад коду для роботи зі звітами

```

"use server"

import { db, eq, desc, and, gte, lte, like } from "@lib/db"
import { reports, reportExecutions, logEntries, logLevels, logSources } from
"@lib/schema"
import { revalidatePath } from "next/cache"
import { sql } from "drizzle-orm"

export interface ReportConfig {
  time_range: string
  start_date?: string
  end_date?: string
  levels?: string[]
  sources?: string[]
  keywords?: string[]
  group_by?: string[]
  metrics?: string[]
}

export interface ReportSchedule {
  frequency: "once" | "daily" | "weekly" | "monthly"
  time: string
  day?: number | string
  timezone?: string
}

export interface ReportData {
  name: string
  description?: string
  reportType: "security" | "performance" | "errors" | "usage" | "custom"
  config: ReportConfig
}

```

```

    schedule?: ReportSchedule
    format: "pdf" | "csv" | "html" | "json"
    recipients?: string[]
    createdBy: string
    isActive?: boolean
  }

export async function createReport(data: ReportData) {
  try {
    const result = await db
      .insert(reports)
      .values({
        name: data.name,
        description: data.description,
        reportType: data.reportType,
        config: data.config,
        schedule: data.schedule,
        format: data.format,
        recipients: data.recipients,
        createdBy: data.createdBy,
        isActive: data.isActive ?? true,
      })
      .returning({ id: reports.id })

    revalidatePath("/reports")
    return result[0]
  } catch (error) {
    console.error("Error creating report:", error)
    throw new Error("Failed to create report")
  }
}

export async function getReports() {
  try {
    return await db.select().from(reports).orderBy(desc(reports.createdAt))
  } catch (error) {
    console.error("Error fetching reports:", error)
    throw new Error("Failed to fetch reports")
  }
}

export async function getReportById(id: number) {
  try {
    const result = await db.select().from(reports).where(eq(reports.id, id)).limit(1)
    return result[0] || null
  } catch (error) {
    console.error(`Error fetching report with ID ${id}:`, error)
    throw new Error("Failed to fetch report")
  }
}

export async function updateReport(id: number, data: Partial<ReportData>) {
  try {
    await db
      .update(reports)
      .set({
        name: data.name,
        description: data.description,
        reportType: data.reportType,
        config: data.config,
        schedule: data.schedule,
        format: data.format,
        recipients: data.recipients,

```

```

        isActive: data.isActive,
        updatedAt: new Date(),
    })
    .where(eq(reports.id, id))

    revalidatePath("/reports")
    return { success: true }
} catch (error) {
    console.error(`Error updating report with ID ${id}:`, error)
    throw new Error("Failed to update report")
}
}

export async function deleteReport(id: number) {
    try {
        await db.delete(reports).where(eq(reports.id, id))
        revalidatePath("/reports")
        return { success: true }
    } catch (error) {
        console.error(`Error deleting report with ID ${id}:`, error)
        throw new Error("Failed to delete report")
    }
}

export async function getReportExecutions(reportId?: number, limit = 20, offset
= 0) {
    try {
        let query = db
            .select({
                id: reportExecutions.id,
                reportId: reportExecutions.reportId,
                reportName: reports.name,
                status: reportExecutions.status,
                filePath: reportExecutions.filePath,
                fileSize: reportExecutions.fileSize,
                executionTime: reportExecutions.executionTime,
                errorMessage: reportExecutions.errorMessage,
                executedAt: reportExecutions.executedAt,
                completedAt: reportExecutions.completedAt,
                format: reports.format,
            })
            .from(reportExecutions)
            .leftJoin(reports, eq(reportExecutions.reportId, reports.id))
            .orderBy(desc(reportExecutions.executedAt))

        if (reportId) {
            query = query.where(eq(reportExecutions.reportId, reportId))
        }

        return await query.limit(limit).offset(offset)
    } catch (error) {
        console.error("Error fetching report executions:", error)
        throw new Error("Failed to fetch report executions")
    }
}

export async function generateReport(reportId: number, parameters?: any) {
    try {
        const execution = await db
            .insert(reportExecutions)
            .values({
                reportId,
                status: "running",
                parameters,
            })
    }
}

```

```

    })
    .returning({ id: reportExecutions.id })

    const executionId = execution[0].id

    const report = await getReportById(reportId)
    if (!report) {
        throw new Error("Report not found")
    }

    const reportData = await generateReportData(report.config, parameters)

    const startTime = Date.now()
    await new Promise((resolve) => setTimeout(resolve, 2000))
    const executionTime = Date.now() - startTime

    const fileName = `${report.name.replace(/\s+/g,
    "_")}_${Date.now()}.${report.format}`
    const filePath = `/reports/${fileName}`

    await db
        .update(reportExecutions)
        .set({
            status: "completed",
            filePath,
            fileSize: Math.floor(Math.random() * 1000000) + 50000, //
Simulate file size
            executionTime,
            completedAt: new Date(),
        })
        .where(eq(reportExecutions.id, executionId))

    revalidatePath("/reports")
    return { executionId, filePath, reportData }
} catch (error) {
    console.error("Error generating report:", error)
    throw new Error(error.message)
}
}

async function generateReportData(config: ReportConfig, parameters?: any) {
    try {
        let timeFilter = {}
        const now = new Date()

        if (config.start_date && config.end_date) {
            timeFilter = and(
                gte(logEntries.timestamp, new Date(config.start_date)),
                lte(logEntries.timestamp, new Date(config.end_date)),
            )
        } else {
            const minutes =
                config.time_range === "15m"
                    ? 15
                    : config.time_range === "1h"
                    ? 60
                    : config.time_range === "24h"
                    ? 1440
                    : config.time_range === "7d"
                    ? 10080
                    : config.time_range === "30d"
                    ? 43200
                    : 43200
        }
    }

```

```

    const pastDate = new Date(now.getTime() - minutes * 60 * 1000)
    timeFilter = gte(logEntries.timestamp, pastDate)
  }

  const conditions = [timeFilter]

  if (config.levels && config.levels.length > 0) {
    const levelIds = await db
      .select({ id: logLevels.id })
      .from(logLevels)
      .where(sql`${logLevels.level} IN (${sql.join(config.levels,
sql`, `))}`)

    if (levelIds.length > 0) {
      const ids = levelIds.map((l) => l.id)
      conditions.push(sql`${logEntries.levelId} IN (${sql.join(ids,
sql`, `))}`)
    }
  }

  if (config.sources && config.sources.length > 0) {
    const sourceIds = await db
      .select({ id: logSources.id })
      .from(logSources)
      .where(sql`${logSources.name} IN (${sql.join(config.sources,
sql`, `))}`)

    if (sourceIds.length > 0) {
      const ids = sourceIds.map((s) => s.id)
      conditions.push(sql`${logEntries.sourceId} IN (${sql.join(ids,
sql`, `))}`)
    }
  }

  if (config.keywords && config.keywords.length > 0) {
    const keywordConditions = config.keywords.map((keyword) =>
like(logEntries.message, `%${keyword}%`))
    conditions.push(sql`${keywordConditions.join(" OR ")}`)
  }

  const logs = await db
    .select({
      id: logEntries.id,
      message: logEntries.message,
      timestamp: logEntries.timestamp,
      level: logLevels.level,
      source: logSources.name,
      context: logEntries.context,
    })
    .from(logEntries)
    .leftJoin(logLevels, eq(logEntries.levelId, logLevels.id))
    .leftJoin(logSources, eq(logEntries.sourceId, logSources.id))
    .where(and(...conditions))
    .orderBy(desc(logEntries.timestamp))
    .limit(10000)

  const summary = {
    totalLogs: logs.length,
    timeRange: config.time_range,
    levelCounts: logs.reduce(
      (acc, log) => {
        acc[log.level || "UNKNOWN"] = (acc[log.level || "UNKNOWN"]
|| 0) + 1
        return acc
      }, {}
    )
  }

```

```

        },
        {} as Record<string, number>,
    ),
    sourceCounts: logs.reduce(
        (acc, log) => {
            acc[log.source || "UNKNOWN"] = (acc[log.source || "UNKNOWN"]
|| 0) + 1
            return acc
        },
        {} as Record<string, number>,
    ),
    generatedAt: new Date().toISOString(),
}

return { logs, summary }
} catch (error) {
    console.error("Error generating report data:", error)
    throw new Error(error.message)
}
}

export async function downloadReport(executionId: number) {
    try {
        const execution = await db
            .select({
                id: reportExecutions.id,
                filePath: reportExecutions.filePath,
                reportName: reports.name,
                format: reports.format,
                reportId: reportExecutions.reportId,
            })
            .from(reportExecutions)
            .leftJoin(reports, eq(reportExecutions.reportId, reports.id))
            .where(eq(reportExecutions.id, executionId))
            .limit(1)

        if (execution.length === 0) {
            throw new Error("Report execution not found")
        }

        const exec = execution[0]

        const report = await getReportById(exec.reportId)
        if (!report) {
            throw new Error("Report not found")
        }

        const reportData = await generateReportData(report.config)

        // Generate different formats
        let content: string
        let mimeType: string
        let filename: string

        switch (exec.format) {
            case "csv":
                content = generateCSVContent(reportData)
                mimeType = "text/csv"
                filename = `${exec.reportName.replace(/\s+/g,
"_" )}_${Date.now()}.csv`
                break
            case "json":
                content = JSON.stringify(reportData, null, 2)
                mimeType = "application/json"

```

```

        filename = `${exec.reportName.replace(/\s+/g,
"_" )}_${Date.now()}.json`
        break
      case "html":
        content = generateHTMLContent(reportData, exec.reportName)
        mimeType = "text/html"
        filename = `${exec.reportName.replace(/\s+/g,
"_" )}_${Date.now()}.html`
        break
      default:
        content = generateHTMLContent(reportData, exec.reportName)
        mimeType = "text/html"
        filename = `${exec.reportName.replace(/\s+/g,
"_" )}_${Date.now()}.html`
    }

    return {
      content,
      mimeType,
      filename,
    }
  } catch (error) {
    console.error("Error downloading report:", error)
    throw new Error("Failed to download report")
  }
}

function generateCSVContent(reportData: any): string {
  const { logs, summary } = reportData

  let csv = "ID,Timestamp,Level,Source,Message\n"

  logs.forEach((log: any) => {
    const escapedMessage = log.message.replace(/"/g, '"')
    csv +=
`${log.id},${log.timestamp},${log.level},${log.source},${escapedMessage}"
\n`
  })

  csv += "\n\nSUMMARY\n"
  csv += `Total Logs,${summary.totalLogs}\n`
  csv += `Time Range,${summary.timeRange}\n`
  csv += `Generated At,${summary.generatedAt}\n`

  csv += "\n\nLEVEL COUNTS\n"
  Object.entries(summary.levelCounts).forEach(([level, count]) => {
    csv += `${level},${count}\n`
  })

  csv += "\n\nSOURCE COUNTS\n"
  Object.entries(summary.sourceCounts).forEach(([source, count]) => {
    csv += `${source},${count}\n`
  })

  return csv
}

function generateHTMLContent(reportData: any, reportName: string): string {
  const { logs, summary } = reportData

  return `
<!DOCTYPE html>
<html>
<head>

```

```

<title>${reportName}</title>
<style>
  body { font-family: Arial, sans-serif; margin: 20px; }
  .header { border-bottom: 2px solid #333; padding-bottom: 10px; margin-bottom: 20px; }
  .summary { background: #f5f5f5; padding: 15px; margin-bottom: 20px; border-radius: 5px; }
  .summary-grid { display: grid; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 15px; }
  .summary-item { background: white; padding: 10px; border-radius: 3px; }
  table { width: 100%; border-collapse: collapse; margin-top: 20px; }
  th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
  th { background-color: #f2f2f2; }
  .level-error { color: #d32f2f; font-weight: bold; }
  .level-warning { color: #f57c00; font-weight: bold; }
  .level-info { color: #1976d2; }
  .level-debug { color: #388e3c; }
  .level-trace { color: #7b1fa2; }
</style>
</head>
<body>
  <div class="header">
    <h1>${reportName}</h1>
    <p>Generated on: ${new Date(summary.generatedAt).toLocaleString()}</p>
  </div>

  <div class="summary">
    <h2>Summary</h2>
    <div class="summary-grid">
      <div class="summary-item">
        <h3>Total Logs</h3>
        <p>${summary.totalLogs}</p>
      </div>
      <div class="summary-item">
        <h3>Time Range</h3>
        <p>${summary.timeRange}</p>
      </div>
      <div class="summary-item">
        <h3>Log Levels</h3>
        ${Object.entries(summary.levelCounts)
        .map(([level, count]) => `<p>${level}: ${count}</p>`)
        .join("")}
      </div>
      <div class="summary-item">
        <h3>Top Sources</h3>
        ${Object.entries(summary.sourceCounts)
        .slice(0, 5)
        .map(([source, count]) => `<p>${source}: ${count}</p>`)
        .join("")}
      </div>
    </div>
  </div>

  <h2>Log Entries</h2>
  <table>
    <thead>
      <tr>
        <th>ID</th>
        <th>Timestamp</th>
        <th>Level</th>
        <th>Source</th>
        <th>Message</th>
      </tr>
    </thead>

```

```

        <tbody>
          ${logs
            .slice(0, 1000)
            .map(
              (log: any) => `
                <tr>
                  <td>${log.id}</td>
                  <td>${new Date(log.timestamp).toLocaleString()}</td>
                  <td class="level-
${log.level?.toLowerCase()}>${log.level}</td>
                  <td>${log.source}</td>
                  <td>${log.message}</td>
                </tr>
              `
            )
            .join("")}
        </tbody>
      </table>

      ${
        logs.length > 1000
        ? `<p><em>Note: Only showing first 1000 entries out of
${logs.length} total logs.</em></p>`
        : ""
      }
    </body>
  </html>
`
}

```

Додаток В

Опис структури елементів бази даних

1. Опис таблиці “alert_events”

```
create table alert_events
(
    id          serial
        primary key,
    alert_id    integer not null
        references public.alerts
            on delete cascade,
    triggered_at timestamp default CURRENT_TIMESTAMP,
    log_entry_id bigint not null
        references public.log_entries
            on delete cascade,
    message     text
);
```

2. Опис таблиці “alerts”

```
create table alerts
(
    id          serial
        primary key,
    name        text not null,
    condition   jsonb not null,
    notification text not null,
    created_by  text not null,
    is_active   boolean default true,
    created_at  timestamp default CURRENT_TIMESTAMP
);
```

3. Опис таблиці “log_entries”

```

create table log_entries
(
    id          bigserial
               primary key,
    source_id   integer
               references public.log_sources,
    level_id    integer
               references public.log_levels,
    message     text                not null,
    timestamp   timestamp with time zone not null,
    context     jsonb,
    created_at  timestamp with time zone default CURRENT_TIMESTAMP
);

```

4. Опис таблиці “log_levels”

```

create table log_levels
(
    id          serial
               primary key,
    level       varchar(50) not null
               unique
);

```

5. Опис таблиці “log_sources”

```

create table log_sources
(
    id          serial
               primary key,
    name        varchar(255) not null
               unique,
    description text
);

```

```
);
```

6. Опис таблиці “report_executions”

```
create table report_executions
```

```
(
```

```
  id          serial
```

```
    primary key,
```

```
  report_id   integer
```

```
    references public.reports
```

```
    on delete cascade,
```

```
  status      varchar(50)          default 'pending'::character varying not null,
```

```
  file_path   text,
```

```
  file_size   bigint,
```

```
  execution_time integer,
```

```
  error_message text,
```

```
  parameters  jsonb,
```

```
  executed_at timestamp with time zone default CURRENT_TIMESTAMP,
```

```
  completed_at timestamp with time zone
```

```
);
```

7. Опис таблиці “reports”

```
create table reports
```

```
(
```

```
  id          serial
```

```
    primary key,
```

```
  name        varchar(255)          not null,
```

```
  description text,
```

```
  report_type varchar(50)          not null,
```

```
  config      jsonb                not null,
```

```
  schedule    jsonb,
```

```
  format      varchar(20)          default 'pdf'::character varying not null,
```

```
recipients text[],
created_by varchar(255) not null,
is_active boolean default true,
created_at timestamp with time zone default CURRENT_TIMESTAMP,
updated_at timestamp with time zone default CURRENT_TIMESTAMP
);
```