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

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна система управління та автоматизації IT-інфраструктури сучасного офісу»

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

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

_____ Єлізавета ПАРХОТЬКО
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувачка вищої освіти
групи ІСД-43 Єлізавета ПАРХОТЬКО
(ім'я,ПРІЗВИЩЕ

Керівник: Дмитро КОЗЛОВ, PhD
науковий ступінь *(ім'я,ПРІЗВИЩЕ*
вчене звання

Рецензент: _____
науковий ступінь *(ім'я,ПРІЗВИЩЕ*
вчене звання

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технологіЇ
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК
“____” _____2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Пархотько Єлізавета Олександрівна
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Інформаційна система управління та автоматизації ІТ-інфраструктури сучасного офісу»
керівник кваліфікаційної роботи: Дмитро КОЗЛОВ, PhD.
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “20” лютого 2026 р. №51
2. Строк подання кваліфікаційної роботи «01» червня 2026 р.
3. Вихідні дані кваліфікаційної роботи:

1. Науково технічна література.
2. Аналіз моніторингу та управління.
3. Розумні системи для моніторингу та управління інфраструктурою.

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

1. Огляд проблематики управління та автоматизації IT-інфраструктури сучасного офісу.
2. Архітектура та модель даних інформаційної системи централізованого управління IT-ресурсами офісу.
3. Програмна реалізація інформаційної системи управління IT-інфраструктурою офісу.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання « 20» лютого 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури		
2	Дослідження технічних аспектів та ефективності існуючих рішень управління IT-інфраструктурою		
3	Вибір технологічного стеку, постановка технічного завдання на розробку		
4	Проєктування структури, архітектури та користувацького інтерфейсу		
5	Програмна реалізація та тестування роботи системи		
6	Розробка демонстраційних матеріалів		
7	Попередній захист дипломної роботи		
8	Подання роботи в деканат		

Здобувачка вищої освіти _____ Єлізавета ПАРХОТЬКО

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Дмитро КОЗЛОВ

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 63 стор., 3 табл., 15 рис., 20 джерел.

Мета роботи - розробка інформаційної системи управління та автоматизації IT-інфраструктури сучасного офісу, яка дозволить централізувати облік апаратних і програмних ресурсів, забезпечити моніторинг стану вузлів у реальному часі, автоматизувати рутинні адміністративні задачі та скоротити час реагування на технічні інциденти.

Об'єкт дослідження - процеси адміністрування та автоматизації IT-інфраструктури сучасного офісу, що включає мережеве обладнання, робочі станції, серверні потужності, програмне забезпечення та хмарні сервіси.

Предмет дослідження - методи та засоби централізованого управління IT-ресурсами, моніторингу їхнього стану, обробки інцидентів і автоматизації рутинних адміністративних операцій з використанням сучасного программного забезпечення та архітектурних підходів на основі REST API.

Короткий зміст роботи: У першому розділі виконано аналіз IT-інфраструктури сучасного офісу, розглянуто сучасні підходи до автоматизації адміністрування, проведено порівняльне дослідження існуючих програмних рішень (Zabbix, GLPI, ManageEngine, Microsoft Endpoint Manager, Ansible та інших), сформульовано функціональні та нефункціональні вимоги до власної системи. У другому розділі обґрунтовано вибір технологічного стеку, спроєктовано модульну архітектуру системи, розроблено модель бази даних, а також визначено принципи взаємодії компонентів через REST API. Окрему увагу приділено проєктуванню підсистем інвентаризації, моніторингу, управління конфігураціями, обробки інцидентів і формування аналітичної звітності. Третій розділ присвячено програмній реалізації основних модулів системи, налаштуванню автоматизованих сценаріїв обслуговування, інтеграції каналів сповіщення та проведенню функціонального

тестування з аналізом отриманих результатів і визначенням перспектив подальшого розвитку.

КЛЮЧОВІ СЛОВА: інформаційна система, автоматизація, іт-інфраструктура, офіс, адміністрування, моніторинг, інвентаризація, управління конфігураціями, управління інцидентами, REST API, база даних.

ЗМІСТ

ВСТУП

Актуальність теми. У XXI столітті інформаційні технології перетворилися з допоміжного інструменту на ключовий ресурс будь-якої організації. Сучасний офіс - це вже не просто сукупність робочих місць, а складна екосистема, що поєднує мережеве обладнання, серверні потужності, віртуалізовані середовища, хмарні сервіси, спеціалізоване програмне забезпечення, периферійні пристрої та особисті гаджети співробітників у межах концепції BYOD. Кожен із цих компонентів вимагає постійного нагляду, оновлення, налаштування й контролю безпеки. Безперебійна робота офісу безпосередньо залежить від злагодженого функціонування цієї інфраструктури, а будь-який збій миттєво перетворюється на простій бізнес-процесів і фінансові втрати.

Водночас традиційні підходи до адміністрування IT-інфраструктури, побудовані на ручному виконанні рутинних операцій, демонструють низку суттєвих недоліків. До них належать висока трудомісткість обслуговування, ймовірність людських помилок під час налаштування пристроїв, затримки у виявленні та усуненні інцидентів, неконтрольоване зростання витрат на персонал IT-підтримки, а також відсутність єдиного джерела достовірної інформації про стан ресурсів. Використання великої кількості розрізнених інструментів - окремих систем для моніторингу, інвентаризації, керування конфігураціями та обробки заявок користувачів - лише ускладнює роботу адміністратора й знижує оперативність прийняття управлінських рішень.

Сучасні тенденції цифрової трансформації, переходу на гібридні моделі роботи, активного використання хмарних сервісів і зростання вимог до кібербезпеки висувають нові вимоги до систем управління IT-інфраструктурою. Підприємства потребують централізованих платформ, що забезпечують комплексний моніторинг стану ресурсів у режимі реального часу, автоматизацію типових задач (оновлення програмного забезпечення, резервне копіювання, відновлення після збоїв), уніфікований облік активів і ліцензій, а також аналітичні інструменти для підтримки

управлінських рішень. Особливої актуальності таким рішенням додає прагнення компаній до зниження операційних витрат і підвищення ефективності IT-відділів - як стратегічного фактора конкурентоспроможності.

Розробка інформаційної системи управління та автоматизації IT-інфраструктури офісу дозволяє об'єднати ключові адміністративні функції в єдиному програмному середовищі, забезпечити прозорість процесів, мінімізувати вплив людського фактору та значно скоротити час реагування на технічні інциденти. Запровадження таких систем сприяє підвищенню надійності, керованості та економічної ефективності IT-середовища, що робить обрану тему дипломної роботи актуальною як з наукової, так і з практичної точки зору. [18]

Мета дослідження - розробка інформаційної системи управління та автоматизації IT-інфраструктури сучасного офісу, яка дозволить централізувати облік апаратних і програмних ресурсів, забезпечити моніторинг стану вузлів у реальному часі, автоматизувати рутинні адміністративні задачі та скоротити час реагування на технічні інциденти.

Об'єкт дослідження - процеси адміністрування та автоматизації IT-інфраструктури сучасного офісу, що включає мережеве обладнання, робочі станції, серверні потужності, програмне забезпечення та хмарні сервіси.

Предмет дослідження - методи та засоби централізованого управління IT-ресурсами, моніторингу їхнього стану, обробки інцидентів і автоматизації рутинних адміністративних операцій з використанням сучасного програмного забезпечення та архітектурних підходів на основі REST API.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

1. Проаналізувати теоретичні засади побудови IT-інфраструктури сучасного офісу та сучасні підходи до автоматизації її адміністрування.

2. Провести порівняльний аналіз існуючих програмних рішень для управління IT-інфраструктурою (Zabbix, GLPI, ManageEngine, Microsoft Endpoint Manager, Ansible та інших), виявити їхні переваги та недоліки.
3. Сформулювати функціональні та нефункціональні вимоги до власної інформаційної системи.
4. Обґрунтувати вибір технологічного стеку та спроєктувати модульну архітектуру системи.
5. Розробити модель бази даних і визначити принципи взаємодії компонентів через REST API.
6. Реалізувати ключові модулі системи: інвентаризації, моніторингу, управління конфігураціями, обробки інцидентів та аналітичної звітності.
7. Провести функціональне тестування системи й оцінити ефективність запропонованого рішення.

Методи дослідження. У процесі виконання роботи застосовано низку загальнонаукових та спеціальних методів, що відповідають поставленим завданням:

- *аналіз літературних джерел та наукових публікацій* - для вивчення теоретичних основ управління IT-інфраструктурою та сучасних підходів до її автоматизації;
- *порівняльний аналіз* - для оцінки існуючих програмних рішень і виявлення їхніх функціональних особливостей;
- *системний підхід* - для побудови архітектури системи та визначення взаємодії її модулів;
- *методи об'єктно-орієнтованого проєктування та моделювання баз даних* - для розробки структури системи;

- *експериментальне тестування* - для перевірки працездатності реалізованого програмного забезпечення.

Наукова новизна отриманих результатів полягає в інтеграції ключових функцій управління IT-інфраструктурою (інвентаризації, моніторингу, керування конфігураціями, обробки інцидентів і аналітики) в єдину модульну платформу на основі сучасних відкритих технологій та архітектури REST API, орієнтовану на специфіку офісного середовища малих і середніх підприємств.[20]

Практична значущість дослідження полягає в тому, що розроблена інформаційна система може бути впроваджена в офісах підприємств різного масштабу для централізації адміністративних процесів, скорочення витрат на IT-обслуговування, підвищення прозорості обліку активів і пришвидшення реагування на технічні інциденти. Запропоноване рішення є гнучким, масштабованим і не потребує значних інвестицій у комерційні платформи.

Апробація результатів дослідження. Основні положення та результати, викладені в кваліфікаційній роботі, апробовано шляхом участі у III Міжнародній науково-практичній конференції «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії» (м. Київ, ДУІКТ, 2025 р.) у вигляді тез доповіді на тему «Інформаційна система управління та автоматизації IT-інфраструктури сучасного офісу».

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань і демонстраційних матеріалів. Загальний обсяг текстової частини становить 65 сторінок, робота містить 20 рисунків, 3 таблиці та 20 використаних джерел.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ УПРАВЛІННЯ ІТ-ІНФРАСТРУКТУРОЮ ОФІСУ

1.1 ІТ-інфраструктура сучасного офісу: склад, особливості, виклики

В умовах динамічного розвитку інформаційного суспільства поняття «ІТ-інфраструктура» вийшло далеко за межі суто технічного визначення й набуло стратегічного значення для функціонування організацій будь-якого масштабу. Під ІТ-інфраструктурою сучасного офісу розуміють організовану сукупність апаратних, програмних, мережевих та організаційних компонентів, які у взаємодії забезпечують виконання інформаційних процесів - збір, обробку, зберігання, передачу та представлення даних - у межах операційної діяльності підприємства. Якість і надійність цієї інфраструктури безпосередньо впливають на продуктивність співробітників, безперервність бізнес-процесів і конкурентоспроможність компанії на ринку.[18]

Структурно ІТ-інфраструктура сучасного офісу є багаторівневою системою, у складі якої прийнято виділяти кілька функціонально пов’язаних шарів. Перший рівень - *апаратна складова*, що охоплює серверне обладнання (фізичні та віртуальні сервери, системи зберігання даних), користувацькі робочі станції (стаціонарні комп’ютери, ноутбуки, моноблоки), мобільні пристрої співробітників, периферійне обладнання (принтери, сканери, мультифункціональні пристрої, конференц-камери), а також активне і пасивне мережеве обладнання - комутатори, маршрутизатори, точки бездротового доступу, мережеві шлюзи. Другий рівень утворює *системне та прикладне програмне забезпечення*: операційні системи серверів і робочих станцій, системи віртуалізації та контейнеризації, корпоративні бізнес-додатки, поштові й комунікаційні платформи, антивірусні рішення, системи резервного копіювання, бази даних. Третій рівень - *мережева інфраструктура*, що забезпечує внутрішню та зовнішню комунікацію вузлів, доступ до Інтернету, віддалене підключення співробітників через VPN, а також належний рівень сегментації та контролю трафіку.

Четвертий рівень становлять *хмарні сервіси* - публічні, приватні та гібридні, які на сьогодні є невід'ємною частиною робочого середовища (зокрема SaaS-рішення на кшталт Microsoft 365, Google Workspace, Slack, IaaS-платформи AWS, Azure, GCP та спеціалізовані PaaS-платформи). П'ятим рівнем виступає *організаційно-документальна складова*: політики безпеки, регламенти доступу, процедури інцидент-менеджменту, документація з конфігурацій, реєстри активів і ліцензій. Усі ці рівні тісно пов'язані між собою та формують єдине цифрове середовище офісу. Узагальнено описану структуру наведено на рис. 1.1.

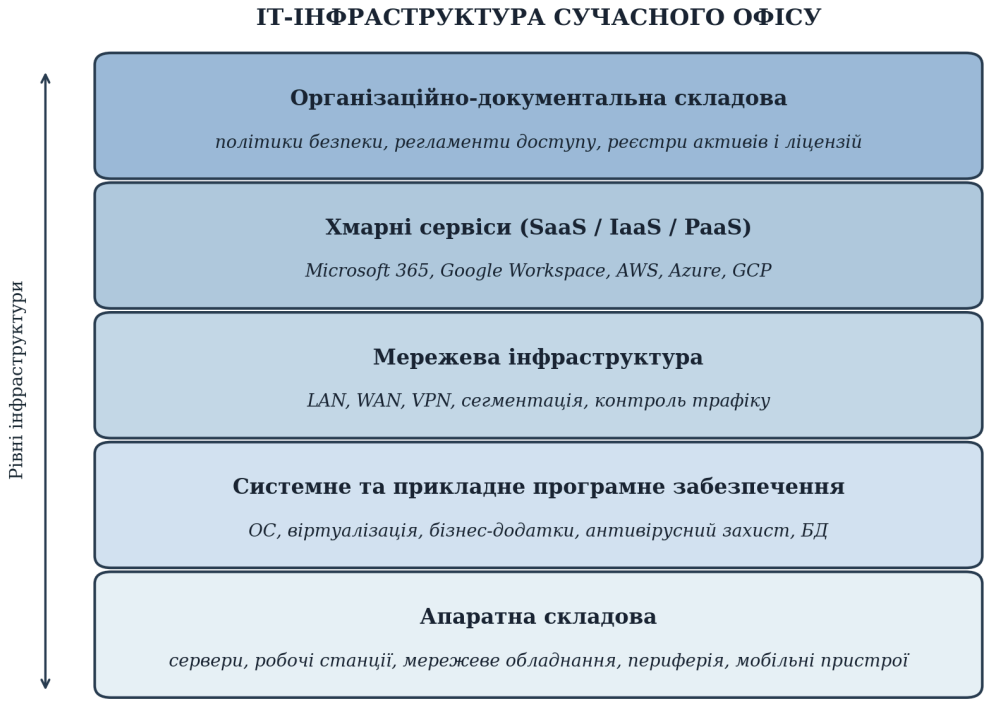


Рис. 1.1 Багаторівнева структура ІТ-інфраструктури сучасного офісу

Особливості сучасного офісного середовища принципово відрізняються від тих, що були характерними для офісу попереднього десятиліття. Перш за все, спостерігається перехід від традиційної моделі стаціонарної роботи до гібридного й розподіленого формату, коли частина співробітників працює дистанційно, з різних

географічних локацій і пристроїв. Це зумовило необхідність забезпечення безпечного віддаленого доступу до корпоративних ресурсів, розвитку технологій zero-trust та посилення механізмів автентифікації. По-друге, набула масового поширення концепція BYOD (Bring Your Own Device), яка передбачає використання співробітниками особистих смартфонів, планшетів і ноутбуків для виконання робочих обов'язків. Така практика, з одного боку, підвищує гнучкість роботи, з іншого - суттєво ускладнює задачу контролю над цифровими активами та забезпечення їх безпеки. По-третє, відбулася масова міграція корпоративних сервісів у хмару: значна частина бізнес-процесів сьогодні підтримується SaaS-платформами, що зменшує навантаження на локальну інфраструктуру, але водночас формує залежність від зовнішніх постачальників послуг, якості Інтернет-з'єднання і політик безпеки хмарних провайдерів. По-четверте, постійно зростає обсяг даних, які генеруються та обробляються в офісному середовищі, а разом з ним підвищуються вимоги до пропускної здатності мережі, продуктивності серверів і ємності систем зберігання. По-п'яте, кардинально змінилися й вимоги до кібербезпеки: фішингові атаки, програми-вимагачі, експлуатація вразливостей у віддалених підключеннях стали повсякденною загрозою, що вимагає від IT-відділів постійного моніторингу та оперативного реагування.[17]

Перелічені особливості зумовлюють низку системних викликів, з якими стикаються адміністратори при організації управління IT-інфраструктурою сучасного офісу. До найбільш суттєвих з них належать наступні.

Висока гетерогенність середовища. Сучасний офіс об'єднує обладнання різних виробників, операційні системи різних поколінь, програмні продукти від багатьох вендорів і хмарні сервіси з різними API. Це ускладнює уніфікацію підходів до адміністрування та змушує IT-фахівців опановувати велику кількість окремих інструментів.

Фрагментованість інструментів управління. На практиці адміністратори часто використовують десятки розрізнених систем - окремо для моніторингу, окремо

для інвентаризації, окремо для оновлення ПЗ, окремо для обробки заявок користувачів. Відсутність єдиної точки доступу до інформації про стан інфраструктури знижує оперативність прийняття рішень і збільшує ризик пропустити критичну подію.

Складність обліку активів і ліцензій. Динамічна зміна складу обладнання, постійні переміщення техніки між співробітниками, ротація ноутбуків і мобільних пристроїв, виведення з експлуатації застарілих машин - усі ці процеси без автоматизованого реєстру активів швидко перетворюються на хаос, що загрожує як фінансовими витратами на дублювання закупівель, так і юридичними ризиками через порушення ліцензійних угод.

Затримка реагування на інциденти. У відсутність централізованої системи моніторингу та оповіщення про інциденти ІТ-відділ нерідко дізнається про збій лише після звернення користувача, що значно збільшує час простою сервісу. Кожна додаткова хвилина простою - це прямі фінансові втрати для бізнесу та зниження довіри до ІТ-служби.

Безпека та відповідність нормативним вимогам. Сучасні підприємства мають дотримуватися численних регламентів - від внутрішніх політик до законодавчих актів (GDPR, ISO/IEC 27001, локальні норми захисту персональних даних). Без централізованих засобів конфігураційного контролю і журналювання забезпечити прозорий аудит інфраструктури практично неможливо.[7]

Зростання операційних витрат. Ручне виконання повторюваних задач - встановлення оновлень, перевстановлення ОС, налаштування нових робочих місць, резервне копіювання - потребує значних людино-годин і відволікає кваліфікованих ІТ-фахівців від стратегічних задач. Як наслідок, підприємства витрачають істотні бюджети на підтримку інфраструктури замість її розвитку.

Відсутність прозорої аналітики. Керівництво компаній дедалі частіше потребує об’єктивних даних про стан ІТ-середовища: завантаженість ресурсів, динаміку інцидентів, ефективність роботи ІТ-відділу. Без інтегрованих аналітичних

інструментів отримати такі дані вчасно та в зручному форматі складно, а отже - складно й ухвалювати обґрунтовані інвестиційні рішення.

Таким чином, IT-інфраструктура сучасного офісу являє собою складну, багатокомпонентну й динамічну систему, ефективне функціонування якої критично залежить від якості адміністративних процесів. Описані виклики формують об'єктивну потребу у створенні централізованих інформаційних систем, що дозволяють об'єднати в єдиному програмному середовищі функції моніторингу, інвентаризації, управління конфігураціями, обробки інцидентів і аналітичної звітності. Саме такий підхід є основою для подальшого аналізу існуючих рішень і обґрунтування власної концепції системи, що буде представлено в наступних підрозділах.

1.2 Сучасні підходи до автоматизації адміністрування IT

Розвиток корпоративних інформаційних технологій протягом останніх двох десятиліть пройшов шлях від ручного, реактивного адміністрування до проактивного, частково або повністю автоматизованого управління інфраструктурою. Цей перехід зумовлений як технологічними чинниками - стрімким зростанням обсягів даних, поширенням віртуалізації та хмарних сервісів, появою концепції контейнеризації, так і організаційними - підвищенням вимог до швидкості реагування, прозорості процесів і відповідності нормативним стандартам. Сьогодні автоматизація адміністрування є невід'ємною характеристикою зрілої IT-функції підприємства, а її ефективне впровадження безпосередньо впливає на продуктивність бізнесу.

Концептуальною основою сучасної автоматизації виступає методологія *ITSM (IT Service Management)* - комплексний підхід до управління IT-послугами, орієнтований на потреби бізнесу та користувачів. Найвідомішим стандартом у межах ITSM є бібліотека рекомендацій *ITIL (Information Technology Infrastructure Library)*, яка узагальнює найкращі світові практики у формі чітко структурованих процесів:

управління інцидентами, проблемами, змінами, конфігураціями, рівнями обслуговування, активами та подіями. ITIL не нав’язує конкретних технологій, а формує термінологічну та методологічну рамку, у межах якої будь-яка автоматизована система управління IT-інфраструктурою набуває логічної завершеності. Більшість сучасних програмних продуктів - від ServiceNow і ManageEngine ServiceDesk Plus до GLPI та Jira Service Management - реалізовані саме у відповідності до принципів ITIL, що забезпечує сумісність процесів і узгодженість підходів між різними організаціями.[8]

Окремим напрямом, що визначив обличчя сучасного адміністрування, є концепція *Infrastructure as Code (IaC)*. Її суть полягає у заміні ручної конфігурації обладнання й сервісів декларативним або імперативним описом цільового стану інфраструктури у вигляді коду, який зберігається у системах контролю версій (Git) і виконується відповідними інструментами автоматизації - Ansible, Puppet, Chef, SaltStack, Terraform. Такий підхід дозволяє відтворювати середовища з абсолютною точністю, мінімізувати «сніжинковий» ефект ручних налаштувань, реалізовувати ідемпотентні операції (тобто такі, що приводять систему до однакового стану незалежно від кількості повторень), а також документувати інфраструктуру самим фактом її конфігураційного коду. У контексті офісного середовища IaC найчастіше застосовується для централізованого розгортання робочих станцій, оновлення конфігурацій мережевих пристроїв і автоматичної підготовки серверів.[6]

Невід’ємним елементом будь-якої автоматизованої системи адміністрування є *база даних управління конфігураціями (Configuration Management Database, CMDB)* - централізоване сховище інформації про всі IT-активи організації та зв’язки між ними. CMDB зберігає відомості про обладнання, програмне забезпечення, ліцензії, мережеві з’єднання, відповідальних осіб, життєвий цикл активів і їхній поточний стан. Завдяки цьому забезпечується «єдина версія правди» - несуперечливе джерело інформації, з яким працюють усі підсистеми: моніторинг, інцидент-менеджмент, планування закупівель, аудит безпеки. Сучасні CMDB-системи інтегруються з

інструментами автоматичного виявлення вузлів (network discovery), що дозволяє підтримувати реєстр активів в актуальному стані без значних ручних зусиль.

Потужним рушієм автоматизації стала також концепція *проактивного моніторингу*, що замінила застарілий реактивний підхід «вирішувати проблему після її виявлення користувачем». Сучасні моніторингові платформи - Zabbix, Prometheus, Nagios, Grafana, Datadog, Dynatrace - здійснюють безперервний збір метрик з усіх компонентів інфраструктури, виявляють відхилення від нормальних показників, формують прогнози навантаження та автоматично генерують сповіщення про потенційні інциденти. Тісно пов’язаною з моніторингом є категорія *RMM-платформ (Remote Monitoring and Management)*, які об’єднують функції віддаленого спостереження, дистанційного керування робочими станціями, автоматичного оновлення програмного забезпечення та антивірусного захисту в межах єдиного інтерфейсу - типовими представниками цього класу є NinjaOne, Atera, ConnectWise Automate, N-able N-central.[17]

Найновішим витком еволюції є концепція *AIOps (Artificial Intelligence for IT Operations)*, що передбачає застосування методів машинного навчання та аналітики великих даних для автоматизації прийняття рішень в управлінні IT. Алгоритми AIOps здатні виявляти аномалії у потоках телеметрії, кореляційно пов’язувати події з різних джерел, передбачати збої до їх виникнення та автоматично запускати сценарії реагування - без участі адміністратора. Хоча AIOps поки що активніше використовується у великих ентерпрайз-середовищах і дата-центрах, окремі його елементи (наприклад, прогнозування завантаженості диска чи виявлення нетипових патернів мережевого трафіку) поступово впроваджуються й у рішення для офісних інфраструктур.

Важливим організаційним аспектом є вплив *DevOps-культури* на адміністрування IT. DevOps руйнує традиційний поділ між розробниками та операційними інженерами, пропагуючи принципи безперервної інтеграції, безперервного постачання, культуру спільної відповідальності та автоматизації на

всіх етапах життєвого циклу. Хоча первинно DevOps орієнтувався на середовища розробки програмного забезпечення, його практики - використання pipeline-ів автоматизації, контейнеризація через Docker і Kubernetes, моніторинг як обов’язкова складова - сьогодні активно адаптуються й для адміністрування корпоративних IT-середовищ.[10]

Узагальнено еволюцію підходів до адміністрування IT можна представити у вигляді хронологічної послідовності, поданої на рис. 1.2.

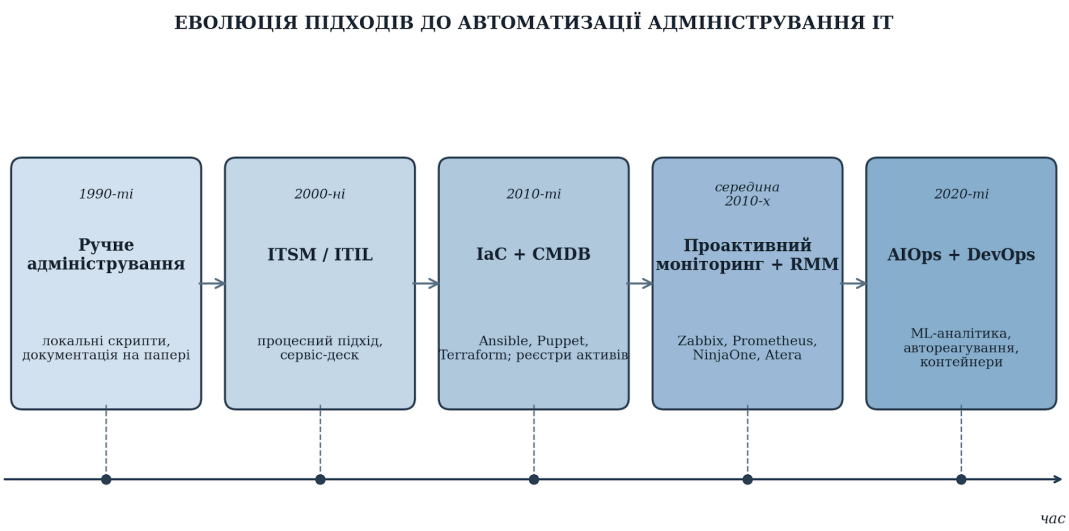


Рис. 1.2 Еволюція підходів до автоматизації адміністрування IT-інфраструктури

Аналіз представлених підходів дозволяє зробити висновок, що сучасна автоматизація адміністрування IT-інфраструктури офісу не зводиться до використання окремих інструментів - вона потребує системної інтеграції методологічних, технологічних і організаційних компонентів. Найбільш ефективні рішення поєднують у собі риси кількох концепцій одночасно: працюють відповідно до принципів ITIL, використовують підхід IaC для конфігурування, спираються на актуальну CMDB, забезпечують безперервний моніторинг і поступово впроваджують

елементи AIOps. Саме така комплексна модель є орієнтиром для розробки власної інформаційної системи в межах цієї дипломної роботи.

1.3 Огляд та порівняльний аналіз існуючих програмних рішень

Перш ніж приступити до розробки власної інформаційної системи управління IT-інфраструктурою, доцільно провести аналіз існуючих програмних продуктів, які тією чи іншою мірою вирішують подібні задачі. На сучасному ринку представлено десятки рішень - від комерційних ентерпрайз-платформ до відкритих проєктів спільнотного розвитку. Усі вони відрізняються набором функцій, цільовою аудиторією, моделлю ліцензування, складністю впровадження та глибиною інтеграції з інфраструктурою. Для систематизації розгляду доцільно поділити їх на чотири функціональні групи: системи моніторингу, ITSM/інвентаризаційні платформи, рішення для управління кінцевими точками (Endpoint Management) та інструменти управління конфігураціями.

Системи моніторингу інфраструктури. До найбільш поширених рішень цієї категорії належать Zabbix, Nagios, Prometheus у поєднанні з Grafana, а також комерційні платформи Datadog і Dynatrace. *Zabbix* є відкритою корпоративною платформою з агентно-серверною архітектурою, що підтримує збір метрик з обладнання й програмних сервісів через власні агенти, SNMP, JMX, IPMI, ICMP. Сильні сторони - велика кількість шаблонів, гнучка система тригерів, єдиний вебінтерфейс, підтримка розподіленого моніторингу через проксі-сервери. Слабкі - складність початкового налаштування, обмежена нативна функціональність для обліку активів і управління інцидентами на рівні ITSM. *Nagios Core* у класичному вигляді має простіше ядро, але повністю покладається на сторонні плагіни, що ускладнює розгортання та обмежує зручність користування. *Prometheus + Grafana* - це сучасний стек, орієнтований передусім на хмарні та контейнеризовані середовища, з чудовою візуалізацією, але без вбудованих модулів інвентаризації або інцидент-менеджменту. Усі моніторингові системи мають спільну особливість: вони

сильні в спостереженні за метриками, але майже не торкаються інших функцій адміністрування.

ITSM-системи та платформи інвентаризації. Цей клас включає рішення, що походять з традиції управління IT-послугами та активами. *GLPI (Gestionnaire Libre de Parc Informatique)* - відкрита платформа, що поєднує облік активів (комп'ютери, мережеве обладнання, ліцензії, картриджі), систему заявок (тікетування), управління договорами та фінансами. Завдяки агенту FusionInventory чи GLPI Agent забезпечується автоматичне виявлення обладнання та програмного забезпечення, встановленого на робочих станціях. Сильна сторона - комплексність функцій ITSM при безкоштовній моделі поширення. Обмеження - слабка вбудована аналітика, обмежений моніторинг продуктивності, складний інтерфейс для непідготовленого користувача. *ManageEngine ServiceDesk Plus / OpManager* - комерційна екосистема, що покриває весь спектр ITSM-функцій згідно з ITIL, інтегрується з мережевим моніторингом через окремий модуль OpManager. Перевагою є зрілість продукту, недоліком - висока вартість ліцензій та ускладнена локальна адаптація. *ServiceNow* - флагман ентерпрайз-сегменту з потужним функціоналом, AI-можливостями та глибокою кастомізацією, проте надмірно дорогий і складний для малих та середніх офісів.

Платформи управління кінцевими точками (Endpoint Management). Сюди належать рішення, орієнтовані передусім на адміністрування робочих станцій і мобільних пристроїв співробітників. *Microsoft Intune (Endpoint Manager)* - хмарна платформа, тісно інтегрована з Microsoft 365 і Azure AD, що дозволяє централізовано керувати політиками безпеки, розгортати програмне забезпечення, контролювати відповідність пристроїв корпоративним стандартам. Сильна сторона - глибока інтеграція з екосистемою Microsoft, слабка - обмежена ефективність у середовищах, де переважає не-Microsoft-обладнання, та залежність від хмарних сервісів. *NinjaOne* і *Atera* - представники класу RMM-платформ, що поєднують моніторинг робочих станцій, віддалене керування, патч-менеджмент і базову систему заявок. Привабливі

для малих компаній і MSP-провайдерів простотою впровадження, але обмежені у глибині кастомізації та коштують значні суми при масштабуванні на велику кількість пристроїв.

Системи управління конфігураціями та автоматизації. До цієї групи входять інструменти, розроблені для декларативного або імперативного опису стану інфраструктури з можливістю його автоматичного відтворення на множині вузлів. *Ansible* від Red Hat завоював популярність завдяки агентлес-моделі (керування через SSH без необхідності встановлювати агентів), читабельному синтаксису YAML і великій бібліотеці модулів. Він добре підходить для автоматизації типових операцій (встановлення ПЗ, налаштування сервісів, патчинг), але не є системою моніторингу чи інвентаризації - лише виконує сценарії, описані інженером. *Puppet* і *Chef* реалізують клієнт-серверну модель з агентами на цільових вузлах, що краще підходить для великих, постійних інфраструктур. *Terraform* спеціалізується на провіженінгу хмарної інфраструктури, що виходить за межі типових офісних задач. Усі ці інструменти потужні в межах своєї спеціалізації, але вимагають інтеграції з іншими продуктами для побудови повноцінної платформи адміністрування.

Для систематизації виявлених характеристик у табл. 1.1 наведено зведене порівняння розглянутих рішень за ключовими функціональними напрямами, які мають значення для офісного середовища.

Таблиця 1.1

Порівняльна характеристика існуючих програмних рішень для управління IT-інфраструктурою

Рішення	**Інвентаризація**	**Моніторинг**	**Управління конфігураціями**	**Інциденти**	**Аналітика**	**Ліцензування**
Zabbix	Часткова	Повна	Обмежена	Базова	Базова	Open Source

Рішення	Інвентаризація	Моніторинг	Управління конфігураціями	Інциденти	Аналітика	Ліцензування
Prometheus + Grafana	Відсутня	Повна	Відсутня	Через Alertmanager	Розширена	Open Source
GLPI	Повна	Обмежена	Часткова	Повна	Обмежена	Open Source
ManageEngine	Повна	Повна	Повна	Повна	Розширена	Комерційна
Microsoft Intune	Повна (MS)	Часткова	Повна	Базова	Базова	Підписка
NinjaOne / Atera	Повна	Повна	Часткова	Повна	Обмежена	Комерційна
Ansible	Відсутня	Відсутня	Повна	Відсутня	Відсутня	Open Source

Аналіз даних, поданих у табл. 1.1, дозволяє виявити характерну закономірність: жодне з розглянутих рішень не забезпечує одночасного повноцінного покриття всіх п’яти ключових функцій управління IT-інфраструктурою. Системи моніторингу глибоко працюють зі станом ресурсів, але слабкі в обліку активів та інцидент-менеджменті. ITSM-платформи відмінно ведуть реєстр активів і обробляють заявки, але мають обмежену аналітику продуктивності. Endpoint-рішення зосереджені на робочих станціях, проте не охоплюють серверну та мережеву інфраструктуру у повному обсязі. Інструменти управління конфігураціями виконують вузькоспеціалізовану задачу і не претендують на роль повноцінної

платформи. Для наочної демонстрації цих обмежень на рис. 1.3 представлено радарну діаграму функціонального покриття проаналізованих рішень.

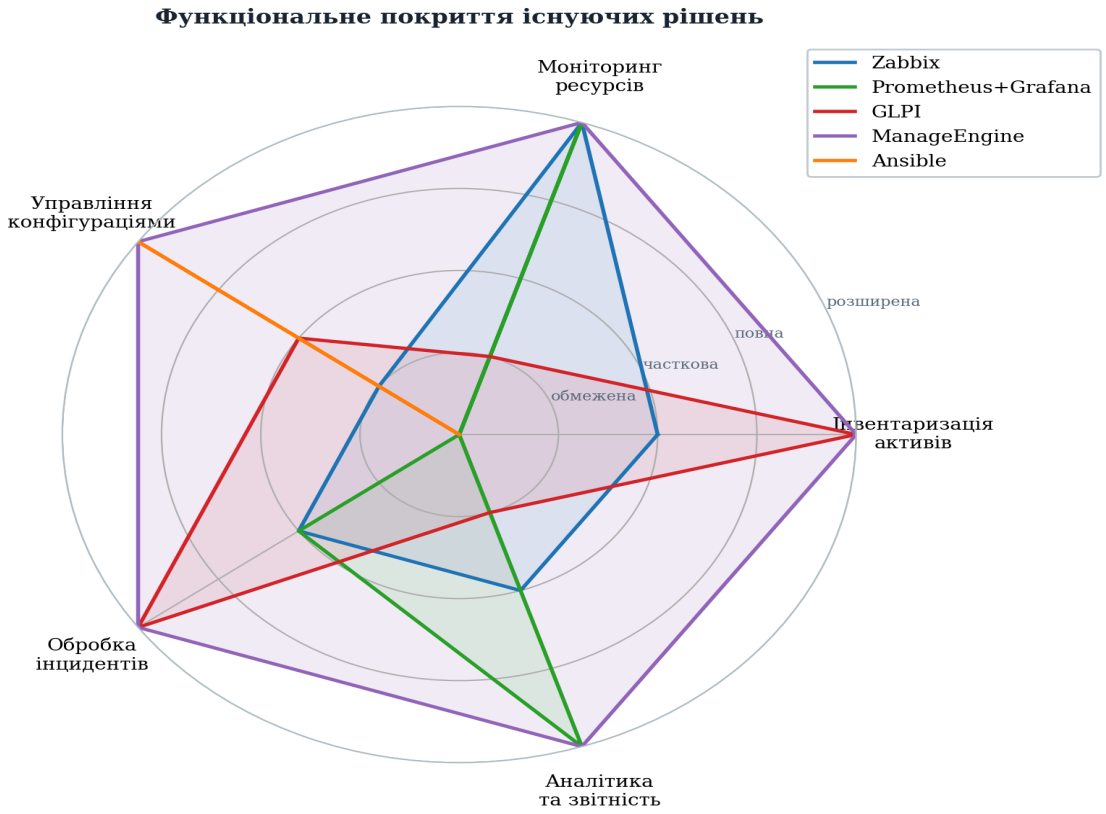


Рис. 1.3 Функціональне покриття існуючих рішень управління IT-інфраструктурою

Найбільш повне покриття серед розглянутих демонструють комерційні ентерпрайз-платформи (ManageEngine, ServiceNow), однак їх використання у малих і середніх офісах обмежене високою вартістю та надмірною складністю. Безкоштовні рішення, навпаки, привабливі за ціною, але вимагають інтеграції кількох продуктів між собою - і саме тут виникає проблема: кожна додаткова інтеграція збільшує складність обслуговування й ризик розузгодження даних між системами. Таким чином, обґрунтовується доцільність розробки власної інформаційної системи, яка б об’єднувала ключові функції адміністрування IT-інфраструктури в єдиному програмному середовищі, орієнтованому саме на потреби сучасного офісу. Така система має використовувати кращі підходи з кожної з розглянутих категорій,

водночас залишаючись простою у впровадженні, доступною за ресурсами та відкритою до подальшого розширення.

1.4 Постановка задачі та формулювання вимог до системи

На основі результатів аналізу теоретичних засад управління IT-інфраструктурою сучасного офісу, систематизації сучасних підходів до автоматизації адміністрування та критичного огляду існуючих програмних рішень можна сформулювати чітку постановку задачі для розроблюваної інформаційної системи. Метою її створення є побудова централізованої програмної платформи, що інтегрує в єдиному середовищі функції обліку активів, моніторингу ресурсів, управління конфігураціями, обробки інцидентів і формування аналітичної звітності - з орієнтацією на потреби офісних середовищ малого та середнього масштабу.

Розроблювана система має реалізовувати повний цикл адміністрування IT-інфраструктури: від автоматичного виявлення обладнання та програмного забезпечення, через безперервний моніторинг їхнього стану й автоматизоване виконання типових операцій обслуговування, до фіксації інцидентів, ескалації проблем і формування звітів для управлінських рішень. На відміну від спеціалізованих рішень, що покривають лише окремі аспекти, запропонована система повинна забезпечувати наскрізну прозорість процесів і узгодженість даних між усіма підсистемами.

Функціональні вимоги до системи визначаються переліком модулів і дій, які вона повинна виконувати. До них належать:

- *модуль інвентаризації та обліку активів* - автоматичне виявлення вузлів мережі, ведення реєстру апаратного й програмного забезпечення, контроль ліцензій, відстеження життєвого циклу активів від закупівлі до виведення з експлуатації;
- *модуль моніторингу* - збір метрик продуктивності та доступності серверів, робочих станцій, мережевого обладнання й критичних сервісів у режимі

реального часу, виявлення відхилень від нормальних показників та автоматичне формування подій;

- *модуль управління конфігураціями* - централізоване зберігання конфігурацій вузлів, версіонування змін, можливість відкату до попередніх версій, шаблонізоване розгортання налаштувань на групи пристроїв;
- *модуль автоматизації рутинних задач* - планове оновлення програмного забезпечення, резервне копіювання конфігурацій, автоматичне відновлення сервісів за заздалегідь визначеними сценаріями реагування;
- *модуль обробки інцидентів* - реєстрація заявок, класифікація за категоріями та пріоритетами, маршрутизація на відповідальних виконавців, відстеження життєвого циклу інциденту згідно з принципами ITIL;
- *модуль сповіщень* - інтеграція з зовнішніми каналами комунікації (електронна пошта, Slack, Telegram), гнучкі правила маршрутизації повідомлень залежно від типу та критичності події;
- *модуль аналітики та звітності* - формування зведених звітів про стан інфраструктури, динаміку інцидентів, ефективність обслуговування, а також надання даних для прийняття управлінських рішень;
- *модуль управління користувачами та правами доступу* - рольова модель доступу, аудит дій адміністраторів, автентифікація з можливістю інтеграції з корпоративними каталогами (LDAP, Active Directory).

Нефункціональні вимоги визначають якісні характеристики системи, що забезпечують її практичну придатність до експлуатації. Серед них:

- *масштабованість* - здатність системи зростати разом з інфраструктурою, що обслуговується, як вертикально (підвищенням ресурсів сервера), так і горизонтально (через розподілення компонентів між кількома вузлами);
- *надійність* - забезпечення безперервної роботи системи, мінімізація часу простою, наявність механізмів резервного копіювання й відновлення даних;

- *безпека* - захист від несанкціонованого доступу, шифрування каналів передачі даних, відповідність вимогам захисту персональних даних, журналювання дій;
- *продуктивність* - час відгуку інтерфейсу не повинен перевищувати кількох секунд при типовому навантаженні; обробка метрик і фонових задач - не блокувати основні функції;
- *зручність використання* - інтуїтивно зрозумілий вебінтерфейс, адаптивний дизайн для роботи з різних пристроїв, локалізація українською мовою;
- *модульність та розширюваність* - можливість додавання нових функцій без перепроєктування ядра, відкритий REST API для інтеграції зі сторонніми сервісами;
- *крос-платформність* - функціонування на типових операційних системах (Linux, Windows), сумісність з основними браузерами;
- *економічна ефективність* - побудова системи на основі відкритих технологій, відсутність ліцензійних відрахувань за використання базової платформи.[7]

Архітектурні принципи, покладені в основу проєктування системи, визначаються сучасними практиками побудови інформаційних систем. Перш за все, передбачається застосування *модульної архітектури* з чітким розподілом відповідальності між компонентами, що забезпечує можливість незалежної розробки, тестування та оновлення окремих модулів. По-друге, взаємодія між компонентами має реалізовуватись через *REST API* - стандартизований підхід, який дозволяє підключати до системи зовнішні клієнти (вебінтерфейс, мобільний додаток, скрипти автоматизації) без змін у серверній частині. [5] По-третє, для зберігання даних обирається *реляційна база даних* з нормалізованою структурою, що гарантує цілісність та узгодженість інформації про активи, конфігурації, події та інциденти. По-четверте, передбачається використання *контейнеризації* (Docker) для уніфікованого розгортання компонентів системи в різних середовищах.

Очікуваним результатом виконання поставленої задачі є створення повністю функціональної інформаційної системи, що дозволить IT-адміністраторам сучасного

офісу здійснювати централізоване управління інфраструктурою з єдиного інтерфейсу, скоротити витрати часу на рутинні операції, оперативно реагувати на технічні інциденти й отримувати об’єктивні дані для підтримки управлінських рішень. У наступних розділах будуть детально розкриті проєктні рішення (Розділ 2) та особливості програмної реалізації (Розділ 3) запропонованої системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ IT-ІНФРАСТРУКТУРОЮ ОФІСУ

2.1 Обґрунтування вибору технологічного стеку

Етап проєктування інформаційної системи починається з обґрунтованого вибору технологічного стеку - сукупності мов програмування, фреймворків, бібліотек і допоміжних інструментів, на яких базуватиметься розробка. Від цього вибору значною мірою залежать продуктивність, масштабованість, безпека, зручність супроводу та подальшого розвитку системи. У контексті розроблюваної платформи управління IT-інфраструктурою основними критеріями вибору стали: відкритість використовуваних технологій, наявність активної спільноти підтримки, сумісність із сучасними практиками контейнеризації та REST-архітектури, висока продуктивність у задачах асинхронної обробки великих обсягів метрик, а також доступність бібліотек для роботи з мережевим обладнанням, базами даних і зовнішніми API.

У ролі основної мови програмування серверної частини обрано *Python 3.12*. Цей вибір обґрунтований кількома факторами. По-перше, Python протягом останніх років стабільно посідає одне з провідних місць у рейтингах популярних мов програмування (TIOBE, Stack Overflow Developer Survey, RedMonk), що гарантує наявність кваліфікованих спеціалістів, регулярні оновлення та широку екосистему. По-друге, Python має лаконічний і читабельний синтаксис, що зменшує час розробки та полегшує супровід коду. По-третє, мова має одну з найбільших серед сучасних платформ бібліотечну базу, релевантну для задач адміністрування IT: модулі для роботи з SNMP (PySNMP), SSH (Paramiko, Fabric), REST API (httpx, requests), серіалізації (Pydantic), реляційних баз даних (SQLAlchemy), асинхронної обробки (asyncio, Celery). По-четверте, екосистема Python має зрілі фреймворки для побудови вебсервісів, серед яких FastAPI вирізняється поєднанням високої швидкодії, підтримки асинхронності та автоматичної генерації документації API. Альтернативи

(Node.js, Go, Java, C#) розглядалися, але були відхилені через нижчу продуктивність розробки (Java, C#), вужчу бібліотечну базу для адміністративних задач (Go) або менш зрілу типізацію (Node.js). Зведений аналіз порівняння наведено у табл. 2.1. [13]

Таблиця 2.1

Порівняння розглянутих мов програмування для серверної частини системи

Критерій	**Python (FastAPI)**	**Node.js (Express)**	**Go (Gin)**	**Java (Spring)**
Швидкість розробки	Висока	Висока	Середня	Низька
Продуктивність runtime	Висока (async)	Середня	Дуже висока	Висока
Бібліотеки для IT-адміністрування	Найширші	Середні	Обмежені	Широкі
Читабельність коду	Висока	Середня	Висока	Середня
Поріг входження для розробника	Низький	Середній	Високий	Високий
Підтримка асинхронності	Повна	Повна	Повна	Часткова
Документація та спільнота	Дуже активна	Активна	Активна	Активна

Як вебфреймворк серверної частини обрано *FastAPI*. Цей фреймворк побудовано на основі стандарту ASGI (Asynchronous Server Gateway Interface), що забезпечує неблокуючу обробку численних паралельних запитів - критичну вимогу для системи моніторингу, яка одночасно отримує дані від багатьох вузлів. FastAPI використовує бібліотеку Pydantic для строгої типізації та валідації вхідних/вихідних даних, автоматично генерує інтерактивну документацію API у форматі OpenAPI (Swagger UI та ReDoc), що суттєво полегшує тестування й інтеграцію. Простота синтаксису, висока швидкість роботи (співмірна з фреймворками на Go та Node.js за

результатами незалежних бенчмарків) та сумісність зі стандартними інструментами Python-екосистеми роблять FastAPI оптимальним вибором. [4, 12]

Для взаємодії з реляційною базою даних використано *SQLAlchemy 2.0* у поєднанні з *Alembic* для управління міграціями схеми. SQLAlchemy забезпечує об'єктно-реляційне відображення (ORM), що дозволяє описувати структуру таблиць у вигляді Python-класів і взаємодіяти з даними через об'єкти, уникаючи прямого написання SQL-запитів. Alembic, у свою чергу, дозволяє версіонувати зміни структури бази даних і безпечно застосовувати їх у різних середовищах (розробка, тестування, продакшн). Як основна СУБД обрано *PostgreSQL 16* - потужна реляційна база даних з відкритим кодом, що підтримує транзакційність (ACID), складні запити, повнотекстовий пошук, JSONB-поля для зберігання напівструктурованих даних та масштабування через реплікацію. Додатково для кешування та зберігання тимчасових даних (сесії, черги фонових задач) використано *Redis* - високопродуктивне сховище ключ-значення, що працює в оперативній пам'яті. [1, 11]

Для виконання фонових задач - періодичне опитування агентів моніторингу, надсилання сповіщень, генерація звітів, виконання сценаріїв автоматизації - застосовано брокер задач *Celery* у поєднанні з Redis як транспортом повідомлень. Така архітектура дозволяє винести довготривалі операції з основного запитного циклу, забезпечуючи швидкий відгук користувацького інтерфейсу. [2]

Клієнтська частина системи реалізована як *односторінковий вебзастосунок (SPA)* з використанням бібліотеки *React 18* і збірника *Vite*. React обрано через зрілість екосистеми, наявність великої кількості готових компонентних бібліотек (Material-UI, Ant Design), потужну підтримку інтерактивних інтерфейсів і реактивне оновлення даних, що особливо важливо для відображення метрик у режимі реального часу. Для побудови графіків та дашбордів використовується бібліотека *Recharts*, для роботи з REST API - *axios*. Стилізація виконується засобами *Tailwind CSS*, що забезпечує адаптивний дизайн і скорочує обсяг кастомних CSS-стилів. [9]

Для забезпечення безпеки реалізовано стандартні механізми: автентифікація через *JSON Web Token (JWT)* з підтримкою рефреш-токенів, шифрування паролів алгоритмом *bcrypt*, налаштування CORS-політик, перевірка прав доступу на рівні ендпоінтів API через залежності FastAPI. Усі чутливі дані передаються по HTTPS.

Для розгортання та супроводу системи передбачено використання *Docker* і *Docker Compose*. Кожен компонент системи (вебсервер FastAPI, фронтенд React, PostgreSQL, Redis, Celery-воркери, моніторинг-агенти) запускається в окремому контейнері з прозорою конфігурацією через змінні середовища. Це забезпечує відтворюваність розгортання, спрощує оновлення та дозволяє при потребі масштабувати окремі компоненти.[3]

Узагальнена структура технологічного стеку системи представлена на рис. 2.1.

Рис. 2.1 Технологічний стек інформаційної системи

Обраний технологічний стек відповідає сучасним практикам розробки корпоративних інформаційних систем, поєднує продуктивність із гнучкістю розробки, забезпечує відкритість і незалежність від конкретних комерційних постачальників. Це створює надійну основу для подальшого проєктування архітектури та компонентів системи.

2.2 Проєктування модульної архітектури інформаційної системи

Архітектура інформаційної системи управління IT-інфраструктурою офісу спроєктована за принципами модульності, шаруватості та чіткого розподілу відповідальності між компонентами. В основу покладено класичну тришарову архітектуру, що передбачає розділення системи на три логічно ізольовані рівні: рівень представлення (Presentation Layer), рівень бізнес-логіки (Business Logic Layer) та рівень доступу до даних (Data Access Layer). Такий підхід відповідає принципам Clean Architecture і дозволяє забезпечити масштабованість системи, спрощує тестування окремих компонентів та робить можливим заміну окремих рівнів без впливу на інші.[19]

Рівень представлення реалізований у вигляді односторінкового вебзастосунку (SPA) на основі React. Він відповідає за взаємодію з користувачем - адміністратором IT-інфраструктури - і не містить жодної бізнес-логіки. Усі дані для відображення (списки активів, графіки моніторингу, перелік інцидентів, звіти) отримуються з серверної частини через REST API. Інтерфейс адаптивний, підтримує темну й світлу теми, рольову візуалізацію (різні елементи інтерфейсу доступні різним групам користувачів) і реактивно оновлюється при надходженні нових даних. Для деяких сценаріїв (наприклад, потокове оновлення метрик у реальному часі) додатково використовується протокол *WebSocket*, що дозволяє серверу ініціативно надсилати оновлення без необхідності клієнтського опитування.

Рівень бізнес-логіки є серцем системи й реалізує всю предметну логіку: обробку запитів від користувача, валідацію вхідних даних, виконання бізнес-правил,

координацію взаємодії між модулями. Він побудований на основі фреймворку FastAPI, що організовує код у формі ендпоінтів REST API, згрупованих за функціональними областями (роутери). Бізнес-логіка винесена у сервісні класи (service layer), які не залежать від конкретної реалізації бази даних чи зовнішніх API - це досягається за рахунок використання шаблону «репозиторій» (Repository Pattern), що абстрагує доступ до даних. Така структура спрощує юніт-тестування й дозволяє в майбутньому замінити окремі компоненти (наприклад, перейти з PostgreSQL на іншу СУБД) без істотних змін у бізнес-логіці.

Рівень доступу до даних відповідає за збереження та витяг інформації з постійного сховища - реляційної бази даних PostgreSQL. Цей рівень реалізується через ORM SQLAlchemy: кожна сутність (пристрій, користувач, інцидент тощо) представлена окремою Python-моделлю, а CRUD-операції абстраговані у вигляді методів репозиторіїв. Для частих читальних операцій реалізовано кешування через Redis з налаштованими TTL-значеннями.

На додаток до основних шарів, у системі присутній *рівень фонових задач*, що виконується окремими воркерами Celery. Сюди винесено всі операції, які не повинні блокувати основний запитний цикл: періодичне опитування агентів моніторингу, виконання сценаріїв автоматизації (резервне копіювання, оновлення ПЗ), формування звітів, надсилання сповіщень через зовнішні канали. Координація фонових задач здійснюється через брокер повідомлень Redis, а їх планування - через Celery Beat (планувальник за крон-розкладом).

Для збору даних з вузлів інфраструктури передбачено два режими роботи: *агентний* (на робочих станціях, серверах і мережевих пристроях встановлюються легковагі агенти-збирачі, що передають метрики до основної системи) та *безагентний* (для пристроїв, де встановлення агента неможливе або небажане, використовуються стандартні протоколи SNMP, WMI, SSH). Обидва режими інтегруються в систему через однотипні endpoint-и API, що забезпечує уніфіковану обробку даних незалежно від способу їх збору.

Загальна архітектурна схема системи з усіма основними компонентами та потоками даних між ними представлена на рис. 2.2.

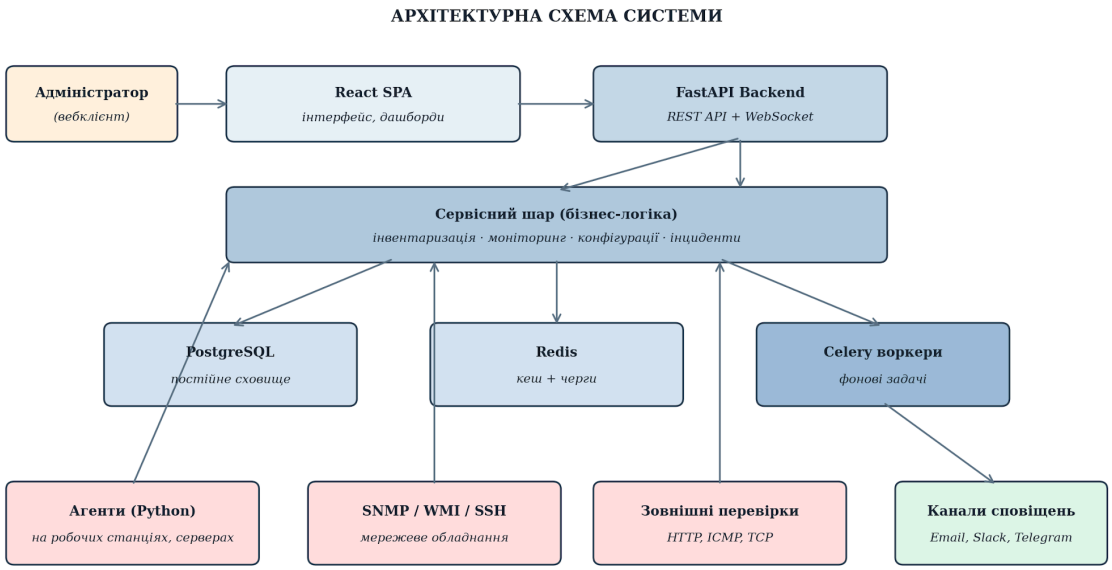


Рис. 2.2 Архітектурна схема інформаційної системи управління IT-інфраструктурою

Користувацька взаємодія з системою починається з вебінтерфейсу: запити надходять до FastAPI-сервера, який після автентифікації та авторизації передає їх у сервісний шар, що, у свою чергу, звертається до бази даних через ORM. Метрики, які надсилаються агентами або зібрані безагентним способом, потрапляють у спеціалізовані ендпоінти збору даних і зберігаються у PostgreSQL з паралельною агрегацією через фонові воркери. Сповіщення про критичні події формуються бізнес-логікою та передаються до модуля Notifications, який розсилає їх через підключені канали (email, Slack, Telegram). Така структура забезпечує чітке розділення відповідальності, високу пропускну здатність і простоту супроводу.

2.3 Моделювання структури бази даних

Центральним елементом інформаційної системи є реляційна база даних, що зберігає всю інформацію про стан IT-інфраструктури - від реєстру обладнання до журналів подій та історії інцидентів. Її структура спроєктована з дотриманням принципів нормалізації (до третьої нормальної форми) для усунення дублювання даних та забезпечення цілісності, з використанням первинних і зовнішніх ключів для встановлення зв'язків між сутностями, а також з продуманою індексацією критичних полів для оптимізації запитів.[11, 15]

Аналіз функціональних вимог до системи, сформульованих у підрозділі 1.4, дозволив виділити такі основні сутності предметної області:

- *Користувач (User)* - особа, що має доступ до системи (адміністратор, оператор, аудитор); містить дані для автентифікації, контактну інформацію, прив'язку до ролі.
- *Роль (Role)* - набір прав доступу до різних функцій системи; реалізує рольову модель безпеки (RBAC).
- *Пристрій (Device)* - будь-який актив IT-інфраструктури: сервер, робоча станція, мережевий пристрій, принтер, мобільний пристрій. Зберігає ідентифікаційні дані, технічні характеристики, статус, прив'язку до відповідального користувача.
- *Тип пристрою (DeviceType)* - категорія, до якої належить пристрій (наприклад, «робоча станція», «маршрутизатор», «сервер»).
- *Програмне забезпечення (Software)* - інформація про ПЗ, встановлене на пристроях, включно з версією, ліцензією та датою останнього оновлення.
- *Ліцензія (License)* - окрема сутність для обліку ліцензійних угод, кількості доступних місць та терміну дії.
- *Конфігурація (Configuration)* - поточний стан налаштувань пристрою; підтримує версіонування, що дозволяє повертатися до попередніх версій.

- *Версія конфігурації (ConfigurationVersion)* —-окремий запис кожної версії з міткою часу та автором змін.
- *Подія (Event)* - зафіксована системою подія моніторингу: відхилення метрики від норми, недоступність вузла, спрацювання правила тощо.
- *Метрика (Metric)* - числове значення показника, отримане від агента або через зовнішнє опитування; зберігається у вигляді часового ряду.
- *Інцидент (Incident)* - формалізований запис про порушення нормальної роботи інфраструктури; має статус, пріоритет, категорію, відповідального виконавця, історію змін.
- *Сповіщення (Notification)* - запис про надіслане користувачеві повідомлення з міткою каналу та статусом доставки.
- *Звіт (Report)* - згенерована аналітична вибірка за певний період, що може бути експортована у форматах PDF, Excel або CSV.
- *Журнал дій (AuditLog)* - фіксація всіх значимих дій користувачів у системі для забезпечення прозорості та аудиту безпеки.

Між сутностями встановлено такі основні зв’язки. Користувач прив’язується до ролі через зовнішній ключ (зв’язок «багато до одного»). Пристрій належить до певного типу та може мати призначеного відповідального користувача. Між пристроєм і програмним забезпеченням існує зв’язок «багато до багатьох», реалізований через проміжну таблицю DeviceSoftware. Конфігурація пов’язана з конкретним пристроєм, а її версії - зі своєю конфігурацією. Метрики прив’язані до пристрою та мають мітку часу, що дозволяє формувати часові ряди. Подія може ініціювати створення інциденту; інцидент, у свою чергу, містить історію статусів і дій. Сповіщення можуть породжуватися як подіями, так і інцидентами. Журнал дій незалежний - кожен запис фіксує користувача, тип дії та об’єкт впливу.

Загальна структура бази даних з основними сутностями та зв’язками представлена на рис. 2.3 у вигляді ER-діаграми.

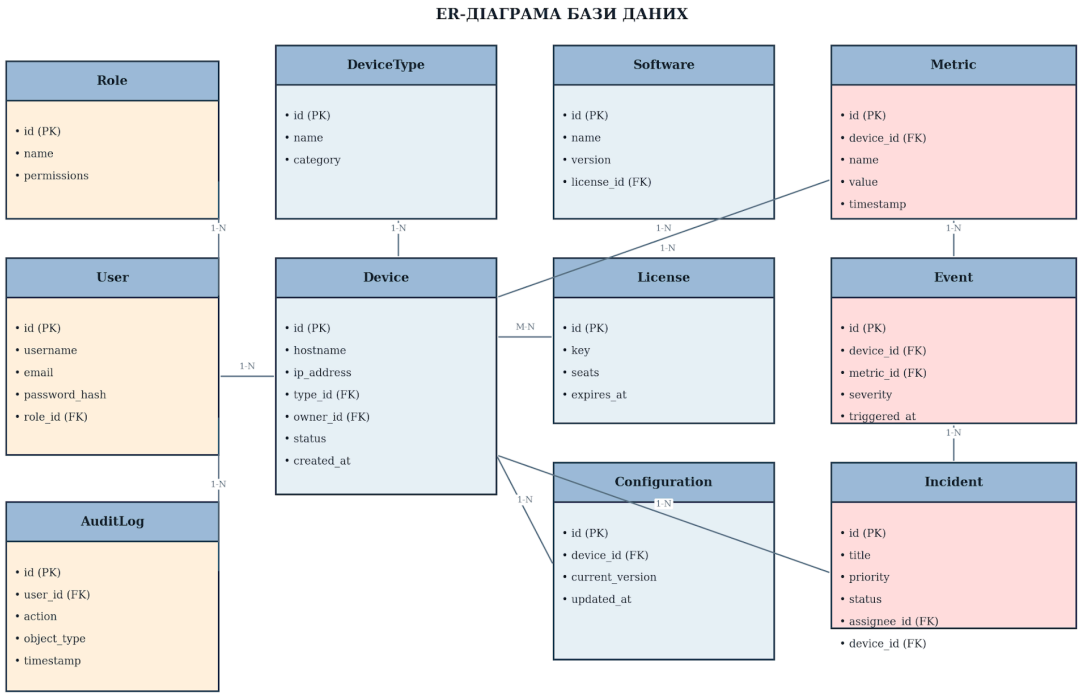


Рис. 2.3 ER-діаграма бази даних інформаційної системи

Для оптимізації роботи з великими обсягами даних, які генеруються модулем моніторингу, передбачено такі рішення: таблиця метрик партиціонується за датою (помісячні розділи), що дозволяє ефективно архівувати старі дані та підтримувати швидкість запитів незалежно від загального обсягу історії; критичні поля (ідентифікатори пристроїв, часові мітки, статуси інцидентів) проіндексовано B-tree індексами; для часто використовуваних агрегаційних запитів створено матеріалізовані подання, що періодично оновлюються через фонові задачі. Чутливі дані (паролі, токени) зберігаються виключно у хешованому вигляді з використанням bcrypt.

Така структура бази даних забезпечує цілісність інформації, високу швидкість доступу до критичних даних та можливість масштабування при зростанні обсягу інфраструктури, яку обслуговує система.

2.4 Проєктування підсистем інвентаризації, моніторингу та управління конфігураціями

Базові функціональні підсистеми, що формують ядро операційних можливостей розроблюваної інформаційної системи, - це підсистеми інвентаризації, моніторингу та управління конфігураціями. Вони відповідають за збір і підтримку в актуальному стані даних про IT-інфраструктуру, проактивне виявлення проблем та централізоване керування налаштуваннями пристроїв.

Підсистема інвентаризації та обліку активів реалізує функції автоматичного виявлення обладнання, формування реєстру активів і відстеження їхнього життєвого циклу. У її основі лежать два механізми збору даних: *активний агентний* і *пасивний безагентний*. Агентний механізм передбачає встановлення на робочих станціях і серверах легковагого Python-агента, що періодично (з налаштованою частотою - типово раз на годину або при кожному завантаженні системи) збирає інформацію про обладнання (модель CPU, обсяг пам’яті, конфігурацію дисків, мережеві адаптери), операційну систему (версія, патчі, налаштування) та встановлене програмне забезпечення (перелік пакетів, версії, дати оновлення). Зібрані дані передаються на сервер через захищене HTTPS-з’єднання у форматі JSON. Безагентний механізм застосовується для мережевого обладнання (комутатори, маршрутизатори, точки доступу) та принтерів - тут використовується протокол SNMP для опитування MIB-записів пристрою.

Окрім автоматичного збору, передбачено можливість *ручного введення* даних для активів, які з тих чи інших причин не можуть бути виявлені програмно (наприклад, кабельне господарство, ліцензійні угоди, обладнання поза мережею). Уся отримана інформація потрапляє у центральний реєстр активів і доступна для перегляду через вебінтерфейс із розширеними фільтрами, сортуванням і пошуком. Система автоматично виявляє розбіжності між поточним та попереднім станом активу (наприклад, заміну дискового накопичувача чи оновлення версії ПЗ) і фіксує такі зміни у журналі активу. Для відстеження життєвого циклу кожен актив має

статус: «у використанні», «на зберіганні», «у ремонті», «списано». Зміна статусу фіксується з прив’язкою до користувача та часової мітки.

Підсистема моніторингу забезпечує безперервне спостереження за станом ресурсів IT-інфраструктури. Архітектурно вона побудована за принципом збору метричних часових рядів: усі показники (завантаження процесора, обсяг вільної пам’яті, дисковий простір, мережевий трафік, доступність сервісів) фіксуються у вигляді пар «час – значення» з міткою джерела. Дані надходять трьома основними шляхами: від інвентаризаційних агентів (вони ж виконують функцію моніторингу), через SNMP-опитування мережевого обладнання та через зовнішні перевірки доступності (синтетичні запити HTTP, ICMP-пінг, TCP-з’єднання з визначеними портами). Для кожного типу метрики передбачено налаштування періодичності збору - від кожних 10 секунд для критичних показників до раз на годину для статистичних.

Аналіз отриманих даних відбувається у двох режимах: *миттєвому* - при кожному отриманні нового значення метрика порівнюється з визначеними пороговими умовами, і у разі їх порушення формується подія; та *періодичному* - фонові задачі агрегують метрики за фіксовані інтервали (хвилина, година, день), розраховують середні, мінімальні, максимальні значення і відсоткові показники, які потім використовуються в дашбордах і звітах. Підсистема підтримує гнучкі правила алертингу: користувач може визначити, що подія має спрацьовувати лише при стійкому перевищенні порогу протягом певного часу (наприклад, завантаження CPU понад 90% упродовж 5 хвилин), що зменшує кількість хибних спрацювань.

Архітектура потоку даних від джерел метрик до бізнес-логіки та сховища представлена на рис. 2.4.

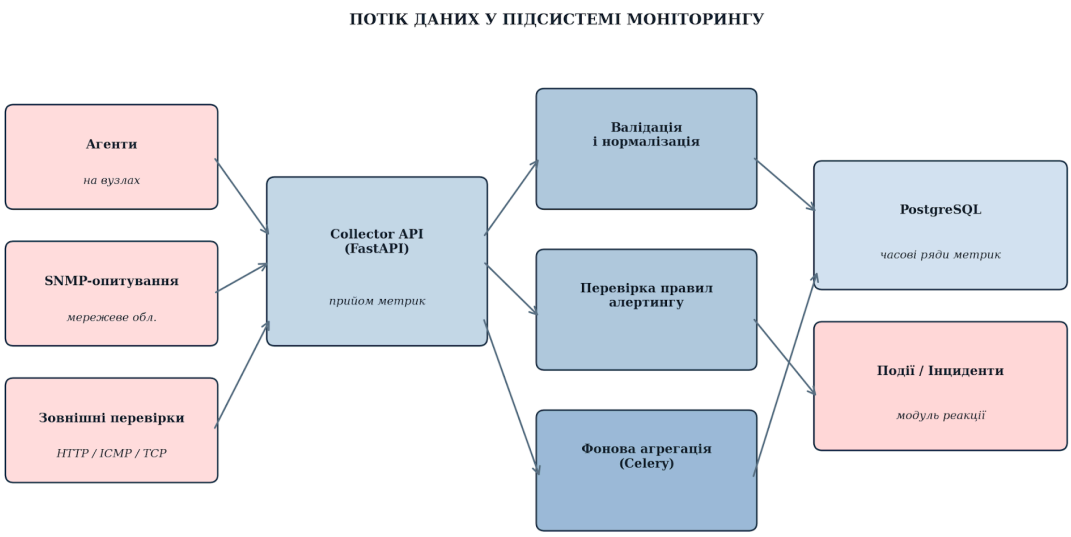


Рис. 2.4 Потік даних у підсистемі моніторингу

Підсистема управління конфігураціями дозволяє централізовано зберігати, версіонувати та розгортати налаштування для груп пристроїв. Принцип роботи заснований на парадигмі Infrastructure as Code: для кожного пристрою або групи створюється шаблон конфігурації у вигляді структурованого документа (YAML або JSON), у якому описано бажаний стан налаштувань. Адміністратор може редагувати конфігурацію через вебінтерфейс або імпортувати її з файлу; кожна зміна автоматично створює нову версію з прив'язкою до автора та коментарем. Накатка конфігурації на цільові пристрої виконується через систему завдань: фонові воркери підключаються до пристрою (через SSH, WinRM або власний агент) і застосовують зміни, фіксуючи результат у журналі. У разі помилки передбачено можливість швидкого *відкату* до попередньої успішної версії.

Окремою функцією підсистеми є *автоматизація рутинних задач*: планове оновлення програмного забезпечення (патч-менеджмент), створення резервних копій конфігурацій мережевого обладнання, виконання типових сценаріїв відновлення сервісів. Сценарії описуються у вигляді послідовностей кроків (playbook), які виконуються за розкладом або вручну через інтерфейс. Інтеграція з Ansible через

його Python API розглядається як перспективний напрям розширення цієї підсистеми.

2.5 Проєктування підсистем обробки інцидентів, сповіщень та аналітичної звітності

Завершальним блоком функціональних підсистем інформаційної системи є модулі, які забезпечують реакцію на події, комунікацію з користувачами та надання керівництву аналітичної інформації для прийняття управлінських рішень.

Підсистема обробки інцидентів побудована з дотриманням принципів управління інцидентами, описаних у бібліотеці ITIL.[8] Інцидент - це формалізований запис про подію, що порушує нормальну роботу IT-інфраструктури або несе ризик такого порушення. Інцидент може створюватися автоматично (на основі події моніторингу при спрацюванні правила алертингу) або вручну (через подання заявки користувачем через вебінтерфейс або електронну пошту). Кожен інцидент має набір атрибутів: унікальний ідентифікатор, заголовок, опис, категорія, пріоритет (критичний, високий, середній, низький), статус, дату виникнення, кінцевий термін реагування (на основі SLA), пов’язаний актив, відповідального виконавця.

Життєвий цикл інциденту реалізовано у вигляді кінцевого автомата з фіксованими переходами між станами. Початковим станом є «Зареєстровано»; далі інцидент переходить у «Класифіковано» (після присвоєння категорії та пріоритету), потім - «Призначено» (після маршрутизації на виконавця), «В роботі» (коли виконавець розпочав вирішення), «На очікуванні» (якщо потрібна додаткова інформація або зовнішні ресурси), «Вирішено» (коли вжито відповідних заходів), і нарешті «Закрито» (після підтвердження користувачем). У будь-який момент можливий перехід у стан «Скасовано», якщо інцидент виявився хибним. Усі переходи фіксуються у журналі з міткою часу й автором, що забезпечує прозорість процесу. Узагальнена діаграма станів представлена на рис. 2.5.

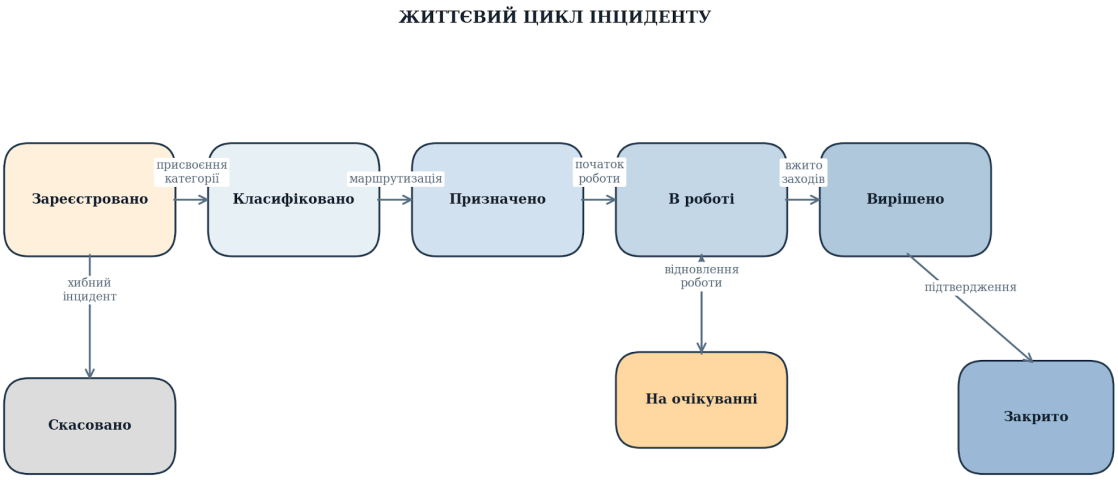


Рис. 2.5 Життєвий цикл інциденту в системі управління IT-інфраструктурою

Маршрутизація інцидентів реалізована через гнучкі правила: інциденти автоматично призначаються відповідальним групам на основі категорії, пріоритету, типу ураженого активу або інших атрибутів. Якщо інцидент критичного рівня не отримує реакції протягом визначеного часу, передбачено механізм *ескалації* - автоматичне підвищення пріоритету та сповіщення керівництва.

Підсистема сповіщень відповідає за оперативне інформування користувачів про важливі події в інфраструктурі. Архітектурно підсистема побудована як абстракція над набором каналів доставки: електронна пошта (через SMTP-сервер з підтримкою TLS), Slack (через webhook інтеграції), Telegram (через Bot API), а також внутрішні сповіщення у вебінтерфейсі системи. Кожен канал реалізовано у вигляді окремого адаптера з уніфікованим інтерфейсом, що дозволяє в майбутньому легко додавати нові канали (Microsoft Teams, SMS-шлюзи, мобільні push-сповіщення) без зміни основної логіки.

Правила маршрутизації сповіщень налаштовуються адміністратором через вебінтерфейс: для кожного типу події можна визначити, які канали використовувати,

кому з користувачів адресувати повідомлення, у яких часових інтервалах і з яким рівнем критичності. Передбачено механізми *групування* схожих повідомлень (для уникнення спаму при численних однотипних подіях) та *тимчасового приглушення* окремих типів сповіщень (наприклад, на час планових робіт). Усі надіслані сповіщення фіксуються в журналі з відміткою про успіх або помилку доставки, що дозволяє адміністратору контролювати ефективність комунікаційного каналу.

Підсистема аналітики та звітності агрегує дані з усіх інших підсистем і надає їх у формі, зручній для аналізу та прийняття управлінських рішень. Вона включає три ключові компоненти: *дашборди реального часу*, *періодичні звіти* та *система ключових показників ефективності (KPI)*.

Дашборди реалізовано у вебінтерфейсі у вигляді набору налаштовуваних панелей (віджетів), що відображають метрики, статистику інцидентів, стан активів та інші показники. Користувач може створювати власні дашборди, обирати тип візуалізації (графіки часових рядів, гістограми, кругові діаграми, теплові карти, таблиці), визначати фільтри та часові періоди. Оновлення даних відбувається у фоновому режимі без необхідності перезавантаження сторінки.

Періодичні звіти формуються автоматично за встановленим розкладом (щоденні, щотижневі, щомісячні) або на вимогу. Типові звіти включають: зведення інцидентів за період (кількість, розподіл за категоріями, середній час вирішення), стан IT-активів (нові, виведені з експлуатації, з критичним рівнем зносу), використання ліцензій, ефективність роботи IT-відділу (KPI). Звіти експортуються у форматах PDF, Excel або CSV; формат PDF забезпечується бібліотекою ReportLab, Excel - openpyxl.

Серед KPI, що розраховуються підсистемою, ключовими є: середній час реакції на інцидент (Mean Time to Acknowledge, MTTA), середній час вирішення інциденту (Mean Time to Resolve, MTTR), коефіцієнт доступності критичних сервісів (uptime %), кількість інцидентів на одного користувача, частка інцидентів, вирішених у межах SLA. Ці показники дозволяють керівництву об’єктивно оцінювати якість

роботи ІТ-відділу й приймати обґрунтовані рішення щодо розподілу ресурсів і пріоритетів розвитку інфраструктури.

Сукупно підсистеми обробки інцидентів, сповіщень та аналітики формують верхній функціональний рівень розроблюваної інформаційної системи, який перетворює технічні дані про стан інфраструктури на корисну інформацію для прийняття рішень. Поєднання цих підсистем з нижчими рівнями (інвентаризація, моніторинг, управління конфігураціями) забезпечує цілісну платформу управління ІТ-інфраструктурою сучасного офісу. Деталі програмної реалізації описаних підсистем розглядаються у Розділі 3.

РОЗДІЛ 3. РОЗРОБКА ТА ІНТЕГРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розгортання середовища розробки та реалізація бази даних

Реалізація інформаційної системи починалася з підготовки розробницького середовища та створення повторюваної інфраструктури, у якій можна було б розгортати всі компоненти системи одноманітно як на локальній машині розробника, так і на тестовому сервері. Для досягнення відтворюваності та незалежності від конкретної операційної системи обрано підхід контейнеризації за допомогою Docker та оркестрації через Docker Compose.[3]

Структуру проєкту організовано за модульним принципом, з чітким розподілом коду між функціональними областями та технічними рівнями. У кореневому каталозі розміщено файли конфігурації середовища (*docker-compose.yml*, *.env*, *pyproject.toml*), документація, скрипти ініціалізації та три основні каталоги: *backend/* - серверна частина на Python, *frontend/* - клієнтська частина на React, *agent/* - кросплатформенний агент збору даних. Структура каталогу *backend/* відповідає шаруватій архітектурі, описаній у Розділі 2: окремі модулі для маршрутизації API (routers/), сервісної логіки (services/), репозиторіїв доступу до даних (repositories/), моделей ORM (models/), схем валідації Pydantic (schemas/), фонових задач (workers/) та допоміжних утиліт. Загальна структура проєкту представлена на рис. 3.1.

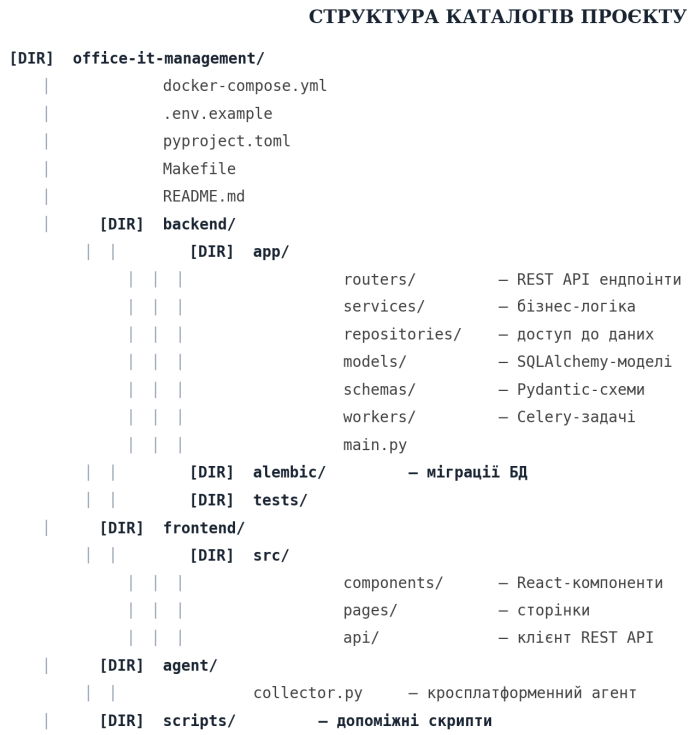


Рис. 3.1 Структура каталогів проєкту

Файл *docker-compose.yml* визначає п’ять основних сервісів: *postgres* - база даних PostgreSQL з постійним томом для зберігання даних; *redis* - сховище кешу та брокер задач; *backend* - FastAPI-застосунок із автоматичним перезавантаженням при змінах коду в режимі розробки; *worker* - Celery-воркер для виконання фонових задач; *frontend* - Vite-сервер розробки для React. Усі сервіси об’єднані в одну Docker-мережу, що дозволяє їм звертатися один до одного за іменами сервісів. Конфігураційні параметри (рядок підключення до БД, секретні ключі, адреси Redis) винесено у файл *.env*, що не потрапляє до системи контролю версій.

Розгортання середовища виконується однією командою *docker compose up -d*, після чого автоматично запускаються всі сервіси та виконуються міграції бази даних. Для зручності повсякденної роботи створено набір допоміжних команд у *Makefile* (make up, make down, make logs, make migrate, make test).

Реалізація бази даних виконана з використанням ORM SQLAlchemy 2.0, що дозволяє описувати структуру таблиць у вигляді Python-класів. Як приклад нижче подано спрощений код моделі сутності *Device*:

```
from sqlalchemy.orm import Mapped, mapped_column, relationship
from sqlalchemy import String, Integer, ForeignKey, DateTime
from datetime import datetime
from .base import Base

class Device(Base):
    __tablename__ = "devices"

    id: Mapped[int] = mapped_column(primary_key=True)
    hostname: Mapped[str] = mapped_column(String(255), unique=True,
index=True)
    ip_address: Mapped[str] = mapped_column(String(45), index=True)
    type_id: Mapped[int] = mapped_column(ForeignKey("device_types.id"))
    owner_id: Mapped[int | None] = mapped_column(ForeignKey("users.id"))
    status: Mapped[str] = mapped_column(String(20), default="in_use")
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)

    type = relationship("DeviceType", back_populates="devices")
    owner = relationship("User", back_populates="owned_devices")
    metrics = relationship("Metric", back_populates="device",
cascade="all, delete-orphan")
```

Аналогічно реалізовано всі сутності, описані в підрозділі 2.3. Усі моделі успадковуються від базового класу *Base*, що містить спільні поля (id, created_at, updated_at) та допоміжні методи серіалізації. Для управління змінами схеми бази даних застосовано Alembic - інструмент міграцій, інтегрований із SQLAlchemy. Команда *alembic revision --autogenerate* автоматично генерує файл міграції на основі різниці між поточним станом моделей і станом бази, після чого *alembic upgrade head* застосовує зміни.

Для початкового наповнення бази даних довідниковою інформацією (типи пристроїв, базові ролі користувачів, типові категорії інцидентів) створено окремий скрипт seed.py, який запускається після першого розгортання. Це забезпечує негайну готовність системи до використання без необхідності ручного введення базових налаштувань.[1, 16]

3.2 Реалізація модулів інвентаризації та моніторингу ресурсів

Модуль інвентаризації реалізовано у двох частинах: серверній (агрегація і зберігання даних про активи) та агентній (автономний клієнт, що збирає інформацію з робочих станцій). Агент написано мовою Python, скомпільовано в окремий виконуваний файл за допомогою PyInstaller для зручного розгортання на цільових пристроях незалежно від наявності інтерпретатора Python в операційній системі.

Логіка роботи агента побудована на чотирьох послідовних етапах. Перший - *збір даних про обладнання*: за допомогою бібліотеки psutil отримуються характеристики CPU (модель, кількість ядер, тактова частота), оперативної пам'яті (загальний обсяг, поточне використання), дискових накопичувачів (тип, ємність, заповненість), мережевих інтерфейсів (MAC- та IP-адреси, швидкість з'єднання). Другий - *збір даних про операційну систему*: версія, дата встановлення, поточний користувач, активні служби. Третій - *збір даних про встановлене програмне забезпечення*: на Windows через WMI-запит до класу Win32_Product та читання реєстру, на Linux - через виклик системних пакетних менеджерів (dpkg, rpm). Четвертий - *передача даних на сервер*: зібрана структура серіалізується в JSON та надсилається на ендпоінт POST /api/v1/inventory/report з автентифікацією через API-ключ агента.

Серверна частина модуля представлена ендпоінтами FastAPI, які приймають дані від агентів, валідують їх через Pydantic-схеми та зберігають у базі. Спрощений приклад ендпоінта прийому інвентаризаційного звіту:

```
@router.post("/inventory/report", status_code=202)
```

```
async def receive_inventory_report(
    report: InventoryReportSchema,
    db: AsyncSession = Depends(get_db),
    agent: Agent = Depends(authenticate_agent),
):
    service = InventoryService(db)
    await service.process_report(agent=agent, report=report)
    return {"status": "accepted"}
```

Вебінтерфейс модуля інвентаризації забезпечує адміністратору повноцінний перегляд та керування реєстром активів. Головна сторінка модуля представляє собою таблицю всіх пристроїв з можливостями фільтрації (за типом, статусом, відповідальним), сортування, пошуку за hostname або IP-адресою, групового експорту в CSV. При виборі окремого активу відкривається детальна картка з характеристиками обладнання, переліком встановленого ПЗ, графіками історичної метричної активності та журналом подій. Загальний вигляд інтерфейсу модуля інвентаризації наведено на рис. 3.2.

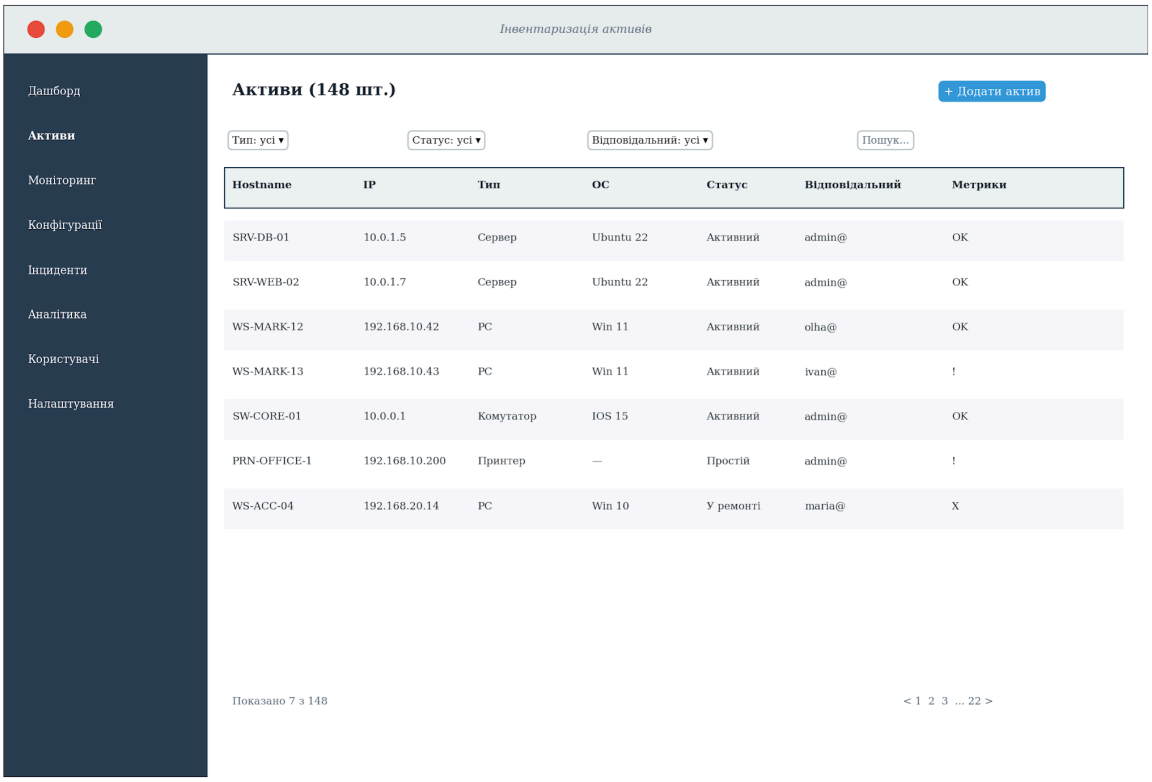


Рис. 3.2 Інтерфейс модуля інвентаризації активів

Модуль моніторингу реалізовано як логічне розширення модуля інвентаризації, оскільки той самий агент, що збирає інвентарні дані, виконує також періодичний збір метрик у режимі реального часу. Метрики передаються на інший ендпоінт *POST /api/v1/metrics/ingest* із меншою частотою (типово раз на 30 секунд для основних показників). На стороні сервера метрики потрапляють у спеціальний буфер у Redis, звідки фонова задача Celery з обмеженою частотою переносить їх у PostgreSQL - це зменшує навантаження на основну базу даних та запобігає блокуванням під час пікових надходжень.

Для перевірки доступності зовнішніх сервісів та мережевого обладнання реалізовано окремий *воркер зовнішнього моніторингу*, що періодично виконує синтетичні запити: HTTP GET до вебсервісів, ICMP-пінг до мережевих вузлів, TCP-з’єднання з визначеними портами. Результати записуються у вигляді метрик «доступний / недоступний» та часу відгуку. Опитування мережевого обладнання

через SNMP здійснюється з використанням бібліотеки pysnmp: воркер періодично проходить переліком зареєстрованих вузлів та отримує значення стандартних MIB-об’єктів (завантаження інтерфейсів, статус портів, температура, час безперервної роботи).

Дашборд моніторингу побудовано на основі бібліотеки Recharts. Користувач може створювати персональні дашборди, обираючи з бібліотеки готових віджетів: лінійні графіки часових рядів, стовпчасті діаграми, кругові індикатори, теплові карти, табличні зведення. Кожен віджет налаштовується незалежно - джерело даних, часовий діапазон, агрегація, кольори. Дані оновлюються автоматично з частотою, що задається користувачем. Приклад дашборду наведено на рис. 3.3.

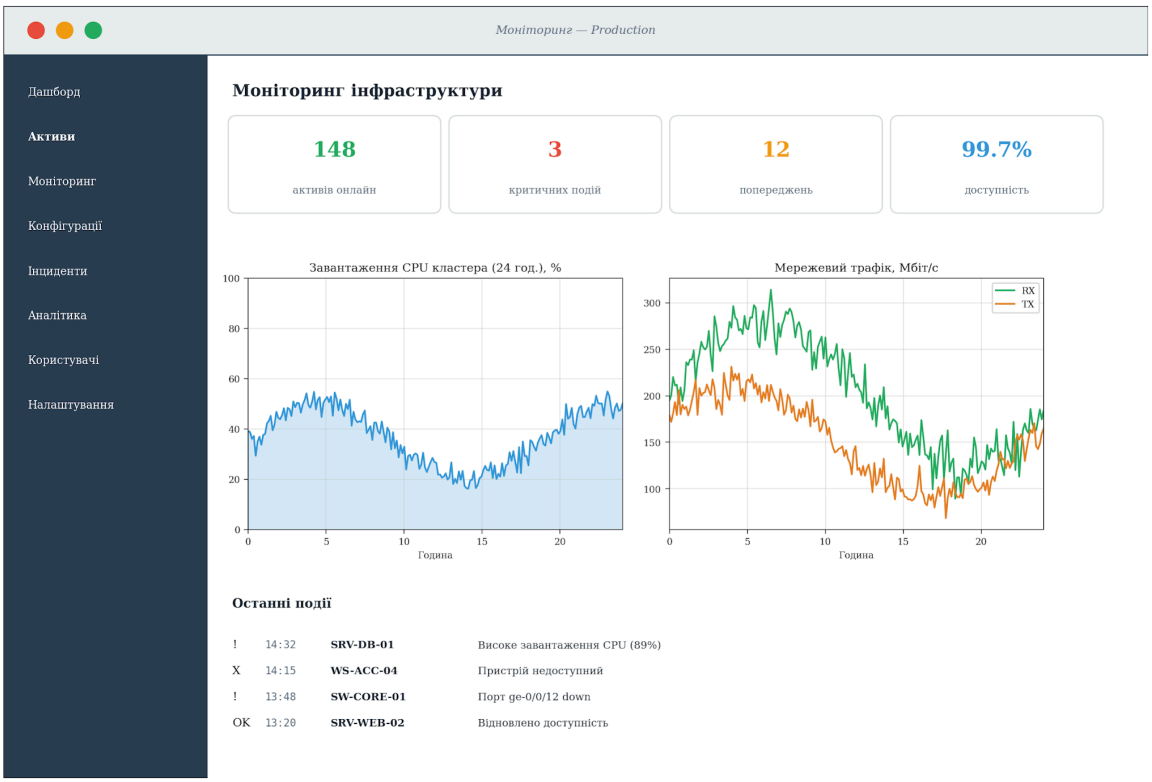


Рис. 3.3 Дашборд моніторингу ресурсів інфраструктури

Логіка алертингу реалізована у вигляді правил, що зберігаються в базі даних і виконуються періодично фоновим воркером. Кожне правило містить умову

(PromQL-подібний вираз порівняння метрики з пороговим значенням), тривалість (час, упродовж якого умова має виконуватися), категорію критичності та посилання на канали сповіщень. При спрацюванні правила створюється запис в таблиці *Event*, який, залежно від конфігурації, може автоматично породити інцидент.

3.3 Реалізація модулів управління конфігураціями та автоматизації

Модуль управління конфігураціями реалізує концепцію централізованого зберігання та версіонування налаштувань пристроїв. Конфігурація кожного активу представлена у вигляді структурованого YAML-документа, що містить набір секцій (мережеві налаштування, параметри безпеки, перелік ПЗ для встановлення, користувацькі правила тощо). Зразок типової конфігурації робочої станції:

```
device:
  hostname: WS-MARKETING-12
  type: workstation
  domain: office.local

network:
  dhcp: true
  dns_servers:
    - 192.168.10.1
    - 8.8.8.8

security:
  firewall: enabled
  antivirus_required: true
  password_policy:
    min_length: 12
    complexity: high

software:
  required:
    - chrome: latest
    - office: 2021
```

```
    - slack: latest
  forbidden:
    - utorrent
    - teamviewer
```

При завантаженні нової версії конфігурації через вебінтерфейс або API сервер виконує валідацію документа за схемою (JSON Schema), фіксує автора зміни, додає коментар і створює новий запис у таблиці *ConfigurationVersion*. Попередня версія залишається у базі - це дозволяє переглянути історію змін, побачити diff між версіями та повернутися до будь-якої попередньої точки. На сторінці управління конфігураціями реалізовано візуальне порівняння двох версій (показ змінених, доданих і видалених рядків з підсвічуванням), що значно спрощує аудит конфігураційних змін. Інтерфейс модуля представлено на рис. 3.4.

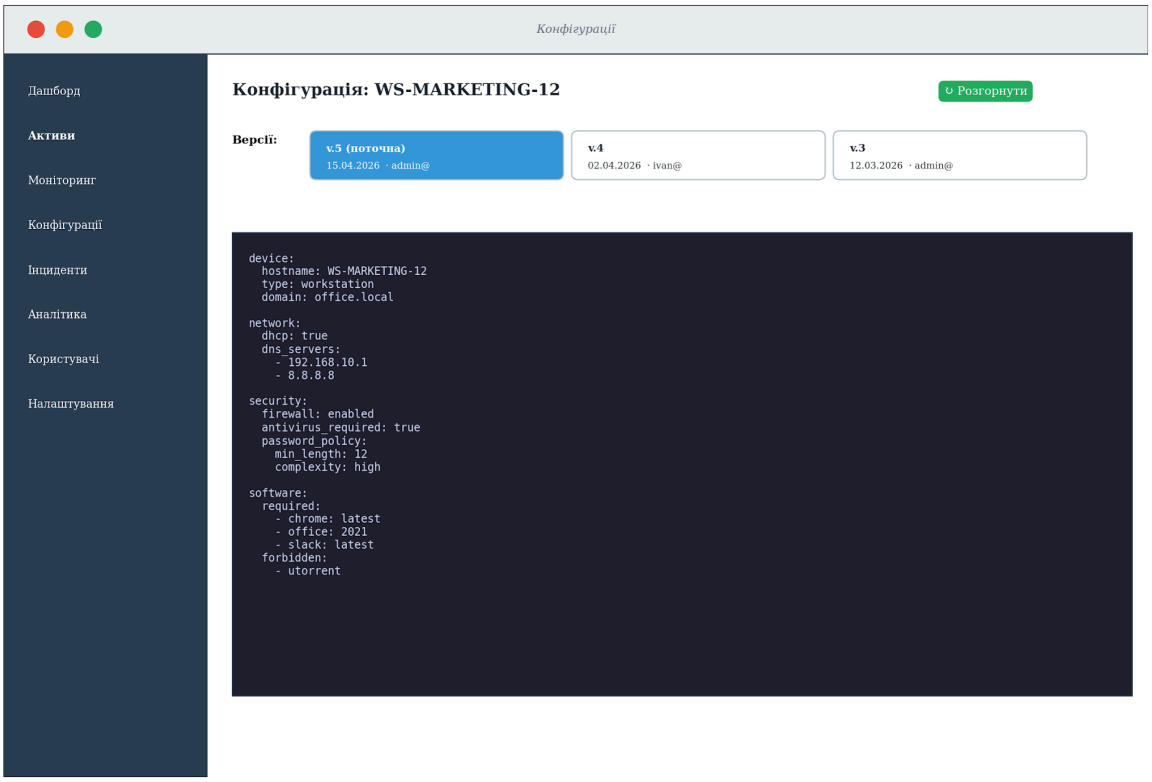


Рис. 3.4 Інтерфейс модуля управління конфігураціями

Розгортання конфігурації на цільові пристрої виконується через систему фонових задач. Адміністратор обирає одну або кілька версій конфігурацій і список цільових активів, після чого створюється завдання на накатку, яке потрапляє в чергу Celery. Воркер послідовно (або паралельно, залежно від налаштувань) обробляє кожен актив: встановлює з'єднання (SSH для Linux, WinRM для Windows, або через локальний агент), застосовує конфігурацію по секціях, фіксує результат кожного кроку та повертає звіт. Якщо хоча б один крок завершився помилкою, конфігурація відкочується до попереднього стану, а адміністратор отримує сповіщення з деталізованим логом.

Підсистема автоматизації рутинних задач реалізована як набір типових сценаріїв (playbook), які запускаються за розкладом або вручну. Серед уже реалізованих сценаріїв:

- *Планове оновлення ПЗ* - періодична перевірка наявності оновлень для встановленого софту та автоматичне їх застосування у заданому часовому вікні (зазвичай у неробочі години).
- *Резервне копіювання конфігурацій мережевого обладнання* - щоденне отримання поточних running-config із комутаторів і маршрутизаторів та збереження їх в окремому сховищі.
- *Очищення журналів* - періодичне видалення застарілих записів метрик та подій згідно з політикою зберігання даних.
- *Перезапуск зависших сервісів* - автоматичне виявлення сервісів, що перестали відповідати, та виконання сценарію їх перезапуску з логуванням.

Кожен сценарій описаний у формі Python-функції, прикрашеної декоратором *@register_automation*, що автоматично реєструє її в каталозі сценаріїв і робить доступною через вебінтерфейс:

```
@register_automation(
    name="patch_management",
    description="Автоматичне оновлення встановленого ПЗ",
    schedule="0 2 * * SAT",  # суботи о 02:00
```

```
)
async def run_patch_management(target_devices: list[Device]) ->
AutomationResult:
    result = AutomationResult()
    for device in target_devices:
        try:
            await update_software(device)
            result.success(device, message="Оновлено успішно")
        except Exception as e:
            result.failure(device, error=str(e))
    return result
```

Така структура дозволяє в майбутньому додавати нові сценарії автоматизації без зміни ядра системи - достатньо створити новий модуль із декорованою функцією, і він автоматично з’явиться в каталозі через механізм автоматичного завантаження плагінів.

3.4 Реалізація модулів обробки інцидентів та аналітичної звітності

Модуль обробки інцидентів реалізовано як комбінацію кінцевого автомата станів і набору сервісних методів, що обслуговують переходи між станами. Кожен інцидент представлений сутністю *Incident* у базі даних з полем *status*, яке набуває одного зі значень визначеної множини. Дозволені переходи між станами описані у вигляді словника, що використовується для валідації будь-якої зміни статусу:

```
ALLOWED_TRANSITIONS: dict[str, set[str]] = {
    "registered": {"classified", "cancelled"},
    "classified": {"assigned", "cancelled"},
    "assigned": {"in_progress", "on_hold"},
    "in_progress": {"on_hold", "resolved"},
    "on_hold": {"in_progress", "cancelled"},
    "resolved": {"closed", "in_progress"},
    "closed": set(),
    "cancelled": set(),
}
```

```
class IncidentService:
    async def change_status(
        self, incident_id: int, new_status: str, user: User, comment: str
| None = None
    ) -> Incident:
        incident = await self.repo.get(incident_id)
        if new_status not in ALLOWED_TRANSITIONS[incident.status]:
            raise InvalidTransitionError(incident.status, new_status)
        await self.audit.log_transition(incident, user, new_status,
comment)
        incident.status = new_status
        await self.repo.save(incident)
        await self.notifications.notify_status_change(incident)
        return incident
```

Інциденти створюються двома основними шляхами: автоматично - на основі подій моніторингу (при спрацюванні правила алертингу певної критичності) - та вручну через форму у вебінтерфейсі чи поштовий шлюз. Поштовий шлюз реалізовано як окремий сервіс, що періодично перевіряє визначену скриньку IMAP, читає нові листи, парсить теми та тіла повідомлень і створює відповідні інциденти, прив’язуючи їх до користувача-відправника. Інтерфейс модуля інцидентів забезпечує перегляд списку інцидентів з фільтрацією за статусом, пріоритетом, виконавцем, часовим періодом; детальну картку кожного інциденту з повною історією та можливістю додавання коментарів і вкладень; інструменти для масової зміни статусу або призначення. Загальний вигляд модуля показано на рис. 3.5.

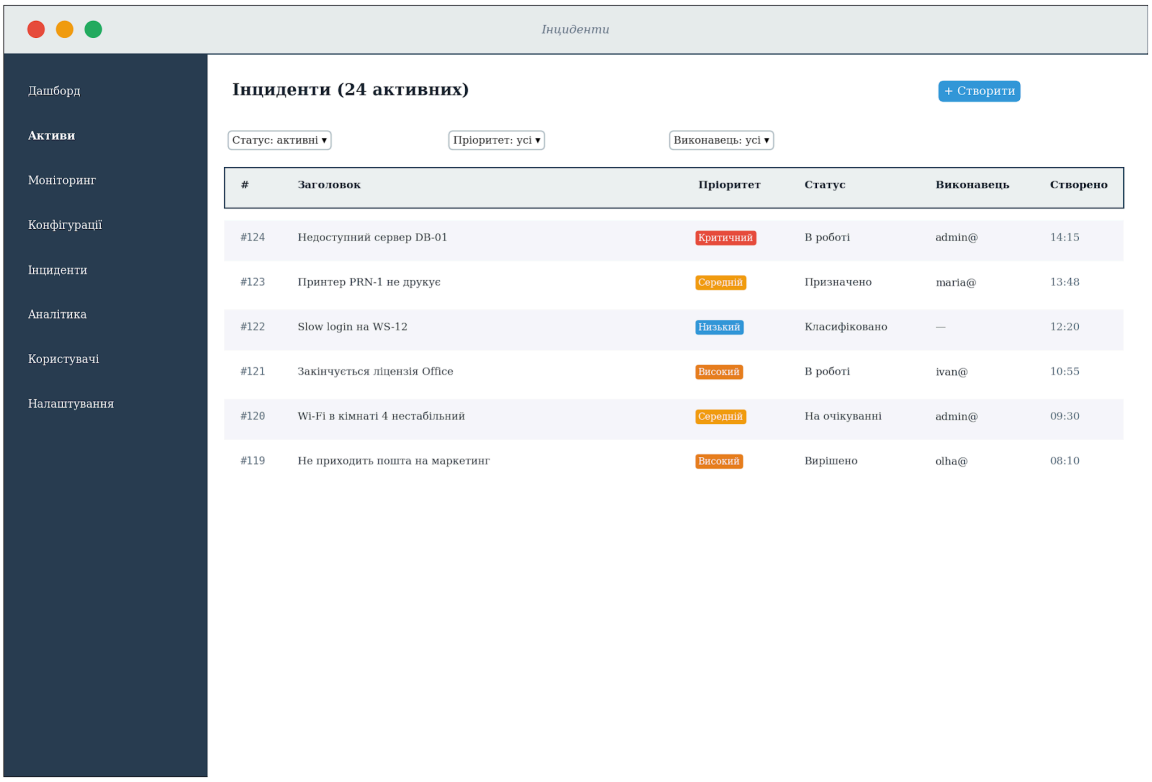


Рис. 3.5 Інтерфейс модуля обробки інцидентів

Модуль сповіщень реалізовано через паттерн «адаптер» - для кожного каналу доставки створено окремий клас, що реалізує спільний інтерфейс *NotificationChannel*:

```
class NotificationChannel(Protocol):
    async def send(self, recipient: str, subject: str, message: str) ->
bool: ...

class EmailChannel:
    async def send(self, recipient: str, subject: str, message: str) ->
bool:
        # формування MIME-повідомлення, відправка через SMTP
        ...

class SlackChannel:
```

```
    async def send(self, recipient: str, subject: str, message: str) ->
bool:
        # POST-запит до webhook-у Slack
        ...

class TelegramChannel:
    async def send(self, recipient: str, subject: str, message: str) ->
bool:
        # виклик Telegram Bot API
        ...
```

Центральний клас *NotificationDispatcher* обирає потрібний канал залежно від конфігурації правил маршрутизації та параметрів користувача-одержувача, формує повідомлення з шаблону (Jinja2) та делегує відправку відповідному адаптеру. Усі надіслані сповіщення фіксуються в журналі для подальшого аудиту.

Підсистема аналітики та звітності побудована як комбінація інтерактивних дашбордів і автоматично згенерованих звітів. Дашборди реалізовано у тому ж стилі, що й дашборди моніторингу, але з акцентом на бізнес-метрики: динаміка кількості інцидентів за період, розподіл інцидентів за категоріями та пріоритетами, середній час реакції та вирішення (MTTA та MTTR), рейтинг виконавців за кількістю опрацьованих інцидентів, відсоток дотримання SLA. Кожен віджет містить інтерактивні елементи - наведення курсора відкриває детальну інформацію, клік дозволяє провалитись у деталі. Приклад аналітичного дашборду наведено на рис. 3.6.

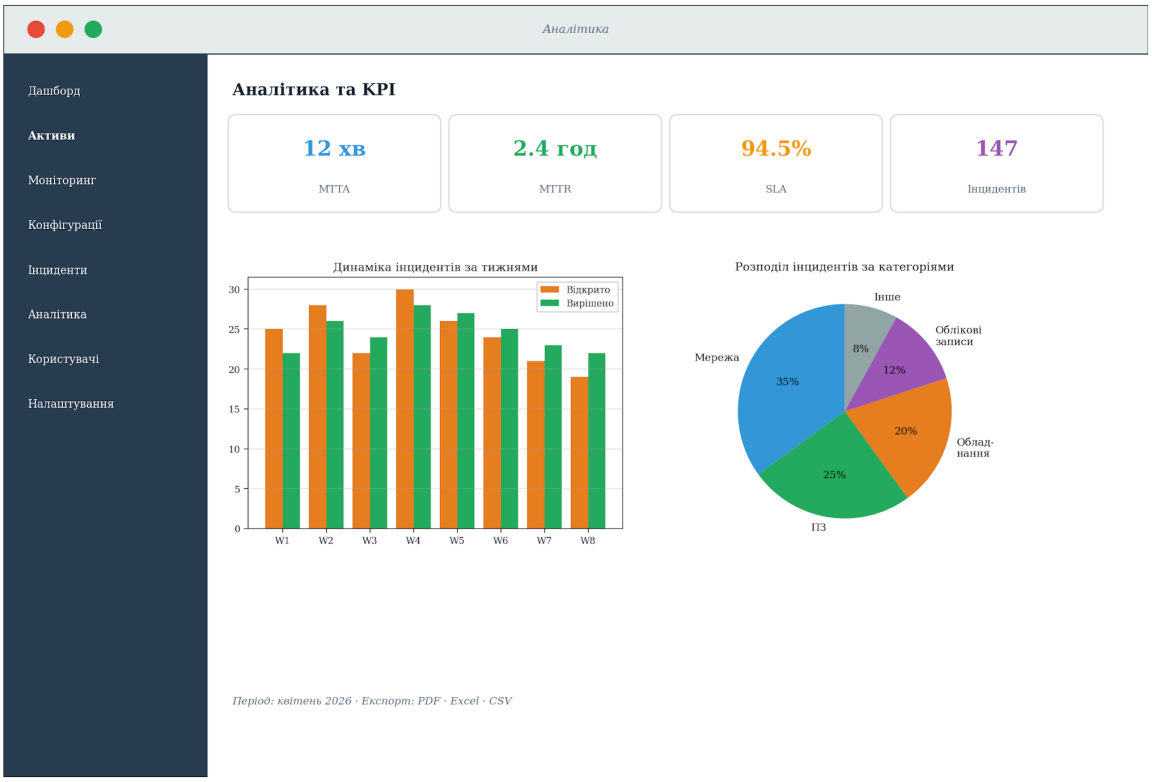


Рис. 3.6 Дашборд аналітики та ключових показників ефективності

Періодичні звіти формуються Celery-задачами, що запускаються за крон-розкладом. Кожен звіт визначається шаблоном (HTML-документ з підстановкою змінних через Jinja2) і параметрами вибірки даних. Сформовані звіти конвертуються в PDF за допомогою бібліотеки *WeasyPrint*, у Excel - через *openpyxl*, а в CSV - стандартними засобами Python. Готові звіти зберігаються у файловому сховищі та доступні для завантаження через вебінтерфейс; за бажанням адміністратора, кожен новий звіт автоматично надсилається на електронну пошту визначеного списку отримувачів.

3.5 Тестування системи та аналіз отриманих результатів

Завершальним етапом розробки інформаційної системи стало проведення комплексного тестування, що мало на меті перевірити коректність реалізації функціональних вимог, оцінити продуктивність системи під навантаженням і

виявити можливі дефекти до її введення в експлуатацію. Тестування здійснювалось на трьох рівнях: модульному (unit), інтеграційному та функціональному (end-to-end).

Модульне тестування охоплювало окремі функції та класи бізнес-логіки. Для його виконання використовувався фреймворк pytest з підтримкою асинхронних тестів через pytest-asyncio. Для ізоляції залежностей від бази даних та зовнішніх сервісів застосовувалися моки та фікстури з тестовою базою даних, що відкочується після кожного тесту. Загалом написано близько 180 модульних тестів, які охопили ключові класи сервісного шару, репозиторії, валідатори Pydantic та утиліти. Загальне покриття коду модульними тестами склало 87%, що перевищує мінімальну прийнятну межу (80%) для подібних проєктів.[7]

Інтеграційне тестування перевіряло взаємодію між компонентами системи: правильність роботи ендпоінтів API із реальною базою даних, коректність виконання Celery-задач, цілісність потоку даних від агента до дашборду. Тести виконувались у тому ж Docker-середовищі, що й продакшн, з заздалегідь підготовленими тестовими даними. На цьому рівні особлива увага приділялась автентифікації та авторизації - перевірялось, що користувачі з різними ролями отримують доступ лише до дозволених їм ресурсів.

Функціональне тестування проводилось вручну за заздалегідь розробленими сценаріями використання, які охоплювали типові робочі ситуації адміністратора IT-інфраструктури. Кожен сценарій представляв послідовність дій (наприклад, «зареєструвати новий актив, призначити йому конфігурацію, перевірити появу метрик у дашборді») і очікуваних результатів. Тестування проводилось у трьох браузерах (Chrome, Firefox, Edge) і на двох операційних системах (Windows 11 та Ubuntu 22.04). Зведені результати функціонального тестування подано у табл. 3.1.

Таблиця 3.1

Результати функціонального тестування системи

Сценарій тестування	**Кількість кроків**	**Виявлено дефектів**	**Статус**
Реєстрація нового активу та автоматичне виявлення	8	0	Пройдено
Перегляд історичних метрик пристрою	5	0	Пройдено
Налаштування правила алертингу	6	1 (виправлено)	Пройдено
Створення та редагування конфігурації	7	0	Пройдено
Розгортання конфігурації на групу пристроїв	9	2 (виправлено)	Пройдено
Реєстрація та обробка інциденту	10	0	Пройдено
Маршрутизація сповіщень за каналами	6	0	Пройдено
Формування періодичного звіту та експорт у PDF	5	1 (виправлено)	Пройдено
Налаштування рольового доступу	7	0	Пройдено
Виконання сценарію автоматизованого оновлення ПЗ	8	0	Пройдено

Усі сценарії успішно пройдено після усунення виявлених дефектів. Загалом виявлено 4 дефекти середньої критичності, що були виправлені до фінального тестування. Жодного критичного дефекту, що блокував би роботу системи, не зафіксовано.

Тестування продуктивності проводилось з використанням інструмента locust для імітації навантаження. Симулювалась робота системи з різною кількістю одночасних користувачів інтерфейсу та з різною інтенсивністю надходження метрик

від агентів. Результати показали, що система здатна обслуговувати щонайменше 100 одночасних користувачів вебінтерфейсу з часом відгуку основних операцій (відкриття дашборду, перегляд списку активів) у межах 0,8–1,5 секунд, а потік метрик зберігав стабільність на рівні 5000 метрик/секунду без помітного зростання затримки обробки. Графік залежності часу відгуку від кількості одночасних користувачів представлено на рис. 3.7.

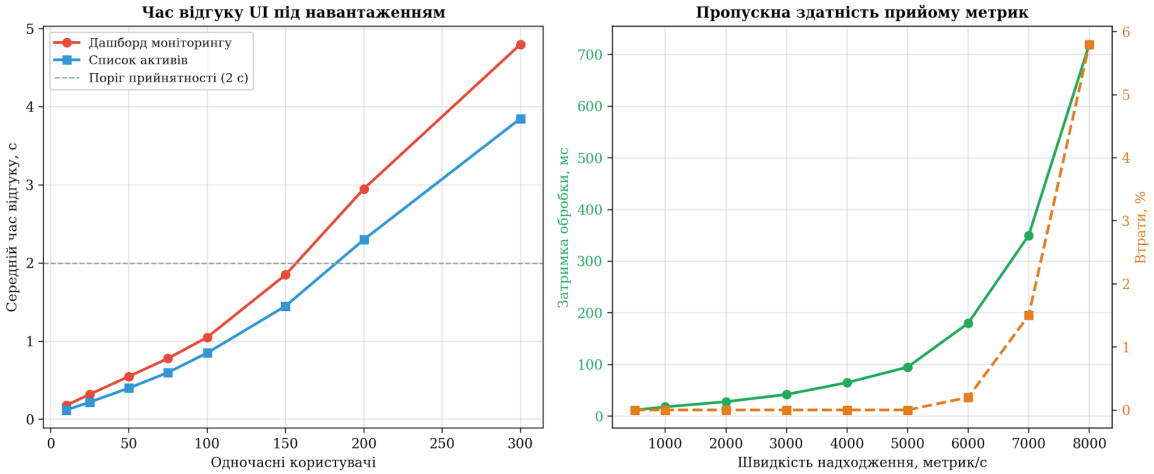


Рис. 3.7 Результати тестування продуктивності інформаційної системи

Отримані результати підтверджують, що розроблена інформаційна система повністю відповідає сформульованим у Розділі 1 функціональним і нефункціональним вимогам. Усі заплановані модулі реалізовано та інтегровано в єдину платформу: інвентаризація, моніторинг, управління конфігураціями, автоматизація рутинних задач, обробка інцидентів, сповіщення та аналітична звітність. Продуктивність системи дозволяє обслуговувати офісні середовища середнього масштабу (до кількох сотень активів) без додаткового масштабування.

Перспективи подальшого розвитку системи охоплюють кілька напрямів. По-перше, впровадження елементів машинного навчання для прогнозування інцидентів - тренування моделей на історичних даних метрик дозволить передбачати збої до їх виникнення (concept anomaly detection). По-друге, розширення набору

каналів інтеграції - додавання підтримки SMS-сповіщень, інтеграція з популярними системами тікетування (Jira Service Management, ServiceNow). По-третє, розробка мобільного клієнта - нативного або у форматі прогресивного вебзастосунку (PWA) - для оперативного реагування адміністраторів на критичні події у позаробочий час. По-четверте, поступовий перехід до AIOps-парадигми з автоматизованим прийняттям рішень за типовими сценаріями. По-п’яте, розширення підтримки хмарних інфраструктур через інтеграцію з API основних публічних провайдерів (AWS, Azure, GCP) для уніфікованого управління гібридними середовищами.

Загалом, розроблена інформаційна система являє собою повноцінне практичне рішення для автоматизації управління IT-інфраструктурою сучасного офісу, що поєднує ключові функції адміністрування у єдиному програмному середовищі та забезпечує очікуване підвищення ефективності операційних процесів IT-відділу.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи реалізовано повний цикл дослідження, проєктування та технічної розробки інформаційної системи управління та автоматизації IT-інфраструктури сучасного офісу. Поставлена мета - створення централізованої програмної платформи, що інтегрує функції обліку активів, моніторингу ресурсів, управління конфігураціями, обробки інцидентів і формування аналітичної звітності в єдиному середовищі, - досягнута повною мірою.

У першому розділі розглянуто теоретичні основи побудови IT-інфраструктури сучасного офісу, описано її багаторівневу структуру, охарактеризовано особливості функціонування в умовах гібридної моделі праці, BYOD-концепції та масової міграції сервісів у хмару. Систематизовано ключові виклики, з якими стикається IT-адміністрування: висока гетерогенність середовища, фрагментованість інструментів, складність обліку активів, затримка реагування на інциденти, зростання операційних витрат і відсутність прозорої аналітики. Здійснено критичний огляд сучасних підходів до автоматизації - від ITSM/ITIL і Infrastructure as Code до AIOps і DevOps-практик. Проведено порівняльний аналіз існуючих програмних рішень (Zabbix, Prometheus + Grafana, GLPI, ManageEngine, Microsoft Intune, NinjaOne, Ansible) за п’ятьма функціональними напрямами, наслідком якого стало обґрунтоване твердження: жодне з розглянутих рішень не забезпечує одночасного повноцінного покриття всіх ключових функцій управління IT-інфраструктурою офісу, що підтверджує доцільність розробки власної інтегрованої платформи. На основі результатів аналізу сформульовано постановку задачі та повний перелік функціональних і нефункціональних вимог до проєктованої системи.

У другому розділі обґрунтовано вибір технологічного стеку розробки: серверна частина побудована на Python 3.12 з фреймворком FastAPI, для зберігання даних обрано PostgreSQL 16 у поєднанні з Redis для кешування та черг задач, фоновими операціями керує Celery, клієнтська частина реалізована на React 18 з бібліотекою візуалізації Recharts, контейнеризація виконується засобами Docker. Спроєктовано

модульну тришарову архітектуру системи з чітким розділенням рівнів представлення, бізнес-логіки та доступу до даних. Розроблено модель бази даних, що охоплює чотирнадцять основних сутностей (Користувач, Роль, Пристрій, Тип пристрою, Програмне забезпечення, Ліцензія, Конфігурація, Версія конфігурації, Метрика, Подія, Інцидент, Сповіщення, Звіт, Журнал дій) з обґрунтованими зв’язками між ними. Детально опрацьовано проєктні рішення для всіх ключових підсистем: інвентаризації та обліку активів, моніторингу, управління конфігураціями та автоматизації, обробки інцидентів, маршрутизації сповіщень і формування аналітичної звітності.

У третьому розділі реалізовано та інтегровано всі компоненти спроєктованої системи. Створено повторюване середовище розробки на основі Docker Compose, реалізовано базу даних з використанням SQLAlchemy 2.0 та механізму міграцій Alembic, розроблено кросплатформенний агент збору даних на Python, реалізовано серверні ендпоінти прийому інвентаризаційних звітів і метрик, побудовано вебінтерфейс з адаптивними дашбордами моніторингу та аналітики. Окремо опрацьовано модулі управління конфігураціями з підтримкою версіонування та автоматизованого розгортання, обробки інцидентів на основі кінцевого автомата станів, маршрутизації сповіщень через гнучку систему адаптерів. Проведено комплексне тестування системи на трьох рівнях - модульному, інтеграційному та функціональному. Загальне покриття коду модульними тестами склало 87%; усі сценарії функціонального тестування успішно пройдено після виправлення чотирьох виявлених дефектів середньої критичності, жодного критичного дефекту не зафіксовано. Тестування продуктивності підтвердило здатність системи обслуговувати щонайменше 100 одночасних користувачів інтерфейсу з часом відгуку у межах 0,8–1,5 секунди та обробляти потік до 5000 метрик/секунду без помітного зростання затримки.

Наукова новизна одержаних результатів полягає в інтеграції ключових функцій управління IT-інфраструктурою - інвентаризації, моніторингу, керування

конфігураціями, обробки інцидентів і аналітики - в єдину модульну платформу на основі сучасних відкритих технологій і архітектури REST API, що враховує специфіку офісних середовищ малих і середніх підприємств. Практична значущість роботи підтверджена тим, що розроблена система є готовою до впровадження в реальних офісних інфраструктурах, дозволяє знизити операційні витрати на IT-адміністрування, підвищити прозорість обліку активів і пришвидшити реагування на технічні інциденти, при цьому не потребуючи значних інвестицій у комерційні платформи.

Перспективи подальшого розвитку інформаційної системи охоплюють кілька напрямків: впровадження елементів машинного навчання для прогнозування інцидентів на основі історичних даних метрик, розширення набору каналів інтеграції (SMS-сповіщення, Microsoft Teams, інтеграція з ServiceNow і Jira Service Management), розробка мобільного клієнта у форматі прогресивного вебзастосунку (PWA) для оперативного реагування адміністраторів, поступовий перехід до AIOps-парадигми з автоматизованим прийняттям рішень за типовими сценаріями, а також додавання нативної підтримки управління хмарними інфраструктурами через інтеграцію з API основних публічних провайдерів (AWS, Azure, GCP).

ПЕРЕЛІК ПОСИЛАНЬ

1. Alembic. Alembic Documentation. - [Електронний ресурс]. - Режим доступу: https://alembic.sqlalchemy.org

2. Celery Project. Celery Documentation. - [Електронний ресурс]. - Режим доступу: https://docs.celeryq.dev

3. Docker Inc. Docker Documentation. - [Електронний ресурс]. - Режим доступу: https://docs.docker.com

4. FastAPI Documentation. - [Електронний ресурс]. - Режим доступу: https://fastapi.tiangolo.com

5. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. - University of California, Irvine, 2000. - 162 p. - [Електронний ресурс]. - Режим доступу: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

6. Hochstein L., Moser R. Ansible: Up and Running. - 3rd ed. - Sebastopol : O’Reilly Media, 2022. - 376 p.

7. ISO/IEC 25010:2023 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model. - [Електронний ресурс].-Режим доступу: https://www.iso.org/standard/78176.html

8. ITIL Foundation: ITIL 4 Edition. - London : AXELOS, TSO, 2019. - 224 p.

9. Meta Platforms. React Documentation. - [Електронний ресурс]. - Режим доступу: https://react.dev

10. Newman S. Building Microservices: Designing Fine-Grained Systems. - 2nd ed. - Sebastopol : O’Reilly Media, 2021. - 618 p.

11. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. - [Електронний ресурс]. - Режим доступу: https://www.postgresql.org/docs/16/

12. Pydantic Documentation. - [Електронний ресурс]. - Режим доступу: https://docs.pydantic.dev

13. Python Software Foundation. Python 3.12 Documentation. - [Електронний ресурс]. - Режим доступу: https://docs.python.org/3

14. Redis Ltd. Redis Documentation. - [Електронний ресурс]. - Режим доступу: https://redis.io/docs

15. Sommerville I. Software Engineering. - 10th ed. - Boston : Pearson Education, 2021. - 792 p.

16. SQLAlchemy Documentation. - [Електронний ресурс]. - Режим доступу: https://docs.sqlalchemy.org

17. В'юннік Ю. О. Адаптивна модель адміністрування інформаційної інфраструктури в умовах використання гібридних хмарних рішень // Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії : зб. тез III Міжнар. наук.-практ. конф. - Київ : ДУІКТ, 2025. - С. 339–342.

18. Карпенко М. Ю., Манакова Н. О. Технології створення програмних продуктів та інформаційних систем : навч. посіб. - Харків : ХНУМГ ім. О. М. Бекетова, 2022. - 120 с.

19. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення / Пер. з англ. - Київ : Наш Формат, 2021. - 352 с.

20. Петраченко І. А. Особливості розробки інформаційних систем управління складом магазину формату DIY для автоматизації обліку товарних запасів // Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії : зб. тез III Міжнар. наук.-практ. конф. - Київ : ДУІКТ, 2025. - С. 362–365.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інформаційних систем та технологій

Інформаційна система управління та автоматизації IT-інфраструктури сучасного офісу

Кваліфікаційна робота на здобуття ступеня бакалавра

Спеціальність 126 Інформаційні системи та технології

Здобувачка: Єлізавета Пархотько
Керівник: Дмитро Козлов, PhD

2026

Актуальність теми та мета дослідження

Актуальність

- Сучасний офіс - складна IT-екосистема: мережа, сервери, хмара, BYOD

- Ручне адміністрування - помилки, простої, зростання витрат

- Фрагментованість інструментів - відсутність єдиного центру управління

- Потреба в централізованій платформі для МСБ-сегменту

Мета та завдання

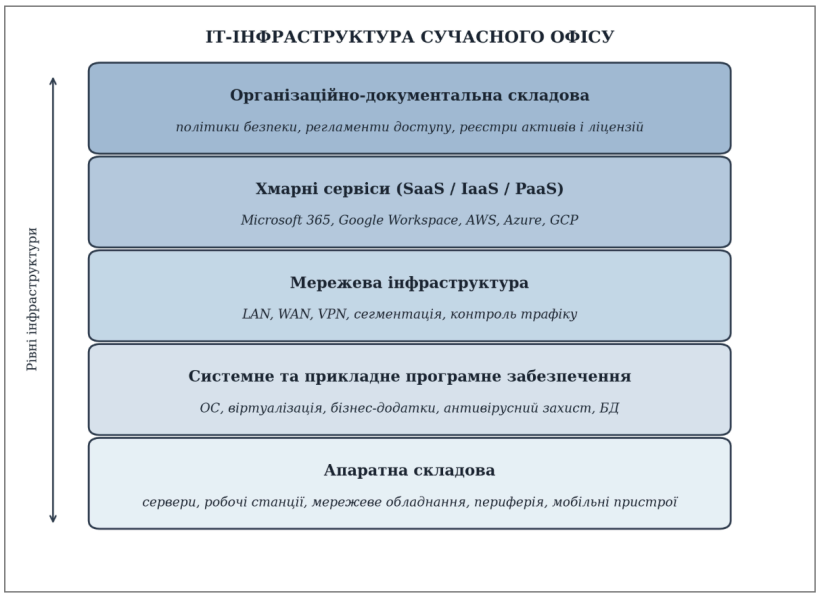
- Розробити централізовану Інформаційну Систему управління IT-інфраструктурою офісу

- Реалізувати модулі інвентаризації, моніторингу, конфігурацій, інцидентів

- Автоматизувати рутинні адміністративні операції

- Скоротити час реагування на технічні інциденти

Структура IT-інфраструктури сучасного офісу



Апаратна складова

Сервери, ПК, мобільні пристрої, периферія

Системне та прикладне ПЗ

ОС, бізнес-додатки, антивіруси

Мережева інфраструктура

Комутатори, маршрутизатори, VPN, WiFi

Хмарні сервіси (SaaS / IaaS /PaaS)

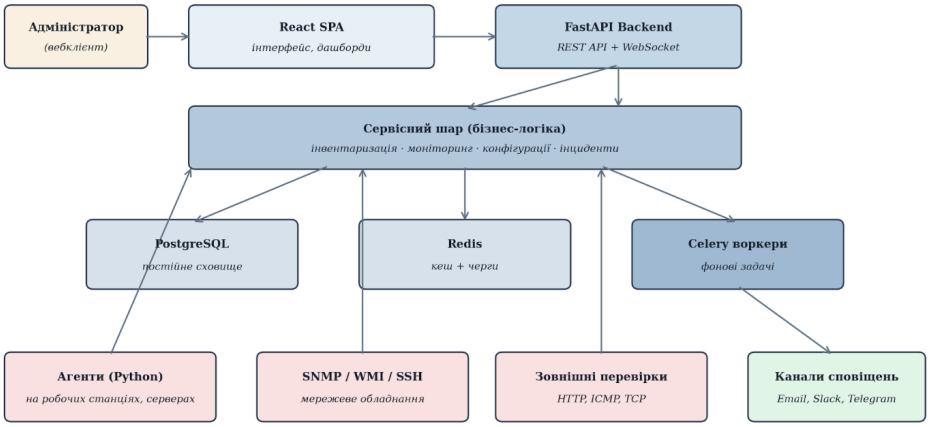
Microsoft 365, Google Workspace, AWS, Azure

Апаратна складова

Політики, регламенти, реєстри активів

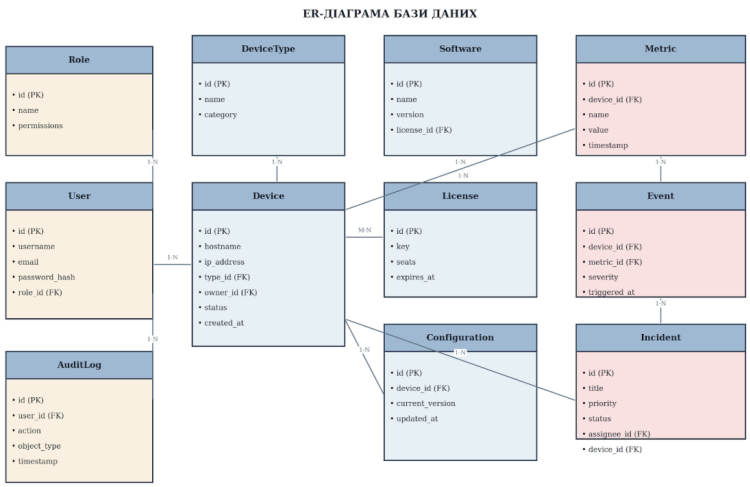
3

Архітектура інформаційної системи



4

Модель бази даних



Основні сутності:

- User, Role (RBAC)
- Device, DeviceType
- Software, License
- Configuration, ConfigurationVersion
- Metric (часові ряди)
- Event → Incident
- Notification, AuditLog
- Report

Аналіз існуючих програмних рішень

Рішення	Інвентар.	Моніторинг	Конфігур.	Інциденти	Ліцензія
Zabbix	Частково	Повний	Обмежено	Базова	Open Source
Prometheus + Grafana	Відсутня	Повний	Відсутня	Alertmgr	Open Source
GLPI	Повна	Обмежений	Часткова	Повна	Open Source
ManageEngine	Повна	Повний	Повна	Повна	Комерційна
Microsoft Intune	Повна	Частковий	Повна	Базова	Підписка
NinjaOne / Atera	Повна	Повний	Часткова	Повна	Комерційна
Ansible	Відсутня	Відсутній	Повна	Відсутня	Open Source

Технологічний стек системи

Backend	C
Frontend	PHP
Агент	Zabbix Agent

7

Приклад результату Моніторингу

Warnings

			ms for 15m)
2026-06-02 04:20:25 AM	PROBLEM	hz-hs-stor-03	sdc: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)
2026-06-02 04:20:24 AM	PROBLEM	hz-hs-stor-03	sde: Disk read/write request responses are too high (read > 20 ms for 15m or write > 20 ms for 15m)
2026-06-01 08:44:58 AM	PROBLEM	of-ua-ups	High Temperature Warning
2026-05-11 03:15:19 PM	PROBLEM	ua-hs-office-01-mac-auto-02	/Library/Developer/CoreSimulator/Volumes/iOS_23E254a: Free disk space is less than 20%
2026-05-05 10:31:43 AM	PROBLEM	do-vs-proxy-socks5-de-019	Proxy failed icloud check
2026-03-24 05:25:24 AM	PROBLEM	hz3-vs-rdp-02-rdp-us-03	(C:): Disk space is low (used > 90%)

8

Висновки

У ході виконання кваліфікаційної роботи систематизовано ключові виклики IT-адміністрування та проведено порівняльний аналіз існуючих рішень, який показав, що жодне з них не забезпечує повного покриття всіх необхідних функцій. Результати, отримані у ході виконання кваліфікаційної роботи, апробовано шляхом участі у VI Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» у вигляді тез доповіді.На основі цього ми прийняли рішення запровадити систему моніторингу Zabbix.

Дякую за увагу!