КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Кібербезпеки та комп'ютерної інженерії

(назва випускової кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему:

Аналіз та оцінка варіантів

програмного забезпечення компонентів комп'ютерної мережі

Чорний Юрій Володимирович

(прізвище, ім'я та по батькові здобувача повністю)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Делембовський М.М.

„___" _____2026 року

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Аналіз та оцінка варіантів

програмного забезпечення компонентів комп'ютерної мережі

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач
Чорний Юрій Володимирович
(прізвище, ім'я по-батькові повністю)
123 «Комп'ютерна інженерія»
(спеціальність)
Комп'ютерні системи та мережі
(освітня програма)
Група КСМ-22

Керівник Ізмайлова О.В
(прізвище та ініціали)
Кандидат технічних наук, доцент
(вчене звання, науковий ступінь)
Рецензент

(прізвище та ініціали)
Ідентичність підтверджую

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій

Випускова кафедра: Кібербезпеки та комп'ютерної інженерії

Ступінь вищої освіти: бакалавр

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Делембовський М.М.
„___" _____2026 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Чорний Юрій Володимирович

(прізвище, ім'я та по батькові здобувача)

1.Тема роботи «Аналіз та оцінка варіантів програмного забезпечення компонентів комп'ютерної мережі»
затверджена наказом ректора КНУБА № 576/52-15/13.2/26 від «14» травня 2026 року

2. Керівник роботи

Ізмайлова Ольга Василівна

кандидат технічних наук, доцент

(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту_____

4. Зміст пояснювальної записки за розділами:

Р. 1. Аналіз предметної області та засобів управління мережевою інфраструктурою

Р. 2. Проєктування інформаційної системи для оцінки програмного забезпечення

Р. 3. Практична реалізація та тестування системи для оцінки програмного забезпечення

5. **Календарний план виконання роботи:**

Види робіт та їх зміст	Дата виконання
Розділ 1. Аналіз предметної області та засобів управління мережевою інфраструктурою	03.05.2026
Розділ 2. Проєктування інформаційної системи для оцінки програмного забезпечення	14.05.2026
Розділ 3. Практична реалізація та тестування системи для оцінки програмного забезпечення	27.05.2026
Остаточне оформлення роботи	29.05.2026
Направлення роботи для перевірки на плагіат	__.__.2026
Попередній захист роботи на випусковій кафедрі	__.__.2026
Направлення роботи на рецензування	

6. Дата видачі завдання ____

Керівник _____ _____
(підпис) (прізвище та ініціали)

Здобувач _____ _____
(підпис) (прізвище та ініціали)

<table>
<tr><td>РЕЗЮМЕ
(SUMMARY)
до кваліфікаційної випускової роботи здобувача</td><td colspan="3">ПІБ здобувача українською та англійською мовами
Чорний Юрій Володимирович
Chornyi Yuriy Volodymyrovych</td></tr>
<tr><td>ЗВО</td><td colspan="3">Київський національний університет будівництва і архітектури</td></tr>
<tr><td>Тема
(українською та англійською)</td><td colspan="3">Аналіз та оцінка варіантів програмного забезпечення компонентів комп’ютерної мережі</td></tr>
<tr><td>Освітній ступінь</td><td colspan="3">бакалавр</td></tr>
<tr><td>Факультет</td><td colspan="3">Автоматизації і інформаційних технологій</td></tr>
<tr><td>Випускова кафедра</td><td colspan="3">Кібербезпеки та комп'ютерної інженерії</td></tr>
<tr><td>Спеціальність</td><td colspan="3">123 «Комп`ютерна інженерія»</td></tr>
<tr><td>Освітня програма</td><td colspan="3">Комп’ютерні системи і мережі</td></tr>
<tr><td>Керівник</td><td colspan="3">Ізмайлова О.В.</td></tr>
<tr><td rowspan="2">Обсяг роботи:</td><td>Пояснювальна записка, стор.</td><td>Розділів</td><td>Презентація, кількість слайдів</td></tr>
<tr><td>106</td><td>3</td><td>11</td></tr>
<tr><td>Розділ 1</td><td colspan="3">Проведено аналіз сучасних методів управління мережевою інфраструктурою та виконано огляд функціональних можливостей популярних програмних засобів для моніторингу та конфігурування мереж.</td></tr>
<tr><td>Розділ 2</td><td colspan="3">Розроблено архітектуру системи підтримки прийняття рішень та обґрунтовано вибір методу аналізу ієрархій як математичного інструменту для проведення багатокритеріального оцінювання програмного забезпечення.</td></tr>
</table>

Розділ 3	Реалізовано програмний комплекс, який шляхом автоматизованих розрахунків забезпечив об’єктивне порівняння альтернативних мережевих рішень та формування підсумкового рейтингу їхньої ефективності.
Висновки по роботі	У результаті виконання кваліфікаційної роботи розроблено інформаційну систему підтримки прийняття рішень, яка завдяки застосуванню методу аналізу ієрархій дозволяє автоматизувати процес багатокритеріального оцінювання та забезпечити математично обґрунтований вибір ефективного програмного забезпечення для управління корпоративною мережевою інфраструктурою.
Ключові слова: Keywords:	Корпоративна мережа, управління інфраструктурою, мережеве програмне забезпечення, інформаційна система підтримки прийняття рішень, метод аналізу ієрархій, багатокритеріальне оцінювання, автоматизація, Python, FastAPI, PostgreSQL.

Здобувач Чорний Ю.В/_____

Керівник Ізмайлова О.В/_____

АНОТАЦІЯ

Кваліфікаційна випускна робота присвячена дослідженню та програмній реалізації інформаційної системи підтримки прийняття рішень для багатокритеріального оцінювання мережевого програмного забезпечення. У роботі проаналізовано теоретичні засади управління корпоративними комп'ютерними мережами, визначено проблеми їхньої гетерогенності та обґрунтовано доцільність застосування математичних методів для вибору ефективних технічних рішень. Запропонована система базується на методі аналізу ієрархій (MAI) Томаса Сааті, що дозволяє автоматизувати процес об'єктивного вибору ПЗ на основі таких критеріїв, як сукупна вартість володіння, масштабованість, зручність розгортання та рівень безпеки. Практична частина реалізована у вигляді клієнт-серверного web-застосунку з backend на FastAPI, базою даних PostgreSQL та веб-інтерфейсом на JavaScript. Система підтримує розмежування прав доступу на основі ролей, ієрархічне структурування критеріїв, математичне моделювання матриць попарних порівнянь та генерацію підсумкових звітів. Отриманий прототип підтверджує ефективність використання методу аналізу ієрархій для мінімізації впливу людського фактора та прийняття математично обґрунтованих управлінських рішень в IT-інфраструктурі.

Ключові слова: Корпоративна мережа, управління інфраструктурою, мережеве програмне забезпечення, інформаційна система підтримки прийняття рішень, метод аналізу ієрархій, багатокритеріальне оцінювання, автоматизація, Python, FastAPI, PostgreSQL.

ABSTRACT

The qualification thesis is dedicated to the research and software implementation of a decision support information system for multi-criteria evaluation of network software. The work analyzes the theoretical principles of corporate computer network management, identifies the problems of their heterogeneity, and substantiates the expediency of using mathematical methods for selecting optimal technical solutions. The proposed system is based on Thomas Saaty's Analytic Hierarchy Process (AHP), which allows for automating the process of objective software selection based on criteria such as total cost of ownership, scalability, ease of deployment, and security level. The practical part is implemented as a client-server web application with a FastAPI backend, a PostgreSQL database, and a JavaScript web interface. The system supports role-based access control, hierarchical structuring of criteria, mathematical modeling of pairwise comparison matrices, and generation of summary reports. The resulting prototype confirms the effectiveness of using the Analytic Hierarchy Process to minimize the influence of the human factor and make mathematically grounded management decisions in IT infrastructure.

Keywords: Corporate network, infrastructure management, network software, decision support information system, Analytic Hierarchy Process, multi-criteria evaluation, automation, Python, FastAPI, PostgreSQL.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний етап цифровізації бізнес-процесів вимагає від підприємств розбудови надійного інформаційного простору на базі корпоративних комп'ютерних мереж. Їхня постійно зростаюча масштабність та гетерогенність роблять неможливим ефективне ручне адміністрування. Оскільки левова частка мережевих інцидентів спричинена помилками персоналу, критичного значення набуває перехід до автоматизованих систем моніторингу та управління інфраструктурою. Водночас велика кількість альтернативних програмних рішень, що суттєво відрізняються своєю архітектурою, функціональними можливостями та сукупною вартістю володіння, вимагає використання математично обґрунтованих підходів для їх вибору. Застосування методів багатокритеріального порівняння дозволяє усунути фактор суб'єктивності та знайти оптимальний баланс технічних і економічних характеристик. З огляду на це, розробка та впровадження спеціалізованої інформаційної системи підтримки прийняття рішень для оцінки мережевого програмного забезпечення є надзвичайно актуальним науково-прикладним завданням.

Мета дослідження полягає у розробці та реалізації інформаційної системи підтримки прийняття рішень (СППР) для автоматизації процесу об'єктивного вибору та багатокритеріального оцінювання мережевого програмного забезпечення на основі методу аналізу ієрархій (MAI).

Завдання дослідження:

- провести аналіз предметної області, виявити проблеми централізованого управління корпоративними мережами та здійснити огляд сучасних засобів управління інфраструктурою;
- визначити функціональні та нефункціональні вимоги до розроблюваної системи, а також сформувати рольову модель доступу, що включає системного адміністратора, експерта та користувача;
- здійснити об'єктно-орієнтоване проєктування архітектури системи з використанням уніфікованої мови моделювання (UML);

- адаптувати та застосувати метод аналізу ієрархій Томаса Сааті як математичний апарат для розрахунку комплексного рейтингу програмного забезпечення;
- виконати програмну реалізацію розробленого алгоритму з використанням сучасної клієнт-серверної архітектури;
- провести практичну апробацію системи шляхом порівняльного оцінювання обраних програмних продуктів за сформованою ієрархією критеріїв.

Об’єкт дослідження: процес багатокритеріального оцінювання та вибору мережевого програмного забезпечення для управління корпоративною IT-інфраструктурою.

Предмет дослідження: математичні моделі та інформаційна система підтримки прийняття рішень на базі методу аналізу ієрархій.

Методи дослідження. Для розв'язання поставлених завдань у роботі використано:

- системний аналіз — для дослідження проблем мережевого адміністрування;
- теорію прийняття рішень (зокрема, метод аналізу ієрархій) — для розв'язання багатокритерійної проблеми вибору шляхом обчислення головних власних векторів матриць;
- уніфіковану мову моделювання (UML) — для проєктування логічної та концептуальної архітектури програмного продукту.

Практичне значення одержаних результатів. Розроблена система є готовим дієвим інструментом, що повністю автоматизує складні математичні обчислення та перетворює суб'єктивні технічні оцінки експертів на строгий, кількісно вимірюваний рейтинг. Впровадження цієї системи дозволить IT-департаментам оптимізувати та стандартизувати процеси закупівлі мережевого програмного забезпечення, гарантуючи уникнення помилок людського фактора.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАСОБІВ УПРАВЛІННЯ МЕРЕЖЕВОЮ ІНФРАСТРУКТУРОЮ

1.1 Завдання та проблеми централізованого управління корпоративними комп'ютерними мережами

У сучасну епоху стрімкої модернізації та цифровізації глобальних бізнес-процесів успішна діяльність будь-якої великої промислової, фінансової чи комерційної організації критично залежить від наявності розвиненого єдиного інформаційного простору. Фундаментом такого простору виступає корпоративна комп’ютерна мережа (Intranet) — складна масштабна система, побудована з використанням протоколів стеку TCP/IP, яка є адаптованою для внутрішнього використання корпоративною версією глобальної мережі Internet. Корпоративна мережа базується на клієнт-серверній архітектурі та забезпечує передачу даних широкого спектра між різноманітними додатками, об'єднуючи локальні (LAN) та територіальні (WAN) мережі всіх структурних підрозділів підприємства.

Масштаби сучасних корпоративних мереж є надзвичайно великими: вони охоплюють сотні або тисячі робочих станцій, десятки виділених серверів, величезні обсяги інформації та розподілені по різних географічних локаціях філії. Головною особливістю таких інфраструктур є їхня яскраво виражена гетерогенність (неоднорідність). Вона полягає у використанні в межах єдиного середовища різнотипного комп'ютерного і комунікаційного обладнання, мережевих операційних систем та прикладного програмного забезпечення від різних компаній-виробників. Згідно з дослідженнями, середньостатистична сучасна корпоративна мережа використовує обладнання від 6 до 9 різних вендорів, що формує від 5 до 8 технологічних рівнів інфраструктури, де інженери змушені працювати з безліччю різних конфігураційних синтаксисів.

Керувати такою кількістю неоднорідних елементів у ручному режимі фізично неможливо. Згідно з аналітичними даними, фахівці здатні ефективно керувати вручну лише близько 275–325 мережевими елементами без втрати якості контролю. Оскільки сучасні мережі містять тисячі елементів, ручне адміністрування призводить до катастрофічних наслідків: людський фактор

становить причину приблизно 65–74% усіх мережевих збоїв, причому некоректні конфігурації є першопричиною у 42–68% випадків [1]. Це зумовлює критичну потребу в переході до систем централізованого автоматизованого управління.

Для стандартизації підходів до комплексного управління гетерогенними мережевими середовищами Міжнародна організація зі стандартизації (ISO) розробила концептуальну еталонну модель управління мережею, відому як FCAPS[4]. Ця абревіатура позначає п'ять фундаментальних категорій завдань, які повинна реалізовувати повноцінна система управління:

- Управління збоями (Fault Management): Цей рівень охоплює процеси виявлення, ізоляції, сповіщення та усунення несправностей у мережі. Мережеві елементи постійно генерують системні повідомлення (syslog) та тривожні сповіщення (SNMP traps), які відстежуються системами моніторингу. Сучасний підхід передбачає проактивне управління збоями на основі специфікацій RMON (Remote Monitoring): пристрої самостійно моніторять задані параметри (наприклад, завантаження процесора, втрати на інтерфейсах) і генерують подію лише при перетині критичних порогів.
- Управління конфігурацією (Configuration Management): Полягає у моніторингу та контролі конфігураційної інформації апаратних і програмних засобів мережі. Це включає збір і збереження конфігурацій з пристроїв, спрощення процесу налаштувань, контроль версій програмного забезпечення та відстеження будь-яких змін. Саме цей рівень є одним із найкритичніших, оскільки більшість відмов виникає як наслідок несанкціонованих або помилкових змін у конфігураційних файлах комутаторів чи маршрутизаторів.
- Управління обліком (Accounting Management): Вимірювання параметрів використання ресурсів мережі окремими користувачами або групами. Це необхідно для формування квот, оптимізації розподілу ресурсів, а також для контролю за виконанням угод про рівень обслуговування (SLA). Дані про використання трафіку збираються за допомогою технологій типу Cisco NetFlow або спеціалізованих зондів для подальшого білінгу та аналітики.

- Управління продуктивністю (Performance Management): Цей напрям зосереджений на безперервному моніторингу, вимірюванні та підтримці загальної продуктивності системи на прийнятному рівні. Використовуючи протокол SNMP, системи управління здійснюють опитування (polling) обладнання щодо метрик на рівні інтерфейсів, пристроїв та протоколів. Це завдання є особливо складним через необхідність спільного функціонування комп'ютерного трафіку з чутливим до затримок мультимедійним трафіком (IP-телефонія, відеоконференції), що вимагає налаштування складних механізмів якості обслуговування (QoS).
- Управління безпекою (Security Management): Контроль доступу до ресурсів та захист інфраструктури від несанкціонованого доступу та саботажу. Управління безпекою включає реалізацію політик за моделлю AAA (Authentication, Authorization, Accounting) та розмежування прав користувачів, часто із використанням спеціалізованих служб каталогів.

Незважаючи на існування стандарту FCAPS, впровадження систем управління стикається зі специфічними проблемами адміністрування на практиці. Першою глобальною проблемою є людський фактор при налаштуванні обладнання через інтерфейси командного рядка (CLI). У корпоративних мережах адміністратори щотижня витрачають близько 12,3 годин виключно на усунення помилок конфігурації, що призводить до понад 640 годин незапланованих ремонтних робіт на рік. Складність полягає в тому, що мережеві інженери змушені володіти кількома різними мовами конфігурації для управління обладнанням різних виробників. Відсутність автоматизації призводить до того, що прості зміни у політиках безпеки або списках контролю доступу (ACL) супроводжуються рівнем помилок до 9,7%, кожна з яких може зупинити роботу бізнесу.

Другою вагомою проблемою є складнощі масштабування служб каталогів та ідентифікації користувачів. Для централізованого управління автентифікацією компанії використовують ієрархічні системи на базі протоколу LDAP, найяскравішим представником яких є Microsoft Active Directory (AD). Незважаючи на швидкість пошуку та організації ресурсів, централізовані системи керування

ідентифікацією (CIMS) створюють єдину точку відмови (Single Point of Failure – SPOF)[5]. Якщо контролер домену стає недоступним через мережеві збої або атаки, вся система блокується, і користувачі втрачають доступ до корпоративних сервісів. Крім того, зі зростанням масштабів мережі та збільшенням кількості користувачів навантаження на систему авторизації експоненціально зростає, що призводить до затримок (performance degradation) при обробці запитів до LDAP-каталогів. Також серйозним викликом безпеці є те, що за замовчуванням LDAP передає облікові дані у відкритому вигляді (plaintext), вимагаючи обов'язкового складного налаштування SSL/TLS-шифрування.

Третьою проблемою виступають труднощі обробки величезної кількості логів (системних журналів) та телеметрії. Збільшення кількості вузлів генерує нескінченний потік подій (traps, syslog). Якщо адміністрування не автоматизоване, інженерам доводиться працювати з розрізненими масивами інформації. Для вирішення цієї проблеми доводиться застосовувати системи класу Manager of Managers (MOM), які здатні проводити кореляцію, пріоритезацію та придушення (suppression) незначних подій, щоб технічний персонал мережевого операційного центру (NOC) міг зосередитись на критичних аваріях. Однак інтеграція таких систем з інфраструктурою різних поколінь (legacy devices) залишається надскладним завданням, для якого підприємствам часто бракує кваліфікованих спеціалістів.

1.2 Огляд програмних рішень для управління мережевим обладнанням

Сучасна архітектура корпоративних комп'ютерних мереж еволюціонувала від статичних структур до високодинамічних і складних гетерогенних середовищ. Традиційні методи ручного керування обладнанням через інтерфейс командного рядка (CLI) на кожному окремому пристрої не лише знижують операційну ефективність, а й створюють критичні ризики для інформаційної безпеки через людський фактор. Для вирішення цих проблем застосовується спеціалізоване програмне забезпечення класу NCM (Network Configuration Management — управління конфігураціями мережі), що дозволяє повністю автоматизувати та взяти під контроль весь життєвий цикл конфігурацій пристроїв.

Сучасний ринок пропонує широкий спектр NCM-рішень: від потужних комерційних платформ корпоративного рівня до спеціалізованих утиліт з відкритим вихідним кодом (Open Source) та інструментів парадигми «Network-as-Code» (мережа як код). Для об'єктивного вибору оптимальної системи необхідно детально проаналізувати їхні архітектурні підходи та функціональні можливості.

Комерційні рішення традиційно орієнтовані на надання цілісного функціоналу «з коробки», який інтегрується в ширші екосистеми моніторингу та управління IT-послугами. Вони пропонують високий рівень абстракції для управління мультибрендовим обладнанням через єдиний графічний інтерфейс.

SolarWinds Network Configuration Manager (NCM) є одним із найбільш поширених корпоративних рішень[6]. Його архітектура тісно інтегрована з платформою SolarWinds Platform (раніше Orion), що забезпечує безшовну взаємодію з модулями продуктивності (NPM) та аналізаторами трафіку (NTA). Система підтримує протоколи SSH, Telnet, TFTP та HTTPS для доступу та передачі конфігурацій з обладнання широкого спектра виробників (Cisco, Juniper, HP, Dell). Важливою технічною перевагою SolarWinds NCM є функція NetPath, яка дозволяє візуально корелювати на часовій шкалі зміни в конфігурації пристроїв із проблемами продуктивності, що значно прискорює пошук кореневих причин інцидентів (root cause analysis). Крім того, платформа здійснює аудит відповідності нормативним стандартам (HIPAA, PCI DSS) та має модуль оцінки вразливостей (Vulnerability Assessment), що ідентифікує пристрої з відомими CVE. Однак, перехід SolarWinds на модель підписки призвів до зростання сукупної вартості володіння (TCO), а сама система, за відгуками інженерів, є досить вимогливою до апаратних ресурсів серверів баз даних (SQL Server).

ManageEngine Network Configuration Manager пропонує альтернативний підхід, роблячи акцент на максимальній мультибрендовій універсальності, підтримуючи пристрої від понад 200 виробників[3]. Ключовим інструментом автоматизації тут виступають «конфіглети» (Configlets) — шаблони сценаріїв для масового виконання операцій, таких як зміна паролів або оновлення списків контролю доступу (ACL) на сотнях пристроїв одночасно. Всі вилучені конфігурації

зберігаються у базі даних із застосуванням надійного 256-бітного AES-шифрування. Для великих розподілених мереж версія Enterprise використовує архітектуру «Central-Probe» (Центральний сервер – Зонд), яка дозволяє збирати дані з віддалених філій із подальшою автоматичною синхронізацією даних на центральному сервері. Значною перевагою є модуль Firmware Vulnerability Management, який безперервно отримує оновлення бази даних NIST і формує списки вразливого обладнання із зазначенням CVE ID, рівня критичності та посилань на відповідні патчі від виробників.

Cisco Catalyst Center (раніше DNA Center) представляє клас рішень нового покоління, заснованих на концепції Intent-Based Networking (мережі на основі намірів)[7]. Ця система використовує штучний інтелект та машинне навчання для автоматизації проектування та забезпечення стабільної роботи мережі. Catalyst Center розмежовує дані на дві категорії: *Automation data* (файли конфігурацій, облікові дані, які передаються завжди повними бекапами через Rsync по порту 22) та *Assurance data* (мережева аналітика та телеметрія, що вимагає розгортання NFS-сервера версій v3/v4 для інкрементального збереження). Незважаючи на фокус на обладнанні Cisco, Catalyst Center здатний керувати пристроями сторонніх виробників за допомогою протоколів MIB2/SNMP.

Для багатьох IT-інфраструктур комерційні платформи є занадто дорогими або надлишковими. У таких випадках широко застосовується відкрите ПЗ, що надає інструментарій для створення гнучких пайплайнів управління.

RANCID (Really Awesome New Cisco conflg Differ) історично вважається золотим стандартом автоматизованого резервного копіювання[8]. Його архітектура працює за простим алгоритмом: система зчитує перелік пристроїв з файлу router.db, підключається до них за допомогою скриптів на мові Expect (наприклад, clogin), виконує необхідні команди, відфільтровує змінну інформацію (час роботи, лічильники інтерфейсів) та зберігає конфігурацію в систему контролю версій (CVS, Subversion або Git). При кожній зміні RANCID автоматично генерує diff-файли та розсилає їх електронною поштою, що стимулює адміністраторів дотримуватися

дисципліни. Недоліком системи є її застаріла кодова база (Perl/Expect), що ускладнює додавання підтримки сучасних пристроїв.

Oxidized був створений як сучасна заміна RANCID. Написаний на мові Ruby, він має багатопотокову архітектуру, що дозволяє значно швидше опитувати великі парки обладнання[9]. Платформа підтримує понад 130 типів операційних систем і має повноцінний REST API, що дозволяє миттєво перемістити пристрій у початок черги на бекап (GET/PUT запити). Oxidized вміє перехоплювати події Syslog про зміни в конфігурації (на IOS або JunOS) та автоматично запускати позачергове збереження, використовуючи механізм git blame для фіксації автора змін. Ключовою перевагою Oxidized є глибока інтеграція з системами моніторингу (наприклад, LibreNMS) та можливість використання NetBox через HTTP API як єдиного джерела істини (Source of Truth) для динамічного оновлення переліку вузлів.

Unimus виступає як проміжна ланка між надскладними NCM-комбайнами та скриптовими утилітами[10]. Це On-Premise рішення, орієнтоване на швидке розгортання та гарантоване відновлення після збоїв (Disaster Recovery). Система є без-агентною (agentless), не вимагає від адміністратора навичок програмування та підтримує понад 450 типів пристроїв від 160 вендорів (Cisco, MikroTik, Juniper тощо). На відміну від RANCID, Unimus самостійно аналізує та визначає тип обладнання й версію ОС під час підключення, автоматично застосовуючи відповідні алгоритми для збереження налаштувань у зашифровану базу даних, що мінімізує витрати часу на конфігурацію.

У сучасних хмарних інфраструктурах активно застосовується декларативний підхід до управління. Такі системи, як Ansible, дозволяють інженерам описувати бажаний стан мережі у легкочитних YAML-файлах (Playbooks)[11]. Ansible використовує модулі (наприклад, ios_command або junos_config), інвентар пристроїв (Inventory) та механізми шаблонізації Jinja2 для автоматичної генерації та застосування конфігурацій, повністю спираючись на дані з NetBox. Додатково використовується бібліотека NAPALM (Network Automation and Programmability Abstraction Layer), розроблена на Python[12]. Вона надає єдиний уніфікований API

для мультивендорних середовищ (Cisco IOS, Arista EOS, Juniper) та підтримує надзвичайно важливі механізми validate (перевірка поточного стану на відповідність вимогам), commit confirm та rollback (відкат змін у разі невдалої транзакції).

1.3 Аналіз систем централізованого управління користувачами та ресурсами

Беззаперечним лідером та галузевим стандартом у сфері локальних (on-premises) систем централізованого управління історично є **Microsoft Active Directory** (AD). Ця служба каталогів була розроблена для середовищ Windows і базується на клієнт-серверній архітектурі. Фундаментальна робота Active Directory Domain Services (AD DS) спирається на три базові мережеві стандарти: протокол LDAP (Lightweight Directory Access Protocol) для запитів до каталогу, протокол Kerberos для безпечної автентифікації та службу DNS (Domain Name System)[13] для локалізації контролерів домену в мережі.

Логічна структура Active Directory є строго ієрархічною і поділяється на ліси (Forests), дерева (Trees), домени (Domains) та організаційні одиниці (OU). Технічні спеціалісти часто помилково вважають домен межею безпеки, проте саме ліс є фактичною і найвищою межею безпеки всієї системи. Адміністратор рівня підприємства (Enterprise Admin) у кореневому домені має технічну можливість скомпрометувати будь-який інший домен у межах одного лісу. Кожен домен зберігає інформацію про свої об'єкти у базі даних (NTDS.dit), яка реплікується між контролерами домену за моделлю multi-master. Для того, щоб користувачі могли швидко знаходити ресурси в інших доменах великого лісу, використовується Глобальний каталог (Global Catalog) — спеціалізований контролер домену, що зберігає повну копію об'єктів свого домену та часткову (Partial Attribute Set) копію об'єктів усіх інших доменів.

Ключовим інструментом централізованого адміністрування в AD є групові політики (Group Policy Objects, GPO). Вони дозволяють масово застосовувати конфігурації ОС, параметри безпеки, правила фаєрволу та обмеження доступу на тисячах пристроїв одночасно. Політики застосовуються за строгим ієрархічним правилом LSDOU (Local, Site, Domain, OU), де конфігурації нижчого рівня мають

пріоритет над вищими. Хоча GPO є надзвичайно потужним інструментом, вони мають суттєві обмеження: відсутність вбудованих інструментів звітності щодо успішності застосування на клієнтах, а також майже повна відсутність нативної підтримки управління пристроями на базі Linux чи macOS.

Зростання гетерогенності мереж (масове використання Unix/Linux систем) та необхідність уникнення ліцензійних витрат на Microsoft Windows Server призвели до розвитку відкритих альтернатив (Open Source) для служб каталогів:

- OpenLDAP: Це чистий, вендорно-незалежний сервер каталогів, що реалізує протокол LDAP[14]. LDAP ефективно передає лише необхідні дані і споживає мінімум системних ресурсів. На відміну від AD, OpenLDAP є надзвичайно гнучким і кросплатформним, але він вимагає високої інженерної кваліфікації для налаштування схем та не має вбудованих засобів управління пристроями чи розвинених механізмів авторизації "з коробки". Крім того, за замовчуванням LDAP передає облікові дані у відкритому вигляді, що вимагає обов'язкового налаштування SSL/TLS (LDAPS) для безпеки.
- FreeIPA: Інтегроване рішення, яке є фактичним аналогом Active Directory для Linux-орієнтованих інфраструктур[14]. Ця система поєднує в собі 389 Directory Server (для LDAP), MIT Kerberos (для автентифікації та SSO), BIND (для DNS) та систему управління сертифікатами (Dogtag PKI). Замість GPO, FreeIPA використовує механізми Host-Based Access Control (HBAC) для керування доступом на рівні sudo та SSH-ключів.
- Samba 4 (Samba AD DC): Революційне рішення з відкритим кодом, що дозволяє Linux-серверу виконувати роль повноцінного контролера домену Active Directory[15]. Вона підтримує ті ж самі протоколи і може безперешкодно керуватися стандартними утилітами Windows (RSAT). Проте система має певні технічні обмеження порівняно з оригінальною AD, зокрема у підтримці реплікації Sysvol через сучасні механізми DFS-R.

Останнім стратегічним зрушенням в управлінні ресурсами є масова міграція до хмарних служб каталогів та рішень класу Identity-as-a-Service (IDaaS). Цей перехід

зумовлений тим, що сучасний периметр безпеки змістився від фізичної локальної мережі до безпосередньо самої "ідентичності" користувача.

У цьому сегменті виділяється **Microsoft Entra ID** (раніше Azure AD). Це не просто хмарна копія AD, а принципово інша платформа, орієнтована на веб-стандарти (SAML, OAuth 2.0, OpenID Connect) замість класичного Kerberos. Entra ID забезпечує глибоку реалізацію парадигми Zero Trust (Нульова довіра)[16] завдяки політикам умовного доступу (Conditional Access), які в режимі реального часу аналізують ризики сесії, локацію користувача та комплаєнс пристрою через інтеграцію з Microsoft Intune.

Серед незалежних IDaaS-платформ домінують **Okta** та **JumpCloud**[17]**.** Okta фокусується на корпоративному управлінні доступом (Single Sign-On) та має понад 7000 попередньо налаштованих інтеграцій (OIN) для управління життєвим циклом користувачів у SaaS-додатках. Проте Okta не має повноцінного вбудованого функціоналу для глибокого управління самими кінцевими пристроями (MDM). JumpCloud, навпаки, позиціонується як уніфікована хмарна директорія, що поєднує службу каталогів (Cloud LDAP, Cloud RADIUS) та управління мобільними пристроями (MDM/UEM) для Windows, macOS і Linux в єдиній консолі, стаючи ідеальною заміною застарілої локальної AD для малого та середнього бізнесу.

1.4 Системи моніторингу стану інфраструктури комплексного управління

Безперервний моніторинг стану корпоративної комп'ютерної мережі та серверної інфраструктури є фундаментальною складовою стратегії управління IT-послугами (ITSM) та забезпечення кібербезпеки. У контексті еталонної моделі управління FCAPS, системи моніторингу безпосередньо забезпечують реалізацію функцій управління збоями (Fault Management) та управління продуктивністю (Performance Management). Зростаюча складність гетерогенних мереж, міграція до гібридних хмарних середовищ та впровадження архітектур мікросервісів роблять ручний контроль неможливим, перетворюючи платформи моніторингу на критично важливі аналітичні центри підприємства. Відсутність прозорості та запізніле реагування на інциденти коштують бізнесу мільярди доларів щорічно у

вигляді втраченого часу на пошук несправностей, що міг би бути витрачений на інновації.

Основними цілями впровадження інфраструктурного моніторингу є забезпечення оптимального рівня продуктивності, проактивне виявлення несправностей до моменту їх впливу на кінцевих користувачів, планування потужностей шляхом відстеження утилізації ресурсів та моніторинг безпеки для реагування на загрози в режимі реального часу. Сучасний підхід до управління вимагає переходу від реактивної моделі (реагування за фактом падіння сервісу) до проактивного моніторингу. Проактивний підхід передбачає безперервне оцінювання систем за допомогою предиктивної аналітики, що базується на історичних даних, і формування ранніх попереджень про потенційні загрози. Для цього системи формують так звані «базові лінії» (baselines) нормальної поведінки інфраструктури, фіксуючи типове завантаження процесорів, використання пам'яті, дисковий ввід/вивід (I/O) та пропускну здатність мережі.

Стратегічна цінність платформ моніторингу також яскраво проявляється в процесах планування безперервності бізнесу (Business Continuity Planning, BCP) та аварійного відновлення (Disaster Recovery, DR). У разі виникнення збоїв системи моніторингу надають інформацію в режимі реального часу для швидкого сортування (тріажу) інцидентів, локалізуючи кореневу причину деградації,. Це критично зменшує середній час до відновлення (Mean Time To Resolution, MTTR) та дозволяє IT-відділам валідувати досягнення цільових показників точки відновлення (RPO) і часу відновлення (RTO).

На сучасному ринку програмного забезпечення для моніторингу виділяються два концептуально різні підходи: відкриті (Open Source) платформи високої масштабованості та комерційні рішення формату «все-в-одному» (All-in-One). Найбільш репрезентативними представниками цих підходів є системи Zabbix та Paessler PRTG Network Monitor.

Zabbix є відкритим програмним забезпеченням (розповсюджується за ліцензією AGPL-3.0), яке розроблене для централізованого моніторингу мільйонів метрик з десятків тисяч пристроїв у режимі реального часу. Архітектура Zabbix є

глибоко розподіленою та базується на моделі «Сервер – Проксі – Агент». Він підтримує як агентний моніторинг (з використанням легких демонів для Linux, Windows, macOS, Solaris), так і безагентний збір даних через протоколи SNMP, IPMI, JMX, HTTP та SSH.

З випуском версії Zabbix 7.0 LTS система отримала низку фундаментальних архітектурних оновлень, орієнтованих на підвищення продуктивності[18]. Ключовою інновацією стала підтримка балансування навантаження та високої доступності (High Availability) для Zabbix-проксі[19]. Тепер хости можна призначати групам проксі-серверів, і у разі падіння одного вузла його навантаження миттєво і прозоро перерозподіляється між іншими,. Крім того, самі проксі-сервери отримали можливість працювати в режимах буферизації «Memory» (оперативна пам'ять) та «Hybrid», що прискорює їхню продуктивність у 10-100 разів порівняно з традиційним дисковим кешуванням.

Процес збору даних у Zabbix 7.0 був оптимізований шляхом переходу на асинхронні поллери (asynchronous pollers) для перевірок агентів, SNMP та HTTP. Асинхронна архітектура дозволяє не чекати відповіді від одного пристрою перед запитом до іншого, підтримуючи до 1000 одночасних перевірок на один процес поллера, що радикально знижує навантаження на систему. Для веб-додатків впроваджено функцію синтетичного моніторингу кінцевих користувачів, що дозволяє виконувати складні багатокрокові сценарії в браузері, збирати метрики доступності та автоматично генерувати скріншоти у разі виявлення помилок рендерингу. Також система підвищила стандарти корпоративної безпеки, додавши нативну підтримку багатофакторної автентифікації (MFA) через Time-Based One-Time Password (TOTP) та Duo Universal Prompt.

Еволюційний шлях системи, заявлений у концепціях майбутньої версії 8.0 LTS, передбачає перехід від класичного метричного моніторингу до повноцінної «обсервабільності» (Observability), що базується на інтеграції метрик, логів та трасувань (Traces) за стандартом OpenTelemetry. Очікується впровадження технології Complex Event Processing для розумної кореляції подій, дедуплікації та їх автоматичного закриття на базі JavaScript-сценаріїв, що наближає Zabbix до

систем класу AIOps. Проте, незважаючи на відсутність ліцензійних платежів, головним недоліком Zabbix залишається високий поріг входження, технічна складність конфігурування за допомогою консолі Linux та значні сукупні витрати на володіння (TCO) через потребу у залученні висококваліфікованих інженерів для адміністрування.

PRTG Network Monitor від компанії Paessler є комерційною альтернативою, що базується на архітектурі «все-в-одному» та жорстко прив'язана до середовища операційної системи Windows. На відміну від розподіленої архітектури Zabbix, PRTG централізує ядро системи, веб-інтерфейс та базу даних на одному сервері, хоча й підтримує використання віддалених зондів (Remote Probes) для збору інформації з ізольованих сегментів мережі.

Фундаментальною концепцією PRTG є використання «сенсорів». Сенсор — це базовий елемент моніторингу, який вимірює одне конкретне значення на пристрої, наприклад, завантаження центрального процесора, обсяг вільного дискового простору, або трафік на конкретному порту комутатора. Для повноцінного моніторингу одного мережевого вузла зазвичай потрібно від 5 до 10 сенсорів. PRTG постачається з більш ніж 250 типами попередньо налаштованих сенсорів (наприклад, Ping v2, HTTP Full Web Page, SNMP Uptime v2, WMI UTC Time), що покривають більшість потреб для моніторингу мережевого трафіку, баз даних, віртуальних машин (VMware, Hyper-V) та хмарних сервісів[20].

Головною перевагою PRTG є його орієнтація на безагентний моніторинг із використанням вбудованих інструментаріїв операційних систем (WMI, SNMP) та протоколів аналізу потоків даних (NetFlow, sFlow, jFlow, IPFIX). Система має потужний механізм автоматичного виявлення мережі (Auto-Discovery), який сканує заданий діапазон IP-адрес і самостійно застосовує відповідні шаблони сенсорів до знайдених пристроїв, що дозволяє розгорнути базовий моніторинг за лічені хвилини. Інтерфейс користувача підтримує створення інтерактивних карт мережі (Custom Maps) та дашбордів за допомогою зручного редактора drag-and-drop, що робить продукт дуже привабливим для невеликих IT-команд.

Водночас, комерційна модель ліцензування PRTG має свої обмеження. Хоча компанія надає повністю безкоштовну версію (Freeware Edition) з обмеженням до 100 сенсорів, масштабування моніторингу для потреб великого корпоративного середовища вимагає придбання дорогих ліцензій. Крім того, закрита Windows-архітектура та менша гнучкість у написанні складних нестандартних скриптів обмежують можливості системи в високодинамічних контейнеризованих або DevOps-орієнтованих середовищах порівняно із Zabbix.

Сучасна система моніторингу не може існувати ізольовано; вона повинна функціонувати в симбіозі з платформами управління IT-послугами (ITSM), такими як ServiceNow або Jira Service Management. Впровадження інтеграції між моніторингом та ITSM забезпечує реалізацію концепції e-Bonding — двосторонньої синхронізації даних. Завдяки вебхукам та REST API, коли система моніторингу фіксує критичний збій, вона не просто відправляє email, а автоматично генерує інцидент у Service Desk із заповненням усіх необхідних технічних полів (ідентифікатор вузла, рівень критичності, телеметрія). Якщо системний адміністратор визнає інцидент у Jira, ця дія синхронізується у зворотному напрямку, і тригер у системі Zabbix автоматично отримує статус підтвердженого (Acknowledged). У випадку з ServiceNow, інтеграція PRTG відбувається через механізм Event Field Mapping, де параметри сенсора мапуються безпосередньо у таблицю подій [em_event] платформи управління, забезпечуючи автоматизоване формування інцидентів. Такий підхід усуває ручне дублювання даних, зменшує рівень помилок і створює прозорий конвеєр реагування на технічні проблеми.

Висновки до Розділу 1

У першому розділі було здійснено комплексний аналіз теоретичних засад та практичних аспектів управління корпоративними комп’ютерними мережами. Дослідження довело, що в сучасних умовах цифровізації успішна діяльність організації критично залежить від стабільності інформаційного простору, який характеризується високим рівнем гетерогенності, складності та постійним динамічним розвитком. Встановлено, що традиційні методи ручного адміністрування мережевої інфраструктури стають неефективними, оскільки

людський фактор є першопричиною переважної більшості інцидентів та мережевих збоїв. Відтак, обґрунтовано нагальну потребу переходу до автоматизованих систем управління, побудованих на базі міжнародних стандартів, зокрема еталонної моделі FCAPS, яка охоплює всі ключові вектори — від конфігураційного управління до моніторингу безпеки та продуктивності.

Детальний огляд програмних рішень, що включає інструменти класу NCM, сучасні служби каталогів та комплексні платформи інфраструктурного моніторингу (такі як Zabbix, PRTG та Microsoft Active Directory), продемонстрував значну архітектурну різноманітність на ринку. Аналіз виявив, що IT-фахівці стикаються з критичним вибором між масштабованими відкритими системами, які вимагають глибокої інженерної експертизи для налаштування, та комерційними продуктами, які забезпечують швидке розгортання, проте супроводжуються високою сукупною вартістю володіння. Крім того, окрему увагу було приділено проблемам відмовостійкості, зокрема подоланню вразливостей єдиної точки відмови в розподілених мережевих середовищах.

Підсумовуючи, можна стверджувати, що незважаючи на широку доступність технологічних рішень, галузь відчуває дефіцит об'єктивних інструментів для порівняльного аналізу та підбору ПЗ. Адміністратори часто опиняються в ситуації, коли рішення про вибір компонентів мережі приймаються на основі несистематизованих відгуків або суб’єктивних оцінок, що призводить до неефективного розподілу ресурсів та помилок при розгортанні систем. Виявлена відсутність уніфікованого методологічного підходу вказує на актуальність та своєчасність розробки об'єктивного математичного інструментарію, який би дозволив перевести процес прийняття управлінських рішень в IT-сфері на рівень строгих кількісних розрахунків. Це стає базовою передумовою для подальшого проектування інформаційної системи оцінювання, що буде представлена у наступних розділах роботи.

2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ОЦІНКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз вимог до системи та визначення ролей

Головною метою створення даної інформаційної системи є автоматизація процесу об'єктивного вибору та оцінки мережевого програмного забезпечення (систем моніторингу, NCM, служб каталогів тощо). Програма розробляється як спеціалізована система підтримки прийняття рішень (СППР), яка математично формалізує процес вибору, спираючись на метод аналізу ієрархій (MAI)[21]. Це дозволить нівелювати суб'єктивність, знайти ідеальний баланс між вартістю, надійністю та функціональністю для конкретної IT-інфраструктури, та забезпечити керівництво математично обґрунтованими рекомендаціями.

Для забезпечення коректної роботи алгоритмів оцінювання та задоволення потреб кінцевих користувачів, архітектура системи повинна відповідати чітко регламентованим вимогам.

Функціональні вимоги (описують безпосередню поведінку системи та процеси обробки даних):

- **Управління базою альтернатив:** Система повинна дозволяти додавати, редагувати та видаляти інформацію про програмні продукти (альтернативи), які підлягають порівнянню (наприклад, SolarWinds, Zabbix, PRTG, Ansible).
- **Побудова ієрархічної структури:** Система має забезпечувати можливість створення багаторівневої моделі критеріїв (мета → критерії → підкритерії → альтернативи), яка є основою методу MAI.
- **Введення експертних оцінок:** Система повинна надавати інтерфейс для заповнення матриць попарних порівнянь критеріїв та альтернатив з використанням 9-бальної шкали відношень Сааті.
- **Математична обробка та ієрархічний синтез:** Система повинна автоматично розраховувати вектори локальних та глобальних пріоритетів на основі введених матриць парних порівнянь.
- **Контроль логіки експерта:** Система має автоматично розраховувати відношення узгодженості (ВУ) для кожної матриці. Якщо значення ВУ

перевищує допустимий поріг (наприклад, 0,1), система повинна сигналізувати про необхідність перегляду оцінок.

- **Генерація звітів:** Система повинна формувати підсумковий рейтинг програмного забезпечення, ранжуючи альтернативи за зростанням глобального пріоритету, та візуалізувати результати у вигляді графіків і звітів.

Нефункціональні вимоги (описують атрибути якості системи, такі як безпека, надійність та швидкодія):

- **Безпека та розмежування доступу:** Система повинна підтримувати дозволи на основі ролей (RBAC — Role-Based Access Control). Необхідно жорстко розмежувати права перегляду, редагування матриць та управління системою, а також вести журнал (аудит) дозволів для запобігання неавторизованим змінам.
- **Швидкодія та продуктивність:** Оскільки обчислення векторів пріоритетів та власних чисел матриць вимагає математичних ітерацій, система повинна виконувати розрахунки матриць розмірністю до 10×10 у режимі реального часу (затримка не більше 1-2 секунд) для забезпечення комфортної роботи експерта.
- **Надійність та відмовостійкість:** Система не повинна мати єдиної точки відмови (SPOF). Усі введені експертні оцінки та матриці повинні автоматично зберігатися в базі даних із регулярним резервним копіюванням, щоб уникнути втрати даних експертизи при збоях сервера.
- **Ергономічність (Usability):** Інтерфейс матриць попарних порівнянь має бути інтуїтивно зрозумілим, щоб експерти могли зосередитися на оцінюванні предметної області (мережевого ПЗ), а не на вивченні інструкцій до самої системи підтримки прийняття рішень.

У контексті об'єктно-орієнтованого моделювання учасники процесу взаємодії з системою визначаються як «Актори» (Actors) — сутності, що знаходяться за межами системи, але безпосередньо з нею взаємодіють. Відповідно до методології теорії прийняття рішень та вимог безпеки, в системі виокремлено три ключові ролі:

«Системний адміністратор», «Експерт» та «Користувач» (Особа, що приймає рішення).

Роль: «Системний адміністратор»

- **Опис ролі:** Технічний спеціаліст, який відповідає за працездатність, конфігурацію та безпеку самої інформаційної системи. Він не бере участі в оцінюванні мережевого ПЗ, а лише забезпечує життєвий цикл програмного середовища.
- **Завдання в системі:** Реєстрація нових облікових записів для Експертів та Користувачів; налаштування параметрів бази даних; моніторинг журналів аудиту та безпеки (перевірка логів дозволів); забезпечення резервного копіювання даних системи.
- **Права доступу:** Має повний (адміністративний) доступ до бекенду системи та управління користувачами. Проте, з метою збереження об'єктивності дослідження, системний адміністратор **не має прав** на модифікацію вже внесених експертних оцінок або зміну розрахованих математичних вагових коефіцієнтів.

Роль: «Експерт»

- **Опис ролі:** Фахівець-професіонал у галузі мережевих технологій (наприклад, мережевий інженер), до якого звертаються за оцінками і рекомендаціями. Експерт володіє глибокими знаннями про архітектуру мережевого ПЗ (NCM, NMS), розуміє специфіку CLI, SNMP, LDAP та протоколів безпеки.
- **Завдання в системі:** Інтуїтивно-логічний аналіз проблеми; формування множини критеріїв оцінки (наприклад, ТСО, надійність, підтримка мультибрендовості); внесення програмних альтернатив до бази; заповнення матриць попарних порівнянь Сааті. Експерт відповідає за генерацію вихідних даних для математичної моделі.
- **Права доступу:** Має права на створення та редагування ієрархічних моделей, додавання альтернатив та критеріїв, а також на внесення і редагування

власних експертних оцінок (матриць). Не має доступу до адміністрування системи чи управління профілями інших експертів.

Роль: «Користувач» (Особа, що приймає рішення - ОПР)

- **Опис ролі:** Керівник ІТ-відділу, технічний директор (CTO) або бізнес-аналітик, для якого, власне, і здійснюється вибір найкращої альтернативи. Ця особа має необхідні повноваження для закупівлі мережевого ПЗ та несе відповідальність за прийняте стратегічне рішення.
- **Завдання в системі:** Ініціація процесу вибору; задання глобальних пріоритетів та обмежень (наприклад, жорсткий ліміт бюджету або вимога обов'язкової підтримки Cisco IOS); перегляд згенерованих системою звітів, графіків та підсумкового рейтингу; аналіз чутливості (стійкості) моделі для фінального вибору. Користувач не формує самі оцінки, а використовує готовий синтезований результат.
- **Права доступу:** Має права доступу виключно на читання (Read-only) до матриць попарних порівнянь, введених експертами. Має повний доступ до модуля генерації звітів, експорту аналітичних даних та перегляду підсумкових глобальних векторів пріоритетів.

Впровадження даної рольової моделі у поєднанні з математичним апаратом MAI забезпечить створення надійної, прозорої та функціональної системи для автоматизації вибору програмного забезпечення управління корпоративними мережами.

2.2 Моделювання архітектури системи

2.2.1 Діаграма прецедентів для процесу оцінювання та модерації відгуків

Відповідно до методології об'єктно-орієнтованого моделювання програмних систем, розробка архітектури системи починається з побудови діаграми прецедентів (Use Case Diagram)[22](Рис. 2.1) Ця діаграма дозволяє уявити типи ролей та їхню взаємодію із системою, а також сформувати функціональні вимоги до програмного продукту з точки зору кінцевого користувача. Діаграма прецедентів описує окремі аспекти поведінки системи (що саме вона робить), абстрагуючись від деталей її технічної реалізації.

У системі підтримки прийняття рішень для оцінки мережевого програмного забезпечення виділено трьох зовнішніх учасників — акторів, які взаємодіють із системою для досягнення своїх специфічних цілей:

Системний адміністратор — користувач, який виконує службові функції з підтримки життєдіяльності системи, управління правами доступу та модерації.

Експерт — фахівець-професіонал у галузі управління мережевою інфраструктурою, до якого звертаються за оцінками. Його головна взаємодія полягає в наповненні системи даними та формуванні експертних оцінок.

Користувач (Особа, що приймає рішення - ОПР) — клієнт системи (наприклад, керівник ІТ-відділу), який має необхідні повноваження для закупівлі ПЗ та використовує систему для вибору найкращої альтернативи.

Детальний опис прецедентів системи

- **Авторизація:** Базовий прецедент, що забезпечує процес ідентифікації та автентифікації користувачів для доступу до закритої частини системи. Оскільки система реалізує розмежування прав на основі ролей (RBAC), авторизація є обов'язковим етапом для кожного з трьох акторів перед виконанням їхніх специфічних завдань.
- **Додавання ПЗ для порівняння:** Прецедент ініціюється актором «Експерт». Він полягає у внесенні до бази даних системи інформації про нові програмні рішення (наприклад, інструменти моніторингу Zabbix, PRTG або системи управління конфігураціями). Експерт створює множину альтернатив, які в подальшому будуть оцінюватися.
- **Внесення експертних оцінок:** Ключовий прецедент, який описує взаємодію «Експерта» з математичним ядром системи на основі методу аналізу ієрархій (MAI). У межах цього прецеденту експерт заповнює множину матриць попарних порівнянь для критеріїв та альтернатив, виражаючи свої судження в цілих числах за дев'ятибальною шкалою Сааті.
- **Перегляд рейтингу:** Прецедент, з яким переважно взаємодіє «Користувач» (ОПР). Після того як система автоматично розраховує вектори локальних і глобальних пріоритетів, користувач ініціює запит на генерацію звіту.

Система візуалізує підсумковий комплексний рейтинг програмного забезпечення, дозволяючи Користувачу зробити математично обґрунтований вибір.

- **Додавання відгуку:** Прецедент, який дозволяє «Користувачу» (або «Експерту») залишати текстові коментарі чи практичні зауваження щодо досвіду експлуатації певного програмного засобу. Ця функція додає системі якісний вимір оцінювання, який доповнює строгі математичні розрахунки MAI.

Рисунок 2.1 - Діаграма прецедентів

2.2.2 Діаграма діяльності та послідовностей системи управління вибором

Для детального моделювання поведінки системи управління вибором мережевого програмного забезпечення використовуються діаграми діяльності та послідовностей. Діаграма діяльності (Рис. 2.2) описує динамічні аспекти системи у вигляді блок-схеми, яка відображає логіку процедур і потоки робіт — переходи від однієї діяльності до іншої. Своєю чергою, діаграма послідовностей (Рис.2.3)

відображає часові особливості взаємодії об'єктів шляхом передачі та прийому повідомлень у межах конкретного сценарію.

Алгоритм роботи Експерта у вигляді діаграми діяльності

Текстовий алгоритм покрокового внесення оцінок актором «Експерт» для методу аналізу ієрархій виглядає наступним чином:

Початковий стан: Експерт успішно проходить процес авторизації у системі.

Вибір елементів для оцінки: Експерт обирає з бази даних критерій або набір альтернатив (програмних продуктів), для яких необхідно провести оцінювання.

Внесення оцінок: Експерт заповнює матрицю попарних порівнянь, використовуючи 9-бальну шкалу відношень Сааті, де 1 означає рівну значущість, а 9 — абсолютну перевагу однієї альтернативи над іншою.

Математична перевірка (Вузол прийняття рішень): Після введення даних система автоматично розраховує відношення узгодженості (ВУ) для заповненої матриці. На цьому етапі потік розгалужується:

- *Умова 1 (Неузгодженість):* Якщо ВУ перевищує допустимий поріг (зазвичай 0,10), система видає попередження про порушення логіки експерта. Потік повертається на крок редагування матриці.
- *Умова 2 (Узгодженість):* Якщо ВУ $\leq 0{,}10$, матриця вважається коректною і дані зберігаються у базі даних.

Цикл оцінювання: Система перевіряє, чи заповнені всі необхідні матриці. Якщо ні — експерт повертається до вибору наступного критерію. Якщо так — процес експертизи успішно завершується (кінцевий стан).

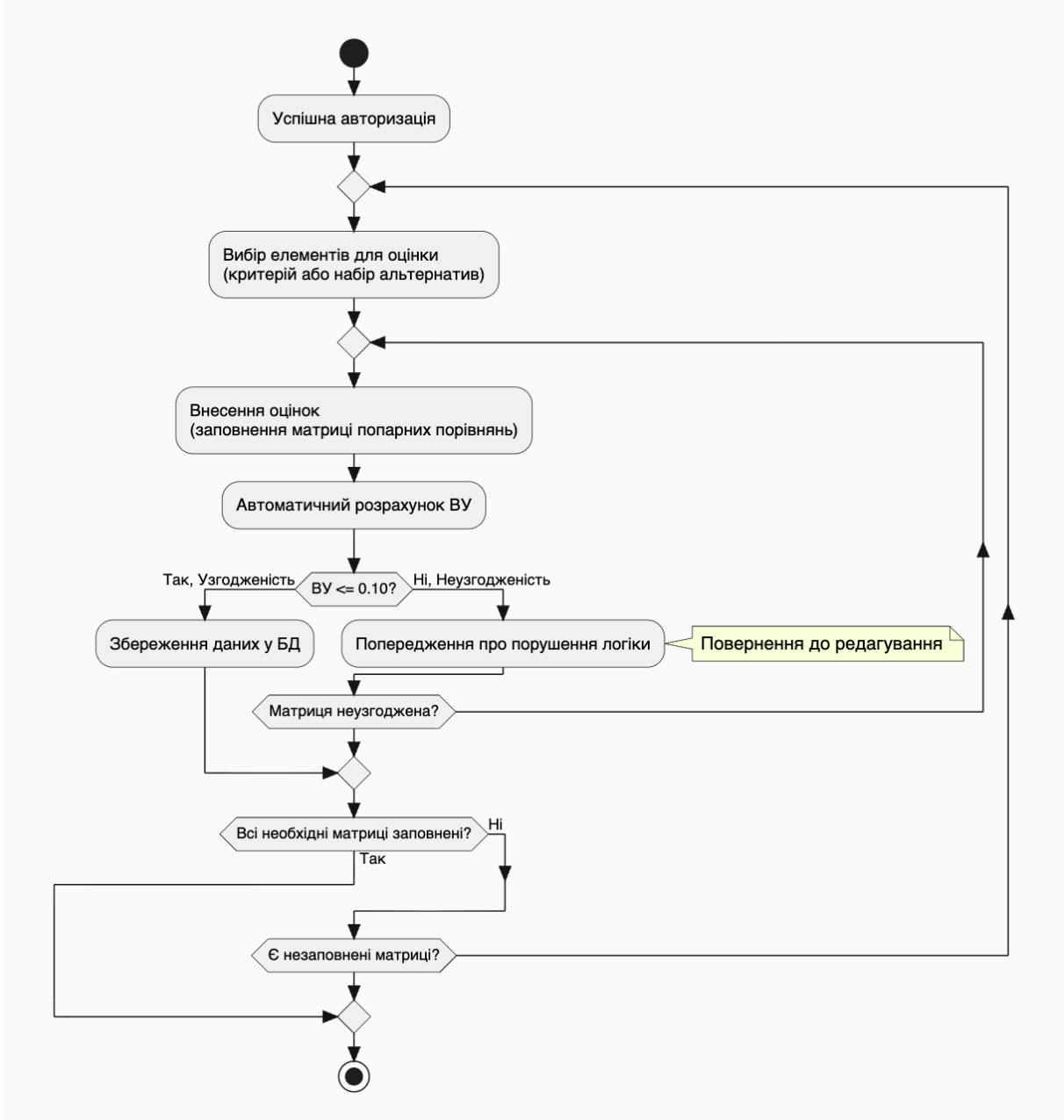


Рисунок 2.2 - Діаграма діяльності покрокового внесення оцінок експертом

Життєвий цикл запиту Користувача у вигляді діаграми послідовностей

Взаємодія Користувача (Особи, що приймає рішення - ОПР) із системою для отримання підсумкового рейтингу відображає структурну організацію об'єктів: Користувач (Актор), Інтерфейс (Межа системи), Контролер розрахунків MAI (Керуючий об'єкт) та База даних (Сутність).

Сценарій (життєвий цикл) відбувається так:

- Користувач ініціює запит на генерацію комплексного рейтингу мережевого ПЗ через Інтерфейс (синхронне повідомлення).
- Інтерфейс активує Контролер розрахунків, передаючи йому команду на формування звіту.
- Контролер розрахунків звертається до Бази даних із запитом на отримання раніше збережених матриць попарних порівнянь та критеріїв.
- База даних повертає необхідний масив експертних оцінок (повідомлення повернення).
- Контролер розрахунків самостійно (через рефлексивні повідомлення) виконує математичну обробку: спочатку розраховує вектори локальних пріоритетів для кожної матриці, а потім здійснює алгоритм ієрархічного синтезу. У результаті ієрархічного синтезу вектори локальних пріоритетів альтернатив зважуються на ваги критеріїв для отримання глобального рейтингу.
- Контролер передає готовий математичний результат (список ПЗ) до Інтерфейсу.
- Інтерфейс візуалізує дані у вигляді зрозумілих графіків та таблиць і повертає кінцевий результат Користувачу.

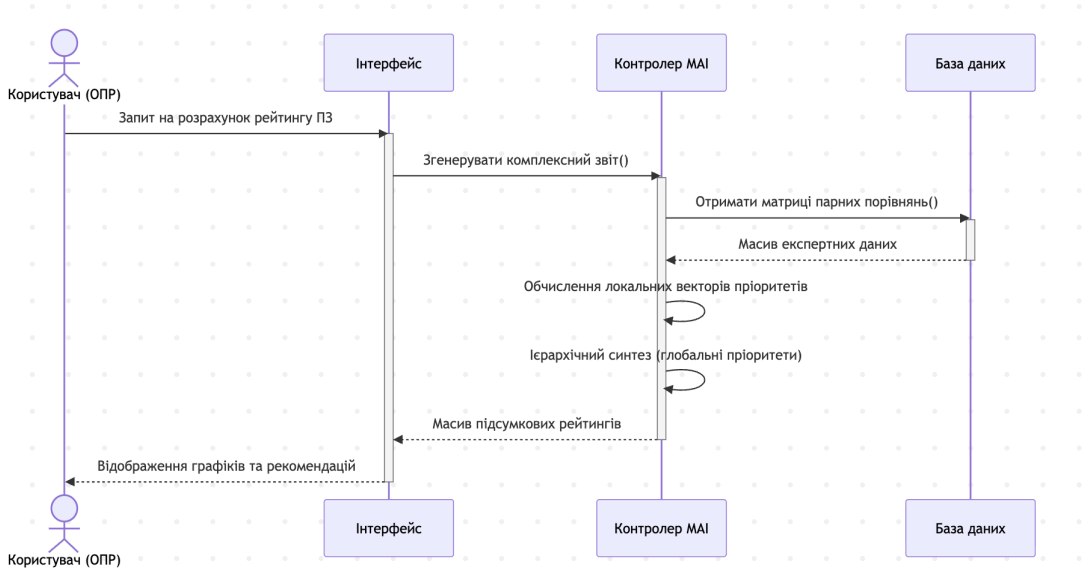


Рисунок 2.3 - Діаграма послідовностей розрахунку рейтингів користувачем

2.2.3 Діаграма класів для прецедентів «Програмний засіб», «Оцінка», «Відгук»

Діаграма класів (Рис.2.4) є фундаментальним інструментом об'єктно-орієнтованого моделювання, що представляє статичну структуру програмної системи. Вона дає найбільш повне і розгорнуте уявлення про класифікацію об'єктів, їхні внутрішні характеристики (атрибути та операції), а також про логічні й фізичні зв'язки між ними. Відповідно до архітектури системи оцінки мережевого ПЗ, статична структура предметної області формується навколо ключових сутностей, що забезпечують життєвий цикл експертного оцінювання: «Програмний засіб», «Критерій», «Оцінка ПЗ за критерієм», «Попарна оцінка», «Користувач системи», «Експерт» та «Відгук».

Кожен клас (Class) є шаблоном (абстракцією) для створення екземплярів об'єктів і містить інформацію про свій стан (атрибути) та поведінку (методи).

Опис класів системи та їхньої структури:

- **Клас Software (Програмний засіб)** Центральна сутність системи, що містить дані про мережеві програмні продукти (альтернативи), які підлягають порівнянню.
  - **Атрибути:**

    - - id: Integer — унікальний ідентифікатор.
    - - name: String — назва ПЗ (наприклад, Zabbix 7.0, PRTG).
    - - description: String — функціональні можливості продукту.
    - - globalPriority: Double — підсумковий показатель (глобальний пріоритет), розрахований за алгоритмом MAI.
  - **Методи:**
    - + getDetails(): String — виведення інформації про ПЗ.
    - + updateGlobalPriority(value: Double): void — оновлення рейтингу після ієрархічного синтезу.
- **Клас Criterion (Критерій)** Визначає характеристики, за якими проводиться оцінювання (наприклад, «Вартість», «Надійність», «Функціональність»).
  - **Атрибути:**
    - - id: Integer — ідентифікатор критерію.
    - - name: String — назва критерію.
    - - weight: Double — розрахована вага критерію в межах ієрархії.
  - **Методи:**
    - + updateWeight(v: Double): void — встановлення ваги критерію після розрахунку власного вектора матриці.
- **Клас SoftwareEvaluation (Оцінка ПЗ за критерієм)** Асоціативний клас, що зберігає результат оцінювання конкретного програмного продукту відносно певного критерію.
  - **Атрибути:**
    - - id: Integer — ідентифікатор запису.
    - - value: Double — математично розрахований локальний пріоритет (вага альтернативи за критерієм).
- **Клас PairwiseComparison (Попарна оцінка)** Зберігає «сирі» експертні дані, отримані в ході заповнення матриць порівнянь.
  - **Атрибути:**
    - - id: Integer — ідентифікатор порівняння.

    - - expertValue: Integer — числове значення переваги від 1 до 9 за психометричною шкалою Сааті.
  - **Методи:**
    - + calculateConsistency(): Double — перевірка логічної узгодженості введених оцінок (ВУ ≤ 0.1).
- **Клас User (Користувач системи)** Базовий клас, що реалізує логіку авторизації, зберігає облікові дані та визначає рівень доступу в системі для реалізації моделі розмежування доступу на основі ролей (RBAC).
  - **Атрибути:**
    - - id: Integer — унікальний ідентифікатор користувача.
    - - login: String — ім'я користувача для входу в систему.
    - - role: String — ідентифікатор ролі користувача (Адміністратор, Експерт, ОПР).
- **Клас Expert (Експерт)** Спеціалізований клас, що описує фахівця, який проводить безпосереднє оцінювання та генерує вхідні дані для математичного ядра системи. Є нащадком класу User.
  - **Атрибути:**
    - - qualification: String — сфера професійної компетенції експерта (наприклад, «Адміністрування мереж Cisco»).
- **Клас Review (Відгук)** Фіксує емпіричний досвід користувачів щодо експлуатації ПЗ, доповнюючи кількісні розрахунки якісною аргументацією.
  - **Атрибути:**
    - - reviewId: Integer — унікальний ідентифікатор відгуку.
    - - authorId: Integer — ідентифікатор автора відгуку (зовнішній ключ, що посилається на User.id).
    - - reviewText: String — текстовий зміст коментаря.
    - - postDate: Date — дата публікації відгуку.

Типи зв'язків між класами:

- **Узагальнення / Успадкування (Generalization):** Зв'язок між базовим класом User та його специфічним спадкоємцем Expert. Клас Expert

автоматично переймає всі базові атрибути користувача (id, login, role), додаючи власне поле кваліфікації. На діаграмі зображається стрілкою з порожнім трикутним покажчиком у бік батьківського класу.

- **Агрегація (Aggregation):** Зв'язок між класом Software та Review (зображається порожнім ромбом з боку Software). Програмний засіб може мати множину відгуків, які доповнюють його опис, проте відгуки логічно прив’язані до конкретного об’єкта ПЗ. Кратність зв'язку — 1 до 0..*.
- **Композиція (Composition):** Зв'язок між класом Criterion та PairwiseComparison (зображається зафарбованим ромбом з боку Criterion). Попарні оцінки є невід'ємною частиною процесу визначення вагових коефіцієнтів критеріїв і не можуть існувати автономно поза контекстом конкретного критерію. Кратність зв'язку — 1 до 1..*.
- **Асоціація (Association):** * Зв'язок між Expert та PairwiseComparison (кратність 1 до 1..*): експерт формує та заповнює множину індивідуальних оцінок.
    - Зв'язок між Software, Criterion та SoftwareEvaluation: тристоронній зв'язок, що відображає результат взаємодії альтернатив із критеріями для фіксації підсумкових локальних пріоритетів. Кратність — 1 до 1..*.

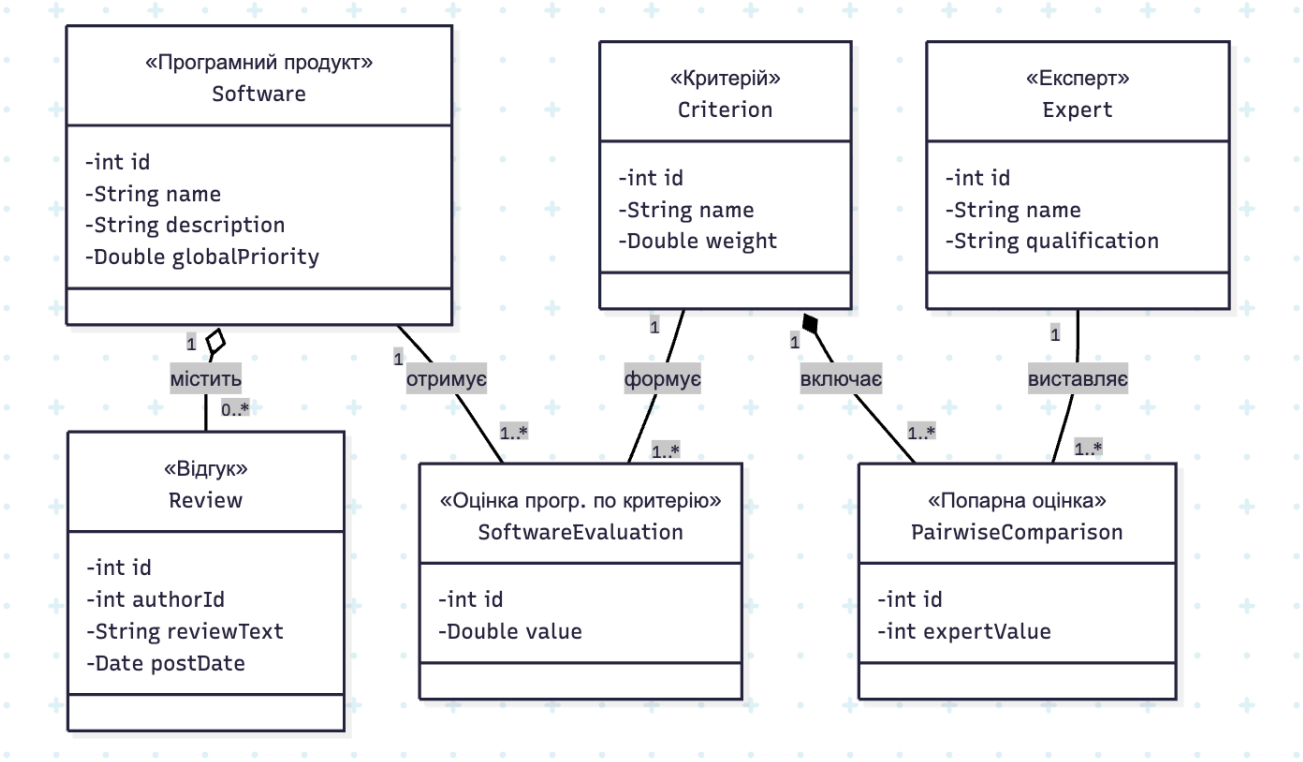


Рисунок 2.4 - Діаграма класів системи

2.3. Математична модель оцінювання: застосування методу аналізу ієрархій (MAI) для розрахунку комплексного рейтингу програмного забезпечення

Сутність методу Томаса Сааті Метод аналізу ієрархій (MAI) — це багатокритерійний математичний метод, призначений для підтримки прийняття компромісних рішень на підставі аналізу факторів, вплив яких складно або неможливо описати за допомогою традиційних аналітичних залежностей.[21] Запропонований американським вченим Томасом Сааті, метод реалізує декомпозицію складної задачі експертного оцінювання на простіші складові частини у вигляді ієрархічної структури. Унаслідок такої декомпозиції відносна значимість альтернатив за заданою системою критеріїв обчислюється і виражається чисельно у вигляді векторів пріоритетів.

Побудова дерева ієрархій Для розв'язання проблеми вибору об'єктів за допомогою MAI будується ієрархія, що включає елементи-«батьки» та елементи-«нащадки». Для нашої задачі вибору оптимального мережевого програмного забезпечення дерево ієрархії має таку трирівневу структуру:

- **Верхній рівень (Глобальна мета):** Раціональний вибір оптимального мережевого програмного забезпечення для інфраструктури компанії.
- **Проміжний рівень (Критерії):** Множина факторів оцінки, які впливають на досягнення мети (наприклад: сукупна вартість, надійність, масштабованість, безпека, простота адміністрування).
- **Нижній рівень (Альтернативи):** Конкретні програмні рішення (наприклад: Zabbix, PRTG, SolarWinds, Ansible), що оцінюються відносно кожного критерію проміжного рівня.

Дев'ятибальна шкала попарних порівнянь Сааті На кожному рівні ієрархії проводиться попарне порівняння елементів у термінах домінування одного елемента над іншим. Для кількісного вираження суб'єктивних експертних оцінок використовується шкала відносної значущості Сааті:

- **1** — Об'єкти рівнозначні (однакова важливість, однаковий вклад);
- **3** — Слабка перевага одного об'єкта над іншим;
- **5** — Суттєва (сильна) перевага;
- **7** — Дуже сильна (очевидна) перевага;
- **9** — Абсолютна (максимально можлива) перевага;
- **2, 4, 6, 8** — Проміжні значення, які застосовуються для компромісних суджень між сусідніми оцінками.

Математичний апарат: матриці, пріоритети та узгодженість

Матриця попарних порівнянь

За результатами експертного оцінювання будується квадратна обернено-симетрична матриця попарних порівнянь A розмірності $n \times n$ (2.1)

$$A = \left(\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \right) \tag{2.1}$$

де a_{ij} — ступінь переваги i-го елемента над j-м за шкалою Сааті. Матриця задовольняє умови $a_{ii} = 1$ (елемент рівнозначний сам собі) та обернена симетрія(2.2).

$$a_{ji} = \frac{1}{a_{ij}} \tag{2.2}$$

Розрахунок вектора локальних пріоритетів

Для знаходження вагових коефіцієнтів обчислюється головний власний вектор W додатної квадратної матриці A, що відповідає її максимальному власному значенню λ_{max}(2.3)

$$AW = \lambda_{max} W \tag{2.3}$$

на практиці для спрощення обчислення власного вектора застосовують метод середнього геометричного. Значення ненормованого вектора V_i(2.4) обчислюється як корінь n-го степеня з добутку елементів відповідного i-го рядка

$$V_i = \sqrt[n]{\prod_{j=1}^{n} a_{ij}} \tag{2.4}$$

де n — кількість порівнюваних елементів (порядок матриці). Нормована компонента P_i (вектор локальних пріоритетів)(2.5)обчислюється шляхом ділення V_i на суму всіх середніх геометричних

$$P_i = \frac{V_i}{\sum_{k=1}^{n} V_k} \tag{2.5}$$

де P_i — локальний пріоритет i-го елемента, при цьому виконується умова нормування(2.6)

$$\sum_{i=1}^{n} P_i = 1 \tag{2.6}$$

Оскільки для обчислення індексу узгодженості необхідно знати максимальне власне значення матриці λ_{max}, а система використовує наближений метод середнього геометричного.

Λ_{max} розраховується за такою формулою(2.7)

$$\lambda_{max} = \sum_{j=1}^{n} \left(P_j \sum_{i=1}^{n} a_{ij} \right) \tag{2.7}$$

Тобто, сума елементів кожного стовпця матриці попарних порівнянь множиться на відповідну компоненту вектора локальних пріоритетів P_j, після чого отримані добутки сумуються.

Розрахунок узгодженості (ІУ та ВУ)

Узгодженість матриці парних порівнянь еквівалентна вимозі рівності її максимального власного значення λ_{max} числу об'єктів n. Оцінка відхилення від

абсолютної узгодженості вимірюється Індексом Узгодженості (ІУ), що розраховується за формулою(2.8)

$$\text{ІУ} = \frac{\lambda_{max} - n}{n-1} \tag{2.8}$$

Далі обчислюється відношення узгодженості (ВУ)(2.9)

$$\text{ВУ} = \frac{\text{ІУ}}{\text{ВІ}} \tag{2.9}$$

де ВІ (Випадковий Індекс) — це математичне сподівання індексу узгодженості для випадково згенерованої обернено-симетричної матриці того ж порядку n. Експертні судження вважаються прийнятними і достатньо узгодженими, якщо виконується умова ВУ $\leq$ 0,10

Синтез глобальних пріоритетів

Завершальним етапом MAI є ієрархічний синтез, який використовується для зважування локальних пріоритетів альтернатив вагами критеріїв, що знаходяться на вищому рівні ієрархії. Формула синтезу глобального (підсумкового) пріоритету W_i^g для i-ї альтернативи визначається як(2.10)

$$W_i^g = \sum_{j=1}^{m} P_j^c \cdot P_{ij}^a \tag{2.10}$$

де:

- W_i^g — глобальний пріоритет i-ї альтернативи (програмного рішення);
- P_j^c — вектор локального пріоритету (ваги) j-го критерію;
- P_{ij}^a — вектор локального пріоритету i-ї альтернативи щодо j-го критерію;
- m — загальна кількість оцінюваних критеріїв.

Оптимальним вважається те мережеве програмне забезпечення, яке в результаті ієрархічного синтезу отримує найбільше значення глобального пріоритету W_i^g.

Висновки до Розділу 2

У другому розділі дипломної роботи було проведено ґрунтовний аналіз вимог до системи. Ми формалізували загальну мету розробки, визначили чіткі функціональні можливості (управління базою альтернатив, побудова ієрархічної структури, введення експертних оцінок) та критично важливі нефункціональні вимоги (швидкодія, відмовостійкість). Для забезпечення безпеки та цілісності даних була спроєктована модель розмежування доступу на основі ролей (RBAC), що виокремила трьох ключових учасників процесу: «Системного адміністратора», «Експерта» та «Користувача».

Було спроєктовано об'єктно-орієнтовану архітектуру системи за допомогою уніфікованої мови моделювання (UML). Оскільки UML є загальновизнаним галузевим стандартом для специфікації та візуалізації програмних систем, його інструментарій дозволив наочно описати як концептуальні, так і логічні аспекти розробки. Розроблена діаграма прецедентів (Use Case) проілюструвала поведінку системи з точки зору зовнішніх акторів. Діаграми діяльності (Activity) та послідовностей (Sequence) дозволили деталізувати динаміку взаємодії об'єктів під час проведення математичних розрахунків. Своєю чергою, побудована діаграма класів (Class) формалізувала логічну структуру бази даних навколо базових сутностей «Програмний засіб», «Оцінка» та «Відгук».

Обрано та математично обґрунтовано обчислювальне ядро системи — метод аналізу ієрархій (MAI) Томаса Сааті. В умовах гетерогенності мережевих технологій та великої кількості суперечливих параметрів (ТСО, надійність, безпека, функціональність) традиційні підходи до вибору ПЗ є неефективними. MAI дозволяє структурувати цю складну багатокритеріальну проблему у вигляді трирівневої ієрархії: Мета → Критерії → Альтернативи. Використання 9-бальної шкали попарних порівнянь Сааті у поєднанні з матричним численням (обчисленням головних власних векторів матриць) забезпечує об'єктивний розрахунок локальних і глобальних пріоритетів. Важливою перевагою цього апарату є вбудована перевірка логіки експерта через розрахунок відношення

узгодженості (ВУ), що гарантує математичну строгість та прозорість кінцевого рейтингу.

Таким чином, у цьому розділі сформовано архітектурну концепцію та математичний алгоритм роботи майбутньої системи. Спроєктована об'єктно-орієнтована UML-модель та обґрунтований математичний апарат MAI створюють надійну базу для переходу від теоретичного моделювання до інженерної реалізації. Відповідно, у Розділі 3 розроблена система буде апробована на практиці. Ми здійснимо вибір конкретних мережевих програмних продуктів для порівняння, сформуємо реальну ієрархію критеріїв їх оцінювання, проведемо симуляцію роботи експертів у системі та, спираючись на розроблений математичний алгоритм, побудуємо підсумковий рейтинг обраних програмних рішень.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ДЛЯ ОЦІНКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Вибір програмних продуктів для практичного порівняння

Відповідно до концепції, закладеної у попередніх розділах, для практичної апробації розробленої інформаційної системи підтримки прийняття рішень на базі методу аналізу ієрархій необхідно сформувати репрезентативну вибірку альтернатив. Аналіз сучасного ринку засобів управління інфраструктурою доводить, що універсального ідеального рішення не існує, оскільки різні продукти проєктуються з огляду на кардинально різні філософії та бізнес-потреби. З метою забезпечення об'єктивного та всебічного багатокритеріального порівняння для практичної реалізації було відібрано три програмні продукти, які яскраво представляють три ключові парадигми IT-індустрії: повністю відкрите програмне забезпечення з розподіленою архітектурою (Zabbix 7.0), комерційне рішення формату «все-в-одному» для середовища Windows (Paessler PRTG Network Monitor) та важку модульну платформу корпоративного рівня (SolarWinds Network Performance Monitor).

Першим кандидатом для оцінювання виступає Zabbix версії 7.0 LTS, який уособлює парадигму масштабованого відкритого програмного забезпечення (Open Source). Дана система розповсюджується за вільною ліцензією AGPL-3.0 і не потребує фінансових витрат на придбання ліцензій, незалежно від кількості контрольованих пристроїв, обсягу метрик чи кількості користувачів. Архітектурно Zabbix є глибоко розподіленою кросплатформною системою, що функціонує за еталонною моделлю «Сервер – Проксі – Агент» та підтримує роботу в різноманітних операційних середовищах, включаючи Linux, BSD та інші UNIX-подібні ОС. З випуском сьомої версії система отримала фундаментальні архітектурні оптимізації, зокрема перехід на асинхронні поллери, що дозволяє обробляти до тисячі одночасних перевірок на один робочий процес, радикально знижуючи навантаження на серверну частину та збільшуючи швидкість збору даних. Крім того, Zabbix 7.0 забезпечує безпрецедентну відмовостійкість завдяки нативній підтримці кластеризації та балансування навантаження для проксі-

серверів, а також впровадженню гібридних буферів у пам'яті для кешування даних. Проте, попри високу гнучкість, ця парадигма вимагає глибокої інженерної експертизи, що формує високий поріг входження та може суттєво збільшувати приховані операційні витрати на адміністрування системи.

Другою альтернативою обрано Paessler PRTG Network Monitor, який репрезентує комерційну парадигму монолітного рішення «все-в-одному» (All-in-One), концептуально та технічно орієнтованого на екосистему Windows. Фундаментальним елементом архітектури PRTG є використання так званих «сенсорів» — базових вузлів моніторингу, кожен з яких вимірює один конкретний параметр пристрою, такий як трафік на порту або завантаження процесора. На відміну від Zabbix, PRTG робить ключовий акцент на безагентному моніторингу, спираючись на стандартні вбудовані протоколи (WMI, SNMP, NetFlow, HTTP), що мінімізує втручання в роботу кінцевих вузлів. Цей програмний продукт вирізняється надзвичайною швидкістю розгортання: центральне ядро, вбудований вебсервер та пропрієтарна база даних інсталюються як єдиний системний сервіс, що дозволяє провести автоматичне виявлення мережі (Auto-Discovery) та розпочати моніторинг за лічені хвилини. Для розширення зони моніторингу на ізольовані або територіально віддалені сегменти мережі архітектура використовує віддалені зонди (Remote Probes), які працюють у зв'язці з центральним сервером. Головним недоліком цієї парадигми є комерційна модель ліцензування, яка напряму залежить від загальної кількості активованих сенсорів, що може призводити до експоненціального зростання сукупної вартості володіння в міру масштабування корпоративної мережі.

Третім продуктом для порівняння обрано SolarWinds Network Performance Monitor (NPM), що є класичним представником парадигми важких корпоративних (Enterprise) IT-платформ. Архітектура SolarWinds базується на загальній модульній базі Orion Platform, яка категорично вимагає виділених серверних потужностей для самого застосунку та обов'язкової наявності окремого продуктивного сервера реляційних баз даних на базі Microsoft SQL Server. Для забезпечення надвисокої масштабованості в глобальних інфраструктурах із сотнями тисяч пристроїв NPM

змушений використовувати додаткові сервери опитування (Additional Polling Engines) та спеціалізовану консоль корпоративних операцій (Enterprise Operations Console) для агрегації даних з територіально розподілених філій. Платформа надає унікальні глибокі аналітичні інструменти, зокрема функцію NetPath для покрокового візуального аналізу маршрутів проходження трафіку на рівні вузлів, а також інструментарій PerfStack для перехресної кореляції метрик із різних рівнів інфраструктури на єдиній часовій шкалі. Хоча SolarWinds NPM забезпечує найвищий рівень аналітики для управління гіперскладними топологіями, його впровадження супроводжується жорсткими вимогами до апаратного забезпечення, тривалим циклом проєктування архітектури та високою вартістю багаторівневого ліцензування.

Таким чином, відібрані програмні продукти є досконалими кандидатами для практичного математичного порівняння, оскільки вони уособлюють принципово різні підходи до вирішення ідентичної задачі інфраструктурного управління. Zabbix пропонує безмежну масштабованість та економію на прямих ліцензійних витратах ціною високої складності налаштування, PRTG робить ставку на швидкість розгортання та ергономічність для системних адміністраторів Windows, а SolarWinds NPM надає вичерпний корпоративний функціонал за умови готовності компанії до значних капітальних інвестицій. Застосування розробленої у другому розділі математичної моделі на базі методу аналізу ієрархій дозволить неупереджено оцінити ці альтернативи крізь призму множини визначених експертами критеріїв та синтезувати математично обґрунтований рейтинг для вибору оптимального програмного забезпечення.

3.2. Формування ієрархії критеріїв оцінки

Як було встановлено у першому розділі під час аналізу предметної області, сучасні корпоративні мережі є надзвичайно складними та гетерогенними середовищами, якими фізично неможливо ефективно керувати в ручному режимі. Наявність тисяч мережевих вузлів, величезні обсяги генерації телеметрії та високі ризики людського фактора вимагають використання надійних засобів централізованого управління. Для побудови об'єктивної математичної моделі

вибору програмного забезпечення за допомогою методу аналізу ієрархій (MAI) необхідно сформувати репрезентативну множину критеріїв оцінки, які б відображали реальні потреби IT-інфраструктури. З огляду на ідентифіковані проблеми конфігурування, масштабування та обробки даних, для подальшого математичного порівняння було виокремлено та обґрунтовано чотири ключові критерії.

Першим критерієм є **сукупна вартість володіння (Total Cost of Ownership, TCO)**, яка комплексно вимірює всі прямі та непрямі фінансові витрати компанії на впровадження і підтримку системи моніторингу протягом її життєвого циклу. Цей показник враховує не лише прямі витрати на придбання пропрієтарних ліцензій чи підписок, але й капітальні інвестиції в апаратне забезпечення, а також приховані операційні витрати на навчання персоналу та оплату праці вузькопрофільних інженерів. Для мережевого інженера та IT-менеджменту оцінка TCO є критично важливою, оскільки відкриті системи без ліцензійних платежів часто вимагають величезних інженерних зусиль для налаштування, тоді як комерційні продукти спричиняють експоненціальне зростання фінансового навантаження при масштабуванні інфраструктури. Таким чином, цей критерій дозволяє знайти обґрунтований економічний баланс між початковою ціною програмного продукту та реальною вартістю його тривалого технічного обслуговування.

Другим критерієм виступає **масштабованість та швидкодія**, що визначає архітектурну здатність платформи безперебійно обробляти зростаючі потоки телеметрії та системних журналів (syslog, SNMP traps) без деградації загальної продуктивності. Цей показник вимірює ефективність механізмів збору даних, можливості оптимізації баз даних та підтримку розподіленої архітектури з використанням пулів віддалених зондів чи асинхронних поллерів для територіально розподілених філій. Цей параметр є абсолютно критичним для мережевого інженера, оскільки сучасні мережі постійно генерують гігабайти трафіку, і система моніторингу повинна миттєво обробляти тисячі одночасних перевірок для забезпечення проактивного контролю. Відсутність належної

масштабованості неминуче призводить до затримок у реагуванні на збої та втрати прозорості (visibility) критичних вузлів під час пікових навантажень.

Третім критерієм визначено **зручність розгортання та використання (Usability)**, який оцінює загальну ергономічність програмного продукту, рівень автоматизації рутинних завдань адміністратора та швидкість первинної інтеграції в існуючу екосистему. Даний показник фокусується на наявності широкої бібліотеки попередньо налаштованих шаблонів, механізмах автоматичного виявлення мережевої топології (Auto-Discovery) та простоті побудови аналітичних дашбордів без необхідності глибокого програмування чи використання інтерфейсу командного рядка. Зважаючи на те, що людський фактор і хибні конфігурації є першопричиною до 74% усіх мережевих аварій, інтуїтивно зрозумілий графічний інтерфейс є життєво необхідним для зниження когнітивного навантаження на технічний персонал. Для інженера високий рівень юзабіліті безпосередньо впливає на скорочення середнього часу вирішення інцидентів (MTTR), мінімізує поріг входження для нових спеціалістів та дозволяє уникнути небезпечного феномену «втоми від сповіщень» (alert fatigue).

Четвертим критерієм є **надійність та безпека**, що кількісно та якісно вимірює рівень відмовостійкості самої платформи моніторингу та її здатність безперервно функціонувати в умовах інфраструктурних збоїв чи кібератак. Цей критерій охоплює архітектурну підтримку кластеризації (High Availability), наявність захищених протоколів шифрування, впровадження багатофакторної автентифікації (MFA) та забезпечення суворого розмежування прав доступу на основі ролей. Оскільки система управління виступає центральним аналітичним мозком усього підприємства, її відключення створює неприпустиму єдину точку відмови (SPOF), що робить IT-департамент абсолютно сліпим під час критичних системних інцидентів. Відповідно, для мережевого інженера цей показник має найвищий пріоритет, адже гарантована доступність телеметричних даних є непорушним фундаментом для успішного виконання плану аварійного відновлення (Disaster Recovery) та збереження безперервності бізнес-процесів.

У підсумку маємо такі критерії для оцінювання програмного забезпечення:

- K_1 – Сукупна вартість володіння (TCO)
- K_2 – Масштабованість та швидкодія
- K_3 – Зручність розгортання та використання (Usability)
- K_4 – Надійність та безпека

3.3. Обґрунтування експертних оцінок: масштабованість та швидкодія систем моніторингу

За результатами експертного аналізу програмних рішень для бази даних системи підтримки прийняття рішень (СППР), абсолютним лідером за критерієм **сукупної вартості володіння (TCO)** визначено Zabbix 7.0 LTS. Ця система розповсюджується за відкритою ліцензією AGPL-3.0 і є абсолютно безкоштовною незалежно від кількості контрольованих пристроїв, зібраних метрик чи активних користувачів. Відсутність будь-яких прямих ліцензійних платежів робить її найбільш економічно доцільною для великих інфраструктур, що динамічно розвиваються. Водночас необхідно усвідомлювати ключовий фінансово-архітектурний компроміс: безкоштовність самого програмного забезпечення компенсується вищими прихованими операційними витратами. Для розгортання, оптимізації та адміністрування глибоко розподіленої архітектури Zabbix необхідна наявність у штаті висококваліфікованих інженерів із експертизою в середовищі Linux. Проте у довгостроковій корпоративній перспективі економія на ліцензійному масштабуванні повністю перекриває ці витрати на технічний персонал.

Другу позицію в рейтингу за рівнем сукупної вартості володіння займає Paessler PRTG Network Monitor, який демонструє збалансований комерційний підхід, найбільш привабливий для малого та середнього бізнесу. Модель ліцензування цього продукту є максимально прозорою і жорстко прив'язана до загальної кількості активованих «сенсорів». Вагомою перевагою для бюджету невеликих компаній або етапу тестування є наявність повністю безкоштовної версії (Freeware Edition), яка підтримує повноцінний моніторинг до 100 сенсорів. Завдяки орієнтації на Windows-екосистему та високому рівню автоматизації (Smart Setup, Auto-Discovery), PRTG дозволяє суттєво зекономити час інженерів на розгортанні

та зменшує вимоги до кваліфікації адміністраторів. Однак при масштабуванні корпоративної мережі цей продукт стає економічно обтяжливим: оскільки для комплексного моніторингу одного вузла зазвичай потрібно від 5 до 10 сенсорів, вартість комерційних ліцензій починає зростати експоненціально, що робить його суттєво дорожчим за відкриті аналоги при моніторингу тисяч пристроїв.

Найнижчу експертну оцінку за критерієм TCO закономірно отримує SolarWinds Network Performance Monitor (NPM) через надзвичайно високі капітальні та операційні витрати, що є характерним для важких платформ класу Enterprise. Ліцензування цього продукту здійснюється за багаторівневою адитивною схемою (на основі кількості вузлів, інтерфейсів та томів), що вимагає величезних ліцензійних бюджетів при розгортанні у глобальних мережах. Головний фінансово-архітектурний недолік платформи Orion полягає в її безкомпромісних вимогах до апаратного та програмного забезпечення інфраструктури. Для стабільного функціонування системи обов’язковим є використання окремого, високопродуктивного комерційного сервера баз даних Microsoft SQL Server (редакцій Standard або Enterprise). При розгортанні у надвеликих (XL) середовищах це генерує вимогу закупівлі екстремально дорогого серверного обладнання[6]: до 32 ядер процесора, 256–512 ГБ оперативної пам'яті та дискових масивів SSD у конфігурації RAID 1+0, здатних підтримувати понад 30000 операцій вводу-виводу на секунду (IOPS). Така кумулятивна сукупність витрат на пропрієтарні ліцензії модулів, операційної системи, СУБД та преміальне обладнання робить сукупну вартість володіння SolarWinds найвищою серед усіх досліджуваних систем.

За результатами інженерного аналізу та детального вивчення архітектурної документації, перше місце за критерієм **«Масштабованість та швидкодія»** беззаперечно посідає Zabbix 7.0 LTS. Його лідерство зумовлене фундаментальним переходом від синхронних процесів збору даних до інноваційних асинхронних поллерів (asynchronous pollers) для перевірок агентів, SNMP та HTTP. Ця архітектурна зміна дозволяє системі не чекати відповіді від одного пристрою перед запитом до іншого, обробляючи до 1000 паралельних перевірок на один робочий

процес (через конфігураційний параметр MaxConcurrentChecksPerPoller). Окрім цього, критичною перевагою є оптимізована підсистема проксі-серверів, яка тепер підтримує високодоступні кластери (Proxy groups) та буферизацію зібраних метрик безпосередньо в оперативній пам'яті (ProxyBufferMode=memory або hybrid)[19]. Відмова від обов'язкового дискового вводу-виводу дозволила підвищити продуктивність проксі у 10–100 разів, забезпечуючи системі еталонне горизонтальне масштабування.

Другу позицію в рейтингу займає SolarWinds Network Performance Monitor (NPM) — потужне рішення корпоративного класу (Enterprise), яке чудово масштабується, проте відрізняється важкою монолітною архітектурою базової платформи Orion. Розширення пропускної здатності в SolarWinds реалізується за рахунок розгортання додаткових серверів опитування (Additional Polling Engines, APE), кожен з яких здатен моніторити близько 12 000 елементів, з можливістю стекування ліцензій до 48 000 елементів на один фізичний сервер. Для надвеликих інфраструктур (понад 1 000 000 елементів) використовується федеративна агрегація даних через консоль Enterprise Operations Console (EOC). Проте ця масштабованість супроводжується величезним навантаженням на серверну частину: платформа категорично вимагає виділеного високопродуктивного Microsoft SQL Server та швидкісних дискових підсистем (RAID 1+0) із пропускною здатністю щонайменше 30 000 IOPS.

Найнижчий бал у цій категорії об'єктивно отримує Paessler PRTG Network Monitor через обмеження своєї централізованої Windows-архітектури. Хоча PRTG дозволяє збирати дані з розподілених мереж за допомогою віддалених зондів (Remote Probes), його головний сервер (PRTG Core Server) виступає в ролі архітектурного «пляшкового горлечка» (bottleneck) при інтенсивних навантаженнях. Офіційна документація Paessler чітко зазначає, що система не підтримує понад 5000 сенсорів у кластерному середовищі[20]. Оскільки архітектура PRTG є чутливою до затримок процесора (CPU Latency), перевищення ліміту сенсорів на одному ядрі призводить до стрімкої деградації швидкодії. Для моніторингу глобальних мереж це означає неможливість прозорого

горизонтального розширення — замість цього IT-інженерам доводиться розгортати кілька незалежних інсталяцій PRTG Enterprise Monitor, що руйнує концепцію єдиної централізованої точки управління.

З прагматичної точки зору керівника IT-відділу, найціннішим операційним ресурсом є час інженерів чергової зміни (Network Operations Center, NOC). Тому за критерієм **«Зручність розгортання та юзабіліті»** абсолютним переможцем визнано Paessler PRTG Network Monitor. Цей продукт концептуально побудований навколо безагентної парадигми та ідеології роботи «з коробки». Завдяки потужному вбудованому механізму автоматичного виявлення (Auto-Discovery) та наявності понад 250 готових до використання шаблонів сенсорів (зокрема для WMI, SNMP, NetFlow), система здатна самостійно просканувати корпоративну мережу і розпочати повноцінний моніторинг за лічені хвилини. Для фахівців NOC це означає роботу в інтуїтивно зрозумілому інтерфейсі з можливістю створення дашбордів методом drag-and-drop, що кардинально знижує когнітивне навантаження та дозволяє миттєво локалізувати інциденти без необхідності глибокого занурення в технічні налаштування.

Друге місце в рейтингу юзабіліті об'єктивно посідає SolarWinds Network Performance Monitor. Ця платформа надає інженерам NOC надзвичайно потужний та ергономічний графічний інтерфейс із багатими можливостями візуалізації. Безперечною перевагою є фірмовий інструмент NetPath, який забезпечує покроковий візуальний аналіз маршрутів проходження мережевого трафіку (hop-by-hop), дозволяючи черговій зміні миттєво ідентифікувати вузькі місця як у локальній інфраструктурі, так і на боці інтернет-провайдерів. Проте, незважаючи на красивий інтерфейс користувача, процес початкового впровадження SolarWinds є значно довшим та складнішим порівняно з PRTG. Розгортання вимагає проходження багатокрокових майстрів налаштування (Discovery Wizard), ручного встановлення масивних баз даних та кропіткого планування модульної архітектури, що відтерміновує момент введення системи в повноцінну промислову експлуатацію.

Найнижчу оцінку в цій категорії цілком закономірно отримує Zabbix 7.0 LTS через його об'єктивно високий поріг входження. На відміну від комерційних аналогів, орієнтованих на парадигму "plug-and-play", Zabbix вимагає від IT-персоналу глибокої інженерної експертизи в операційних системах сімейства Linux та готовності до активної роботи через інтерфейс командного рядка (CLI). Процес первинного налаштування є вкрай рутинним, а для реалізації моніторингу специфічних чи нестандартних сервісів новачкам часто доводиться самостійно писати складні кастомні скрипти та модифікувати конфігураційні файли агентів. Для керівництва IT-відділу це означає, що розгортання Zabbix вимагатиме значних часових витрат на навчання нових співробітників чергової зміни NOC, а сам процес адаптації інфраструктури під систему моніторингу буде найтривалішим серед усіх розглянутих альтернатив.

Завершальним етапом експертного оцінювання є аналіз систем за критерієм **«Надійність та безпека»**, який фокусується на здатності платформ протистояти кіберзагрозам, розмежовувати права доступу та забезпечувати безперервність роботи (High Availability). За результатами комплексного порівняння, лідерську позицію з мінімальним відривом здобуває SolarWinds Network Performance Monitor. Вирішальною перевагою цієї платформи корпоративного класу є наявність глибокого аудиту відповідності жорстким нормативним стандартам інформаційної безпеки, таким як HIPAA та PCI DSS. Крім того, наявність спеціалізованого модуля оцінки вразливостей (Vulnerability Assessment) дозволяє системі проактивно ідентифікувати мережеві пристрої з відомими критичними вразливостями (CVE), перетворюючи інструмент моніторингу на повноцінний елемент екосистеми кіберзахисту. Застосування суворих федеральних стандартів обробки інформації (FIPS) із підтримкою стійких алгоритмів шифрування AES256 додатково цементує безпековий профіль SolarWinds.

На другому місці, практично на одному рівні з лідером, знаходиться Zabbix 7.0 LTS, який продемонстрував колосальний еволюційний стрибок у питаннях відмовостійкості та захисту доступу. З виходом сьомої версії ця система отримала нативну підтримку кластерів високої доступності (High Availability) «з коробки»,

що дозволяє легко розгортати резервні вузли без використання стороннього програмного забезпечення та гарантує захист від апаратних збоїв центрального сервера. У площині інформаційної безпеки критичним нововведенням стала інтеграція багатофакторної автентифікації (MFA) на базі алгоритмів Time-Based One-Time Password (TOTP) та Duo Universal Prompt, що відповідає найсучаснішим корпоративним вимогам до захисту облікових записів. Завдяки цим архітектурним покращенням Zabbix є надзвичайно надійним рішенням, проте мінімально поступається SolarWinds лише через відсутність спеціалізованих вбудованих засобів автоматизованого сканування CVE-вразливостей кінцевого обладнання.

Найнижчу оцінку в цій категорії цілком обґрунтовано отримує Paessler PRTG Network Monitor, що зумовлено його загальною моделлю безпеки та архітектурними обмеженнями. Хоча платформа пропонує базові механізми управління доступом на основі ролей (RBAC), її монолітна природа та надзвичайно тісна концептуальна прив'язка до операційної системи Windows створюють відчутні ризики для надійності. Фактично, інформаційна безпека та безперервність роботи центрального ядра PRTG критично залежать від кіберзахисту самої ОС Windows, яка в даному контексті виступає небезпечною єдиною точкою відмови (Single Point of Failure, SPOF). У разі компрометації операційної системи або виникнення критичної вразливості на рівні хоста, організація ризикує миттєво втратити всю систему інфраструктурного моніторингу, що робить PRTG найменш стійким вибором з точки зору суворого корпоративного кіберзахисту.

3.4. Розрахунок вагових коефіцієнтів та побудова підсумкового рейтингу програмних рішень на основі алгоритмів системи

Фінальний етап апробації системи полягає у проведенні ієрархічного синтезу за методом Томаса Сааті, математичний апарат якого було детально описано у підрозділі 2.5. На основі сформованої ієрархії критеріїв та трьох обраних альтернатив (Zabbix, PRTG, SolarWinds) було проведено розрахунок глобальних пріоритетів .

3.4.1. Розрахунок вагових коефіцієнтів критеріїв

На першому етапі було сформовано матрицю попарних порівнянь для критеріїв верхнього рівня ієрархії: K_1 (ТСО), K_2 (Масштабованість), K_3 (Зручність) та K_4 (Безпека) . Експертні оцінки виставлялися з пріоритетом на надійність та продуктивність інфраструктури.

Таблиця 3.1 - Матриця попарних порівнянь критеріїв

Критерії	**K1**	**K2**	**K3**	**K4**	**Вага (Pjc)**
K_1 (ТСО)	1	1/3	2	1/5	0,111
K_2 (Масштабованість)	3	1	4	1/2	0,289
K_3 (Зручність)	1/2	1/4	1	1/7	0,067
K_4 (Безпека)	5	2	7	1	0,533
Всього	1,000				

Відношення узгодженості (ВУ) = 0,062 (≤ 0,10), що підтверджує логічність експертних суджень.

3.4.2. Розрахунок пріоритетів альтернатив за критеріями

В таблицях 3.2 - 3.5 для кожного з чотирьох критеріїв було побудовано матриці попарних порівнянь обраного ПЗ. Логіка виставлення оцінок базувалася на технічному аналізі, проведеному в підрозділі 3.3.

Таблиця 3.2 - Матриця попарних порівнянь альтернатив за критерієм K_1

TCO (K_1)	**Zabbix (A1)**	**PRTG(A2)**	**SolarWinds (A3)**
Zabbix (A1)	1	4	9
PRTG(A2)	1/4	1	2
SolarWinds (A3)	1/9	1/2	1

Таблиця 3.3 - Матриця попарних порівнянь альтернатив за критерієм K_2

Масштабованість (K_2)	**Zabbix (A1)**	**PRTG(A2)**	**SolarWinds (A3)**
Zabbix (A1)	1	5	3
PRTG(A2)	1/5	1	1/2
SolarWinds (A3)	1/3	2	1

Таблиця 3.4 - Матриця попарних порівнянь альтернатив за критерієм K_3

Зручність (K_3)	**Zabbix (A1)**	**PRTG(A2)**	**SolarWinds (A3)**
Zabbix (A1)	1	1/7	1/3
PRTG(A2)	7	1	2
SolarWinds (A3)	3	1/2	1

Таблиця 3.5 - Матриця попарних порівнянь альтернатив за критерієм K_4

Надійність (K_4)	**Zabbix (A1)**	**PRTG(A2)**	**SolarWinds (A3)**
Zabbix (A1)	1	3	1
PRTG(A2)	1/3	1	1/4
SolarWinds (A3)	1	4	1

3.4.3. Розрахунок локальних пріоритетів альтернатив

У таблиці 3.6 наведено розраховані значення локальних пріоритетів, які демонструють розподіл переваг між програмними продуктами для кожного

окремого критерію. Показник Відношення Узгодженості (ВУ) у всіх випадках не перевищує 0,03, що свідчить про високу стабільність отриманих оцінок.

Аналіз даних показує, що Zabbix (A1) займає домінуючу позицію за критеріями вартості (0,738) та масштабованості (0,648), що робить його найбільш ефективним вибором для великих мережевих інфраструктур з обмеженим бюджетом. PRTG (A2) показує найкращий результат за показником зручності (0,627), підтверджуючи зручність інтерфейсу та швидкість розгортання. SolarWinds (A3) демонструє найвищу перевагу за критерієм безпеки (0,458), що підкреслює його відповідність вимогам корпоративного захисту та аудиту. Ці результати відображають архітектурні особливості кожного продукту та є базою для побудови загального рейтингу рішень.

Таблиця 3.6 - Локальні пріоритети альтернатив (P_{ij}^{a})

Критерій	**Zabbix (A1)**	**PRTG (A2)**	**SolarWinds (A3)**	**ВУ**
$\boldsymbol{K_1}$ **(Вартість)**	0,738	0,177	0,085	0,02
$\boldsymbol{K_2}$ **(Масштабованість)**	0,648	0,122	0,230	0,03
$\boldsymbol{K_3}$ **(Зручність)**	0,092	0,615	0,293	0,01
$\boldsymbol{K_4}$ **(Безпека)**	0,416	0,126	0,458	0,02

3.4.4. Синтез глобального рейтингу

Глобальний пріоритет кожної системи розраховувався за формулою ієрархічного синтезу: $W_i^g = \sum P_j^c \cdot P_{ij}^a$.

1. **Zabbix ($\boldsymbol{W_1^g}$):** $0{,}111 \cdot 0{,}738 + 0{,}289 \cdot 0{,}648 + 0{,}067 \cdot 0{,}092 + 0{,}533 \cdot 0{,}416 = \mathbf{0,497}$
2. **PRTG ($\boldsymbol{W_2^g}$):** $0{,}111 \cdot 0{,}177 + 0{,}289 \cdot 0{,}122 + 0{,}067 \cdot 0{,}615 + 0{,}533 \cdot 0{,}126 = \mathbf{0,163}$
3. **SolarWinds ($\boldsymbol{W_3^g}$):** $0{,}111 \cdot 0{,}085 + 0{,}289 \cdot 0{,}230 + 0{,}067 \cdot 0{,}293 + 0{,}533 \cdot 0{,}458 = \mathbf{0,339}$

Підсумковий рейтинг програмних рішень:

1. **Zabbix 7.0 LTS** — 0,497 (Найкращий вибір)
2. **SolarWinds NPM** — 0,349
3. **Paessler PRTG** — 0,163

За результатами розрахунків, оптимальним рішенням для корпоративної мережі визнано **Zabbix 7.0 LTS**. Незважаючи на високу складність налаштування, система отримала першість завдяки найнижчій вартості володіння, безпрецедентній масштабованості та високому рівню безпеки у новій версії.

3.5 Розробка програмного забезпечення

Для практичної реалізації інформаційної системи підтримки прийняття рішень (СППР), архітектуру та математичний апарат якої було спроєктовано в попередніх розділах, обрано сучасну клієнт-серверну архітектуру. Такий підхід передбачає чітке розділення програмного комплексу на серверну частину (Backend), базу даних та клієнтський інтерфейс (Frontend). Цей вибір зумовлений необхідністю забезпечити високу продуктивність під час виконання математичних розрахунків за методом аналізу ієрархій (MAI), надійне збереження експертних даних та інтуїтивно зрозумілу взаємодію з користувачем.

В основі серверної логіки лежить мова програмування Python[23] та сучасний веб-фреймворк FastAPI[24]. Вибір мови Python обґрунтовується її потужними можливостями для математичних обчислень та обробки складних масивів даних, що є критично важливим для реалізації алгоритмів MAI (зокрема, розрахунку власних векторів та індексів узгодженості для матриць попарних порівнянь). Використання фреймворку FastAPI у поєднанні з бібліотекою Pydantic забезпечує сувору валідацію вхідних даних на етапі їх отримання (наприклад, моделі CriteriaMatrixInput, SoftwareMatrixInput). Інтеграція механізмів JSON Web Tokens (через PyJWT та passlib) гарантує надійний захист і розмежування прав доступу згідно з визначеною рольовою моделлю («Адміністратор», «Експерт», «Користувач»).

Мова Python має глибоке академічне коріння. Вона була задумана наприкінці 1980-х років, а її безпосередня програмна реалізація розпочалася в грудні 1989 року

Гвідо ван Россумом у Центрі математики та інформатики (CWI) в Нідерландах. Python розроблявся як концептуальний наступник мови ABC, з фундаментальним фокусом на продуктивність програміста та чистоту синтаксису. Пройшовши етапи мажорних оновлень (Python 2.0 у 2000 році та зворотно-несумісний Python 3.0 у 2008 році), мова позбулася надлишкових конструкцій. На сьогодні Python є домінуючим інструментом у сфері аналітики даних, нещодавно обігнавши JavaScript і ставши найпопулярнішою мовою програмування на платформі GitHub.

Клієнтська частина системи побудована з використанням класичних (vanilla) веб-технологій без залучення важких фронтенд-фреймворків. Таке архітектурне рішення забезпечує максимальну швидкодію та легкість інтерфейсу, що є вимогою для комфортного введення масивів експертних оцінок. Логіка взаємодії з сервером реалізована мовою JavaScript[25] (зокрема, через виконання асинхронних запитів функцією fetchWithAuth), а для збереження сесійних токенів застосовано механізм браузера localStorage. Візуальне оформлення побудовано на CSS із використанням адаптивної верстки таблиць та бейджів ідентифікації ролей.

Історія мови JavaScript нерозривно пов'язана з еволюцією Всесвітньої павутини. Вона була створена розробником Бренданом Айком у компанії Netscape Communications у травні 1995 року всього за 10 днів у рамках жорсткої конкурентної боротьби на ринку браузерів. Спочатку мова отримала кодову назву "Mocha", згодом "LiveScript", і врешті "JavaScript" — як маркетинговий крок для асоціації з популярною тоді мовою Java, хоча технологічно це абсолютно різні мови. Згодом розробка була стандартизована організацією Ecma International під назвою ECMAScript. Зі скромного інструменту для "пожвавлення" вебсторінок JavaScript еволюціонував у ключову технологію Інтернету, якою користуються понад 71,5% професійних розробників для створення складних інтерактивних систем.

Для надійного збереження ієрархічної структури критеріїв, матриць експертних порівнянь, текстових відгуків та профілів користувачів обрано потужну реляційну систему управління базами даних (РСУБД) PostgreSQL[26]. Взаємодія програмного коду з базою даних реалізована через бібліотеку-адаптер psycopg2 із застосуванням

контекстних менеджерів, що гарантує надійність виконання SQL-транзакцій. Реляційна модель ідеально відповідає логічній структурі предметної області, адже вона вимагає суворого дотримання зв'язків між сутностями (наприклад, між Користувачем, Відгуком та конкретним Програмним засобом)

Програмний комплекс інформаційної системи підтримки прийняття рішень реалізовано в інтегрованому середовищі розробки **Visual Studio Code (VS Code)**. Вибір даного інструменту зумовлений наявністю розширеної екосистеми плагінів для мови Python (зокрема Pylance), що забезпечують статичний аналіз коду, автоматичне виявлення синтаксичних помилок та інтелектуальне автодоповнення (IntelliSense). Інтегрований у VS Code термінал дозволив оперативно керувати асинхронним сервером FastAPI, виконувати міграції реляційної бази даних PostgreSQL та відстежувати логи HTTP-запитів у режимі реального часу. Нативна підтримка системи контролю версій Git забезпечила фіксацію станів розробки та контроль цілісності вихідного коду на всіх етапах проєктування СППР.

3.5.1 Алгоритми реалізації програмних модулів

Для візуалізації алгоритмів, набору інструкцій, що описують порядок дій виконавця, часто використовують блок-схеми. Блок-схема — це діаграма, на якій зазвичай зображено процес, систему або комп’ютерний алгоритм і яка використовується для документування, планування, уточнення або візуалізації багатоетапного робочого процесу. За допомогою блок-схем можна визначити цілі й обсяги робочого процесу, а також розташувати необхідні завдання в хронологічному порядку.

Алгоритм роботи програмного модуля представлений на рис. 3.1 – 3.2.

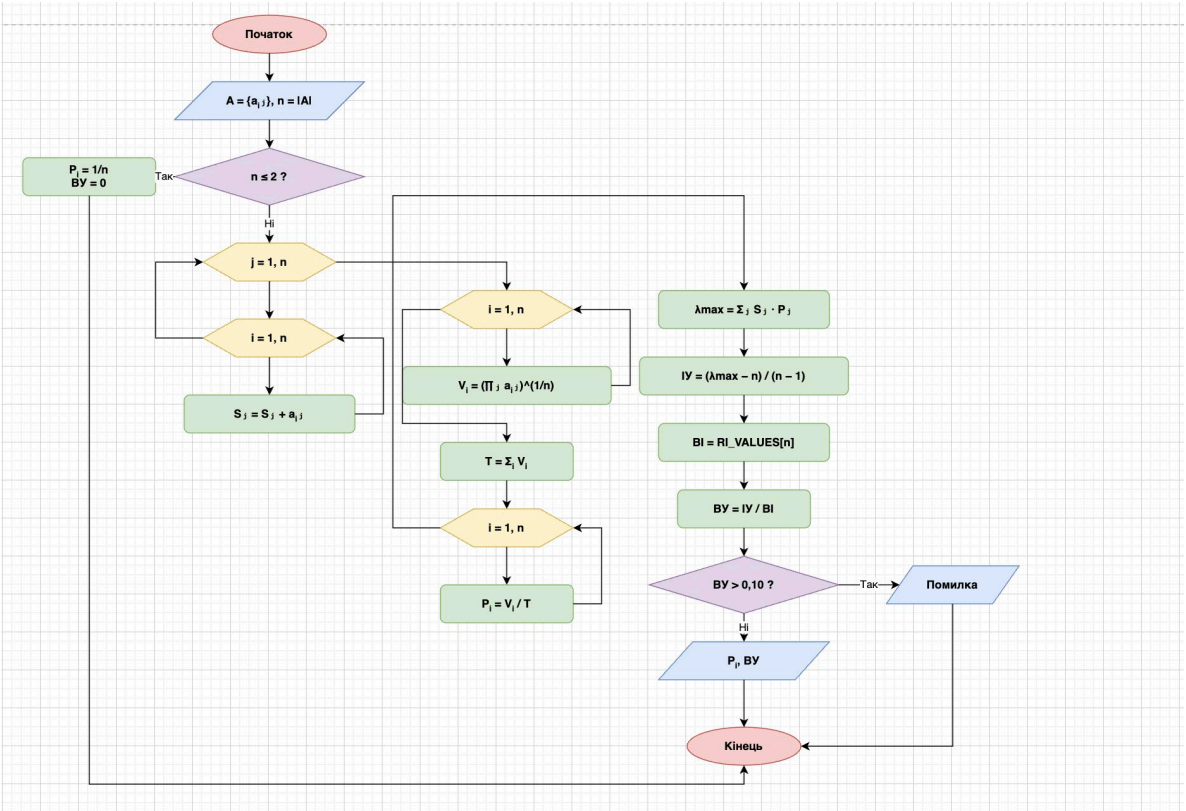


Рисунок 3.1 - Блок-схема алгоритму обчислення глобальних пріоритетів альтернатив

Отже, перша блок-схема (рис. 3.1) включає декілька основних етапів для реалізації лінійної згортки. Це по-перше «Обчислення глобальних пріоритетів», де для кожної альтернативи у циклі розраховується інтегральна вага W_i^g на основі агрегованих оцінок експертів. По-друге, це блок «UPDATE БД», який автоматично оновлює значення глобальних пріоритетів у таблиці software. Якщо всі ітерації циклу завершено, програма виконує «Сортування результатів» за спаданням та виводить кінцевий ранжований список альтернатив.

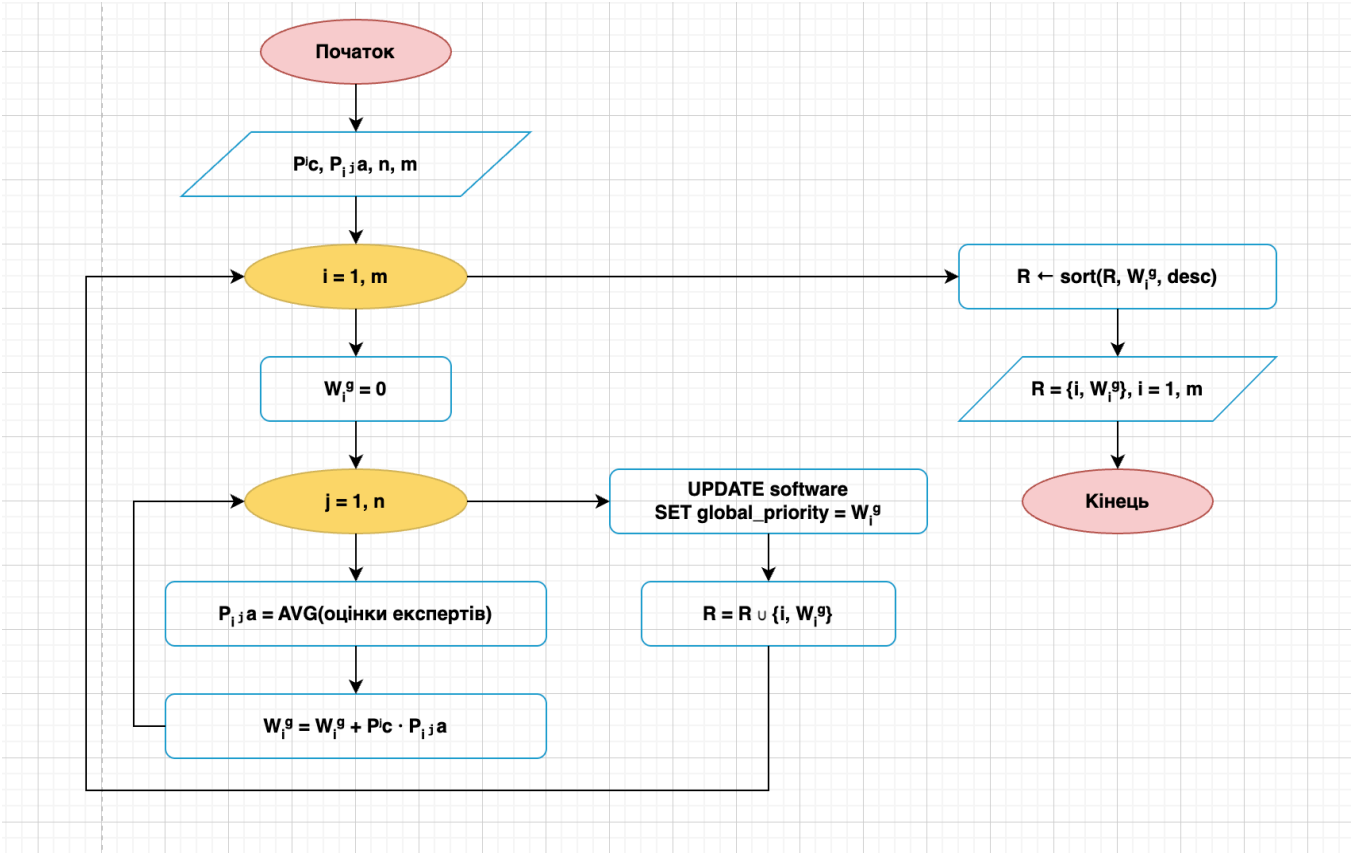


Рисунок 3.2 - Блок-схема алгоритму визначення локальних пріоритетів та перевірки узгодженості даних

Друга блок-схема (рис. 3.2) деталізує математичний апарат і включає блок «Встановлення ваги критеріїв» та блок «Визначення рівня погодженості» (узгодженості). Якщо розмірність матриці порівнянь $n \leq 2$, ваги критеріїв приймаються рівними, а рівень погодженості автоматично вважається ідеальним. Для матриць більшого порядку програма розраховує суми стовпців, обчислює середнє геометричне та проводить нормування вектора пріоритетів.

Наступним кроком є безпосереднє «Визначення рівня погодженості». Якщо розраховане значення ВУ є недостатнім для даного типу даних (перевищує 0,10), алгоритм видає «Помилку» та вимагає уточнення даних від експертів. Якщо ж усі дані введені коректно і матриця є узгодженою, програма успішно виводить фінальний вектор пріоритетів P_i та значення ВУ.

3.5.2 Структура програмного забезпечення

Розроблений програмний комплекс має модульну структуру і функціонально розподілений на дві загальносистемні підсистеми (основні модулі) та шість

спеціалізованих допоміжних модулів (компонентів), що взаємодіють через єдине програмне середовище.

Загальносистемні підсистеми (основні модулі):

- **Модуль автентифікації та розмежування прав доступу**

Забезпечує реєстрацію користувачів(Рис 3.3), безпечний вхід до системи(Рис 3.4) та реалізацію моделі керування доступом на основі ролей (RBAC). Модуль здійснює генерацію та перевірку криптографічних токенів (JWT) для ідентифікації трьох категорій користувачів: Адміністратор, Експерт та Особа, що приймає рішення (ОПР).

- **Модуль навігації та керування сесіями користувачів**

Відповідає за збереження стану поточної сесії на стороні клієнта (localStorage), організацію безпечного міжмодульного обміну даними через захищені асинхронні HTTP-запити до вебсервера (fetchWithAuth) та коректне завершення роботи із системою.

Спеціалізовані компоненти (допоміжні модулі):

- **Модуль керування контентом оцінювання**

Інтегрується у робоче середовище експерта та дозволяє здійснювати динамічне формування простору рішень. Забезпечує функції додавання та модифікації об'єктів оцінювання (альтернативного програмного забезпечення), а також побудову ієрархічної структури критеріїв порівняння.(Рис 3.5)

- **Модуль формування матриць попарних порівнянь**

Реалізує інтерактивний механізм побудови матриць попарних порівнянь Сааті на основі визначених критеріїв та альтернатив. Програмно забезпечує автоматичне обчислення та заповнення дзеркальних (обернених) значень елементів матриці для оптимізації роботи експерта.(Рис.3.6)

- **Модуль розрахунку вагових коефіцієнтів та математичної валідації**

Являє собою обчислювальний блок, який розраховує вектори локальних пріоритетів методом геометричного середнього. Модуль автоматично здійснює перевірку математичної узгодженості експертних суджень, обчислюючи індекс та

відношення узгодженості (ВУ). У разі перевищення нормативного порогу (ВУ $>$ 0.10) система блокує внесення некоректних даних.(Рис 3.7)

- **Модуль аналізу та агрегації локальних пріоритетів**

Функціонує у межах інтерфейсу ОПР. Забезпечує імпорт, статистичне усереднення та структурування проміжних оцінок, отриманих від групи експертів, з подальшим відображенням зведеної матриці локальних пріоритетів альтернатив за кожним критерієм.(Рис 3.9)

- **Модуль глобального синтезу та генерації звітності**

Виконує фінальну процедуру ієрархічної згортки для обчислення глобальних пріоритетів альтернатив, здійснює їх підсумкове ранжування та визначає оптимальне рішення. Модуль містить підсистему документування для експорту розрахункових даних у формат CSV та інтегрований блок рев'юінгу для фіксації текстових зауважень і відгуків користувачів.(Рис 3.9)

- **Модуль адміністрування та аудиту облікових записів**

Забезпечує функціонал робочого простору системного адміністратора. Призначений для централізованого моніторингу зареєстрованих учасників процесу оцінювання, верифікації їхнього системного статусу та аудиту професійних кваліфікацій експертів.(Рис 3.10)

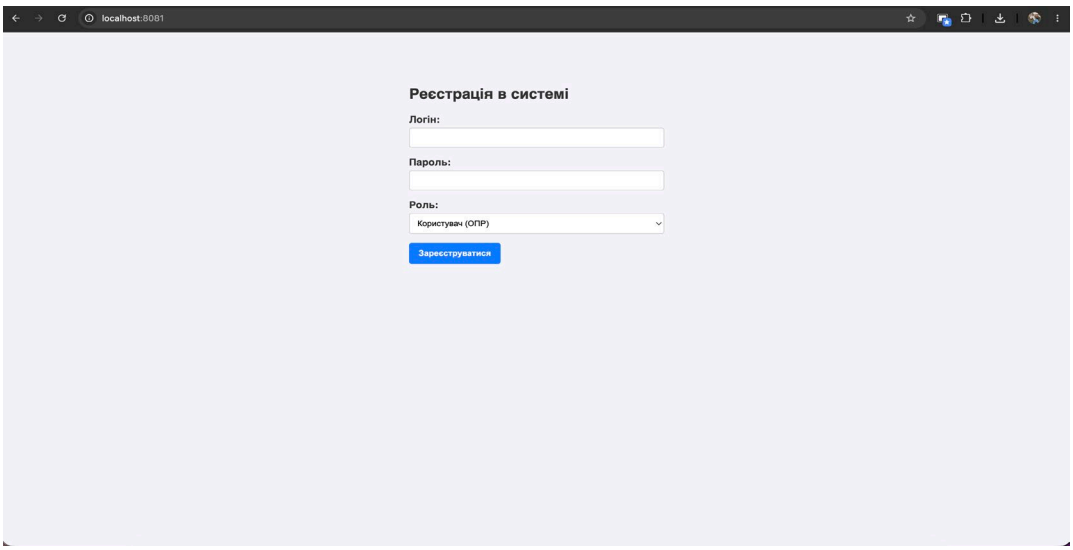


Рисунок 3.3 - Модуль реєстрації

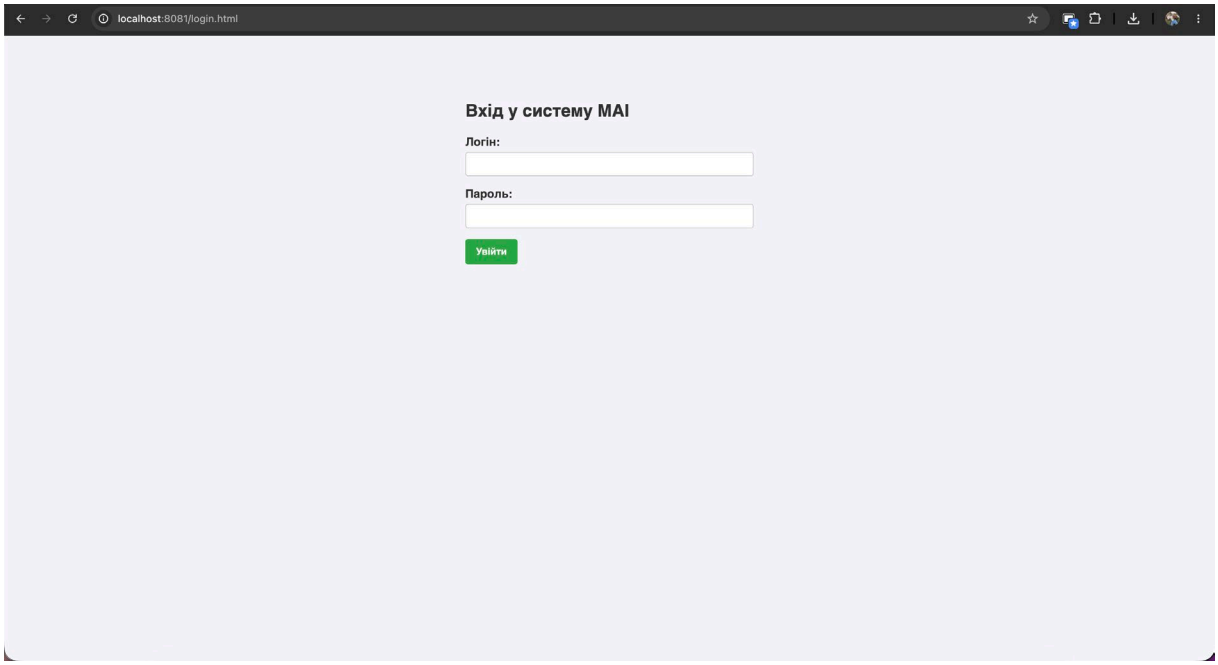


Рисунок 3.4 - Модуль логіну

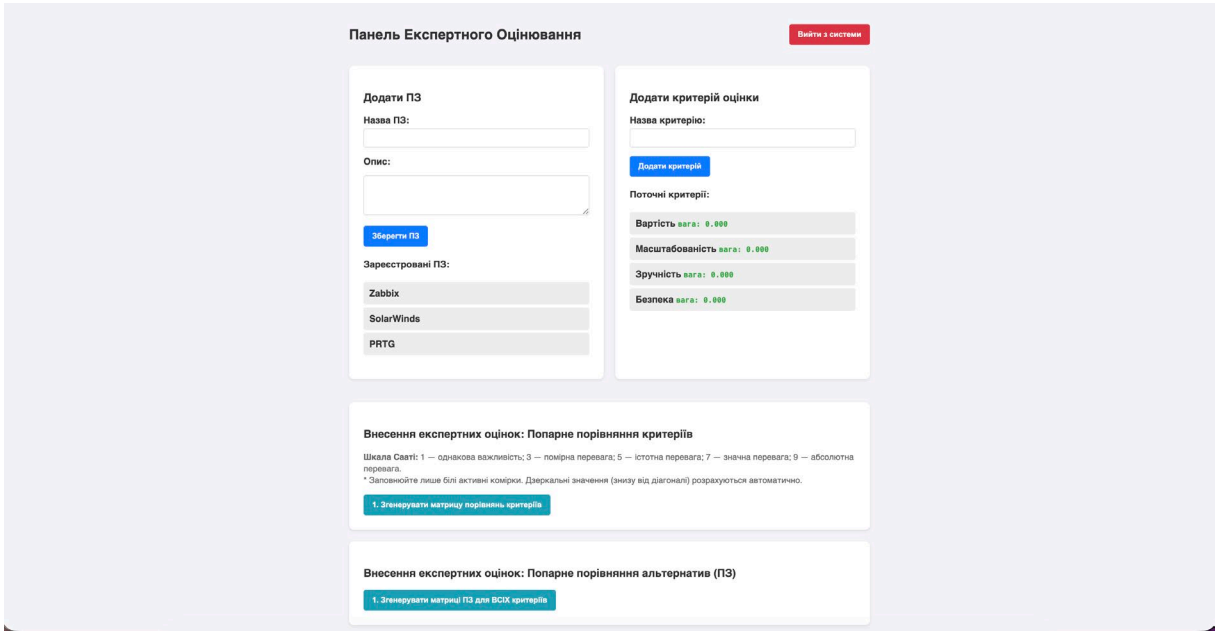


Рисунок 3.5 - Модуль вікна попарних порівнянь та додавання критеріїв та ПЗ

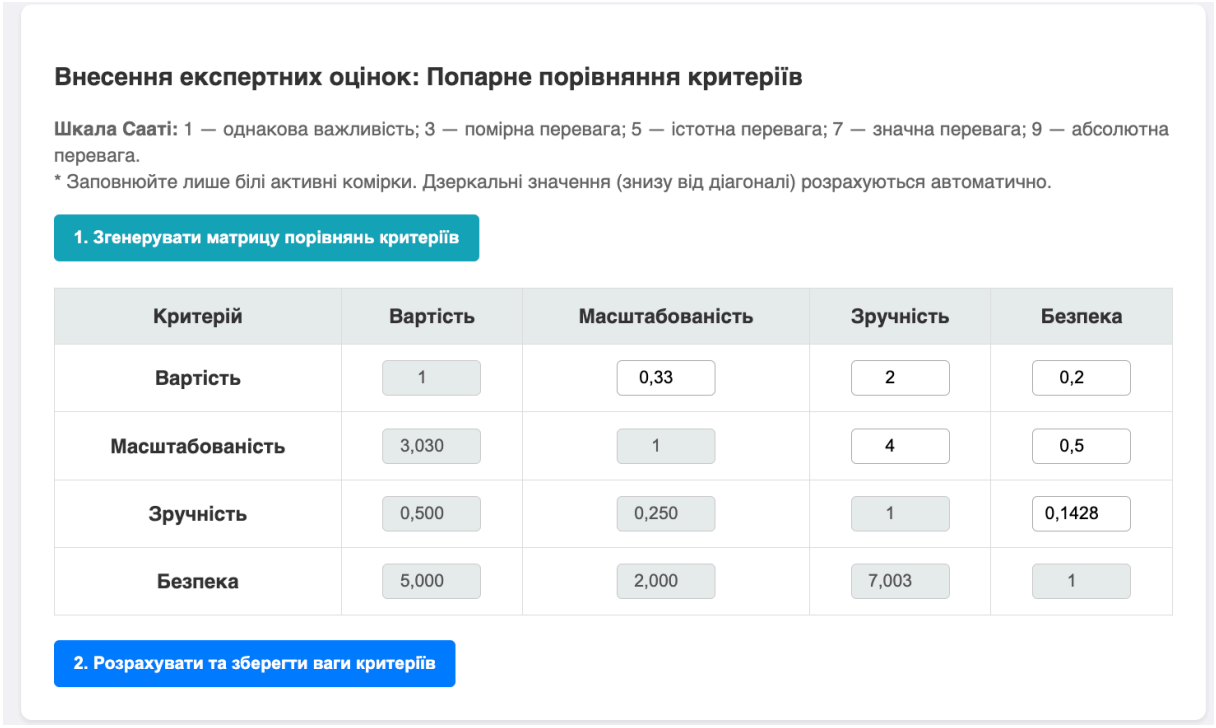


Рисунок 3.6 - Модуль попарного порівняння критеріїв

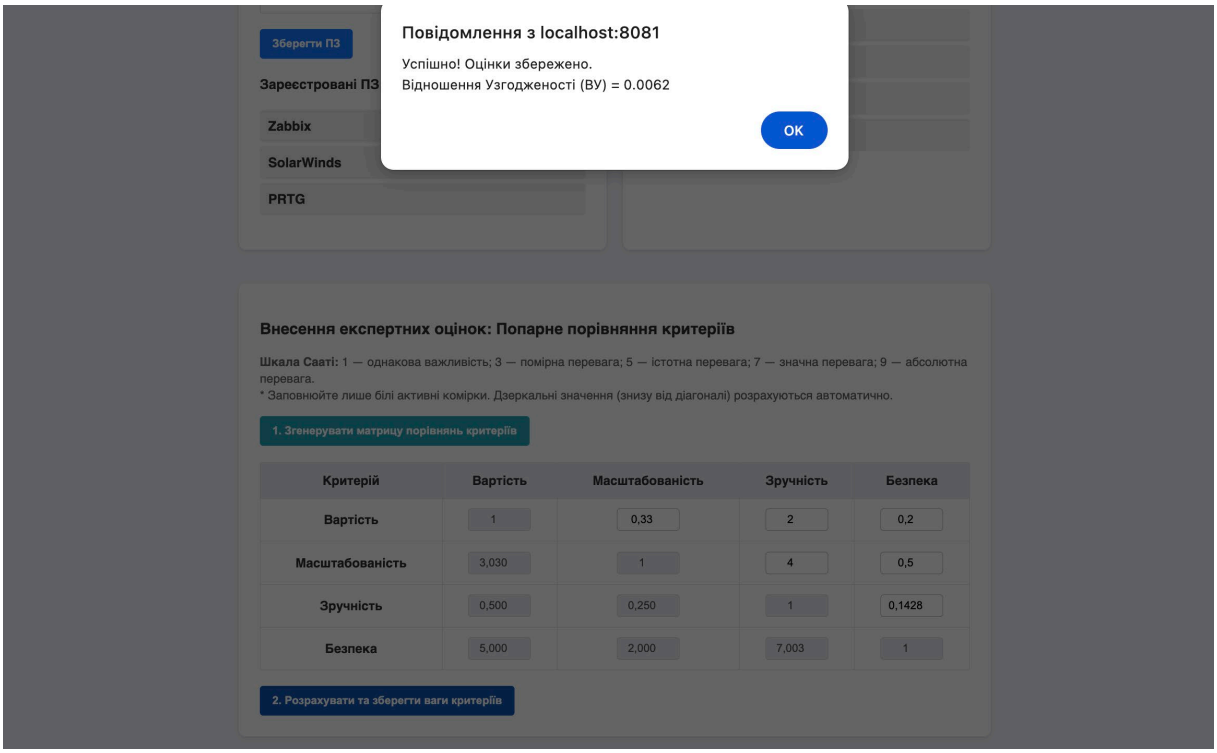


Рисунок 3.7 - Модуль розрахунку вагових коефіцієнтів та математичної валідації

Внесення експертних оцінок: Попарне порівняння альтернатив (ПЗ)

1. Згенерувати матриці ПЗ для ВСІХ критеріїв

За критерієм: "Вартість"

ПЗ	Zabbix	SolarWinds	PRTG
Zabbix	1	4	9
SolarWinds	0,250	1	2
PRTG	0,111	0,500	1

За критерієм: "Масштабованість"

ПЗ	Zabbix	SolarWinds	PRTG
Zabbix	1	5	3
SolarWinds	0,200	1	0,5
PRTG	0,333	2,000	1

За критерієм: "Зручність"

ПЗ	Zabbix	SolarWinds	PRTG
Zabbix	1	0,1428	0,33
SolarWinds	7,003	1	2
PRTG	3,030	0,500	1

За критерієм: "Безпека"

ПЗ	Zabbix	SolarWinds	PRTG
Zabbix	1	3	1
SolarWinds	0,333	1	0,25
PRTG	1	4,000	1

Рисунок 3.8 - Модуль попарного порівняння програмного забезпечення

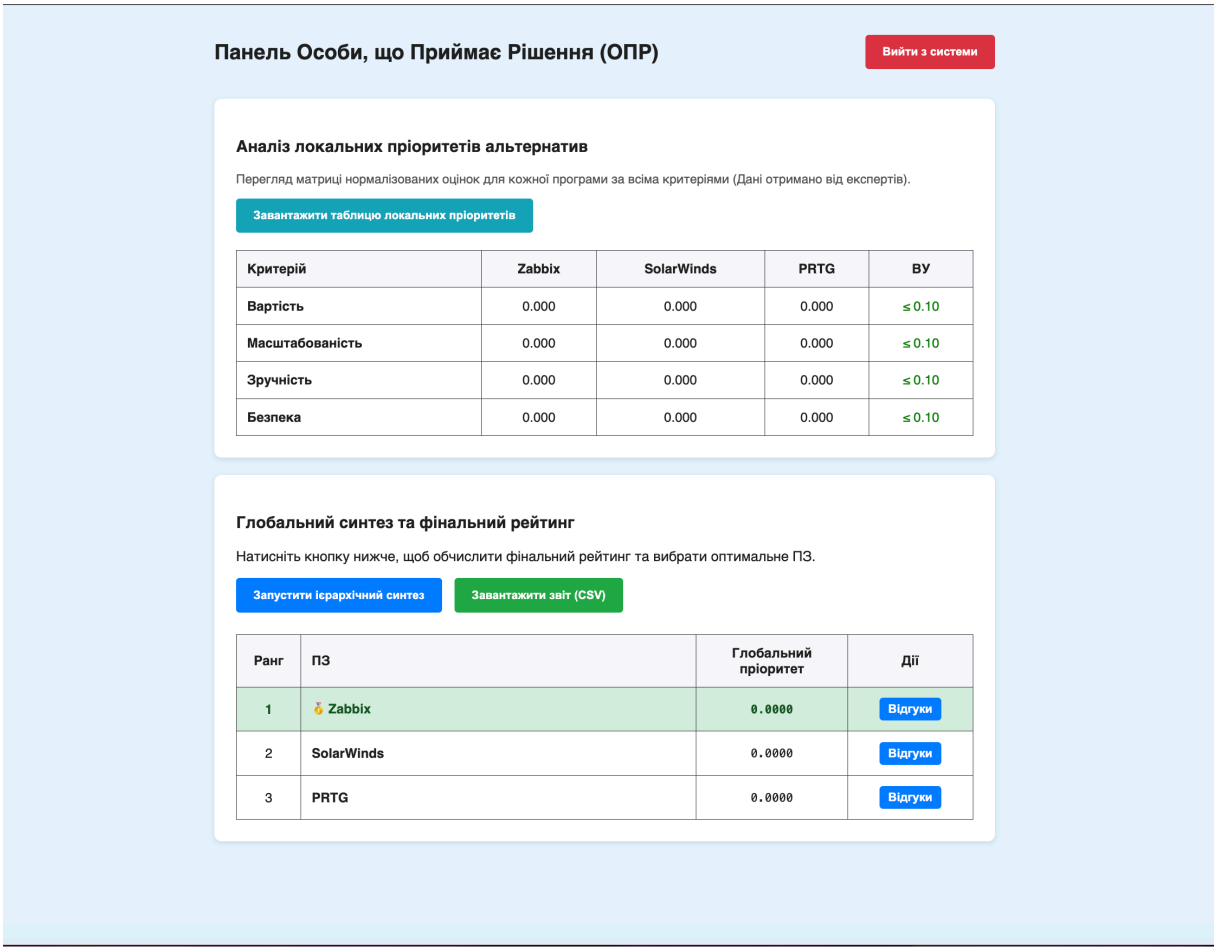


Рисунок 3.9 - Модуль аналізу локальних пріоритетів та глобального синтезу

Рисунок 3.10 - Модуль адміністрування та аудиту

3.5.3 Тестування програмного забезпечення

Щоб перевірити роботу модулів програми та їх взаємодії між собою, необхідно створити тест комплект (Табл 3.7)

Таблиця 3.7 Тестування програми

ID	Назва тест-кейсу	Кроки відтворення	Очікуваний результат
ТС-01	Реєстрація нового користувача (ОПР)	1. Відкрити сайт 2. Ввести унікальний логін та пароль. 3. Вибрати роль "Користувач (ОПР)". 4. Натиснути "Зареєструватися".	З'являється зелене повідомлення про успіх. Відбувається перенаправлення на меню логіну.
ТС-02	Реєстрація нового Експерта	1. На сторінці реєстрації вибрати роль "Експерт". 2. Перевірити появу поля "Кваліфікація". 3. Заповнити всі поля та натиснути "Зареєструватися".	Поле "Кваліфікація" стає обов'язковим. Реєстрація проходить успішно.
ТС-03	Авторизація користувача (Вхід)	1. На сторінці логіну ввести дані створеного Експерта. 2. Натиснути "Увійти".	Відбувається перенаправлення до меню експерта

ТС-04	Додавання нових альтернатив (ПЗ)	1. Увійти як Експерт. 2. У формі "Додати ПЗ" ввести назву та опис. 3. Натиснути "Зберегти ПЗ".	Нове ПЗ додається в базу даних та миттєво відображається у списку "Зареєстровані ПЗ".
ТС-05	Додавання критеріїв оцінки	1. Перебуваючи в панелі Експерта, у формі "Додати критерій" вказати назву. 2. Натиснути "Додати критерій".	Критерій з'являється у списку "Поточні критерії" з початковою вагою 0.000.
ТС-06	Генерація та заповнення матриці критеріїв	1. Натиснути "1. Згенерувати матрицю порівнянь критеріїв". 2. Внести оцінки (наприклад, 3 або 5) в активні комірки. 3. Перевірити автоматичний розрахунок дзеркальних комірок.	Дзеркальні комірки автоматично заповнюються оберненими значеннями (наприклад, 0.333 або 0.200).
ТС-07	Розрахунок та збереження ваг критеріїв	Натиснути "2.Розрахувати та зберегти ваги критеріїв" (при узгодженій матриці, де CR≤0.10).	З'являється повідомлення з успішним розрахунком та значенням ВУ (CR). Список критеріїв оновлює свої ваги.

ТС-08	Порівняння альтернатив (ПЗ) за критеріями	1. Натиснути "1. Згенерувати матриці ПЗ для ВСІХ критеріїв". 2. Заповнити матриці оцінками. 3. Натиснути "2. Розрахувати та зберегти оцінки для ВСІХ матриць".	Дані успішно записуються в таблицю software_assessments. Виводиться повідомлення про успішне збереження.
ТС-09	Перевірка захисту ролей (RBAC)	1. Спробувати зайти в меню користувача або адміністратора під акаунтом Експерта.	З’являється повідомлення "Доступ заборонено", додаток перенаправляє користувача на сторінку логіну
ТС-10	Перегляд локальних пріоритетів (ОПР)	1. Авторизуватися під роллю ОПР 2. Натиснути "Завантажити таблицю локальних пріоритетів".	З'являється зведена таблиця з оцінками, які внесли експерти, та поміткою про узгодженість (≤0.10).
ТС-11	Запуск ієрархічного синтезу (Розрахунок рейтингу)	1. У панелі ОПР натиснути "Запустити ієрархічний синтез".	З'являється таблиця результатів. Найкраще ПЗ підсвічується зеленим кольором та іконкою

ТС-12	Робота з відгуками та зауваженням и	1. У таблиці результатів ОПР натиснути кнопку "Відгуки" навпроти будь-якого ПЗ.2. Ввести текст у поле та натиснути "Зберегти відгук".	Відгук миттєво додається в блок нижче із зазначенням логіну ОПР, його ролі та дати.
ТС-13	Експорт фінального звіту в CSV	1. Після проведення синтезу в панелі ОПР натиснути кнопку "Завантажити звіт (CSV)".	Браузер автоматично завантажує файл ahp_report_YYYY-MM-DD.csv, що містить ранг, назву, опис та глобальний пріоритет.
ТС-14	Вихід із системи	1. Натиснути кнопку "Вийти з системи" у верхній шапці будь-якої панелі.	Користувача перенаправляє на сторінку логіну.

Проведемо тестування системи на основі математичних прикладів, розрахованих раніше.

Заходимо на веб-сайт програми. Відкривається меню реєстрації користувача. (Рис 3.11). Вводимо довільний логін та пароль, та натискаємо кнопку «Зареєструватися».

Після натискання кнопки, під нею з’являється зелено повідомлення що реєстрація пройшла успішно, і після цього відбувається перенаправлення на сторінку логіну.

Після цього повертаємось на сторінку реєстрації та реєструємо експерта в системі (Рис 3.12). Для експерта необхідно вибрати його кваліфікацію. У випадку якщо експерт не введе свою кваліфікацію, з’явиться повідомлення «Заповніть це поле» (Рис 3.13), яке не запустить користувача далі, поки він не введе кваліфікацію.

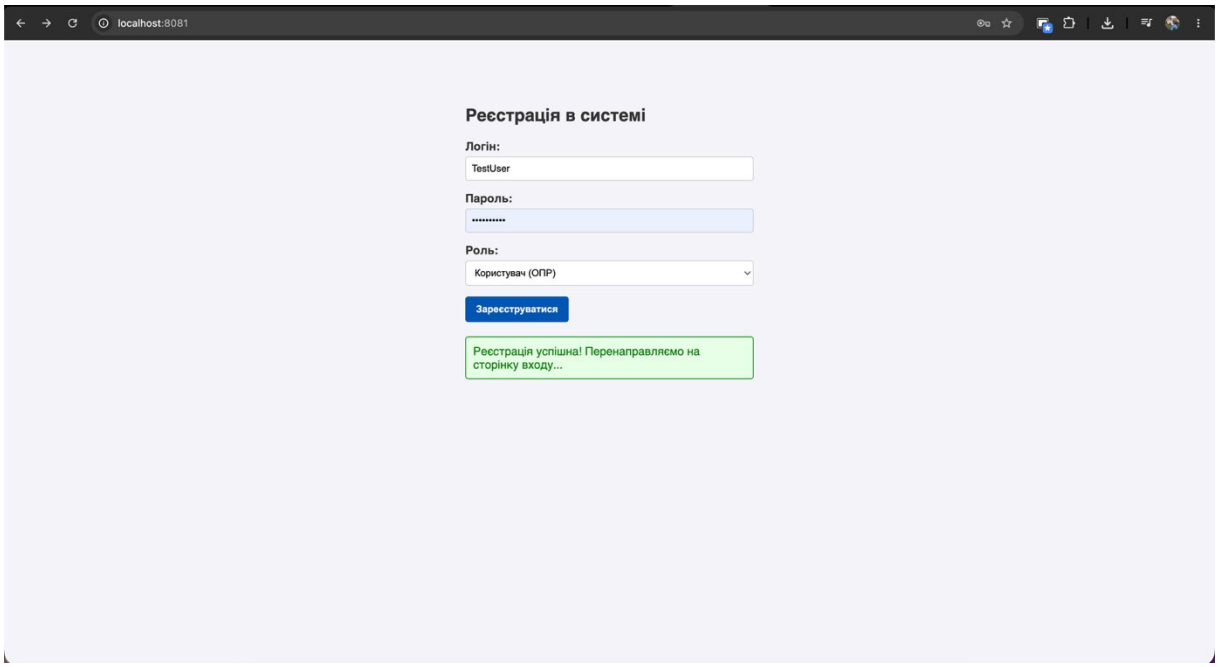


Рисунок 3.11 - Сторінка реєстрації

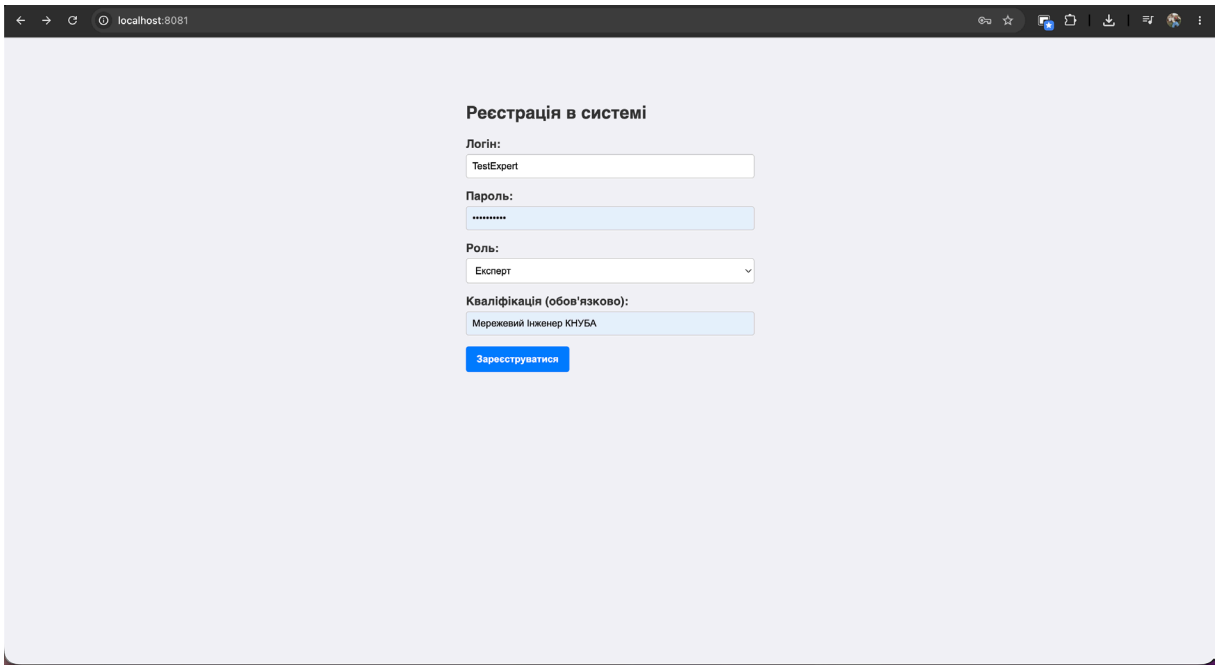


Рисунок 3.12 - Реєстрація експерта

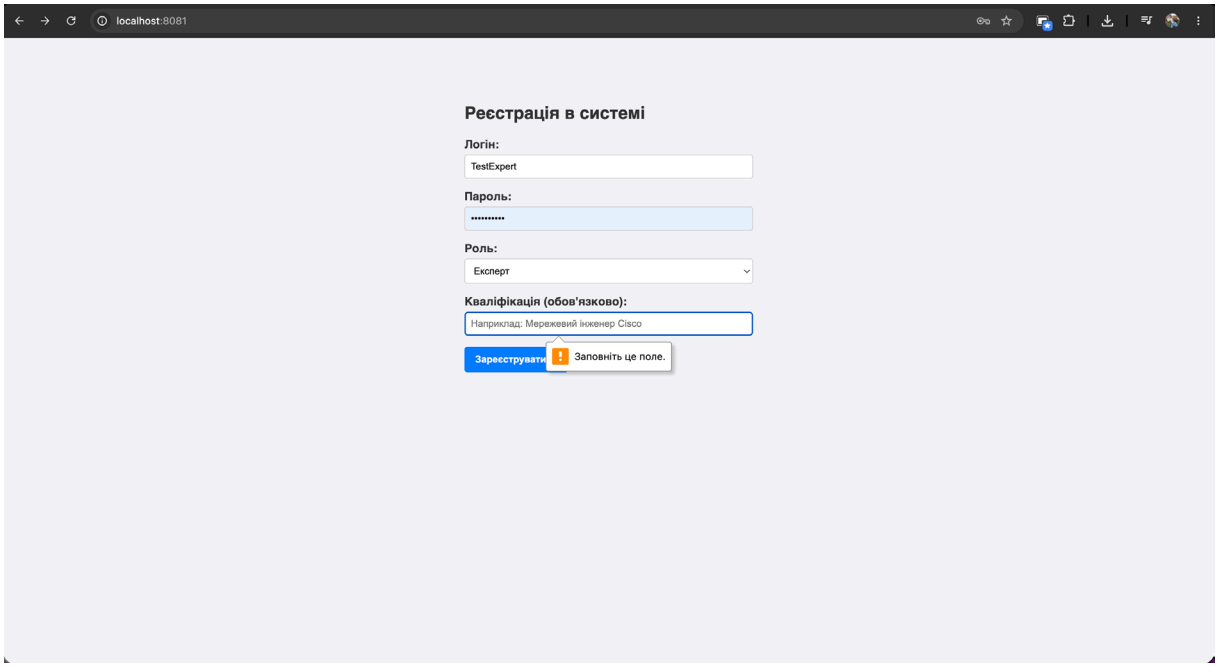


Рисунок 3.13 - Повідомлення «Заповніть це поле»

На сторінці логіну(Рис. 3.14) вписуємо дані експерта та натискаємо кнопку «Увійти». Якщо дані, які вводить користувач некоректні, то з’являється червоне повідомлення з помилкою (Рис.3.15).

Якщо дані експерта коректні, відкривається сторінка з функціоналом експерта.

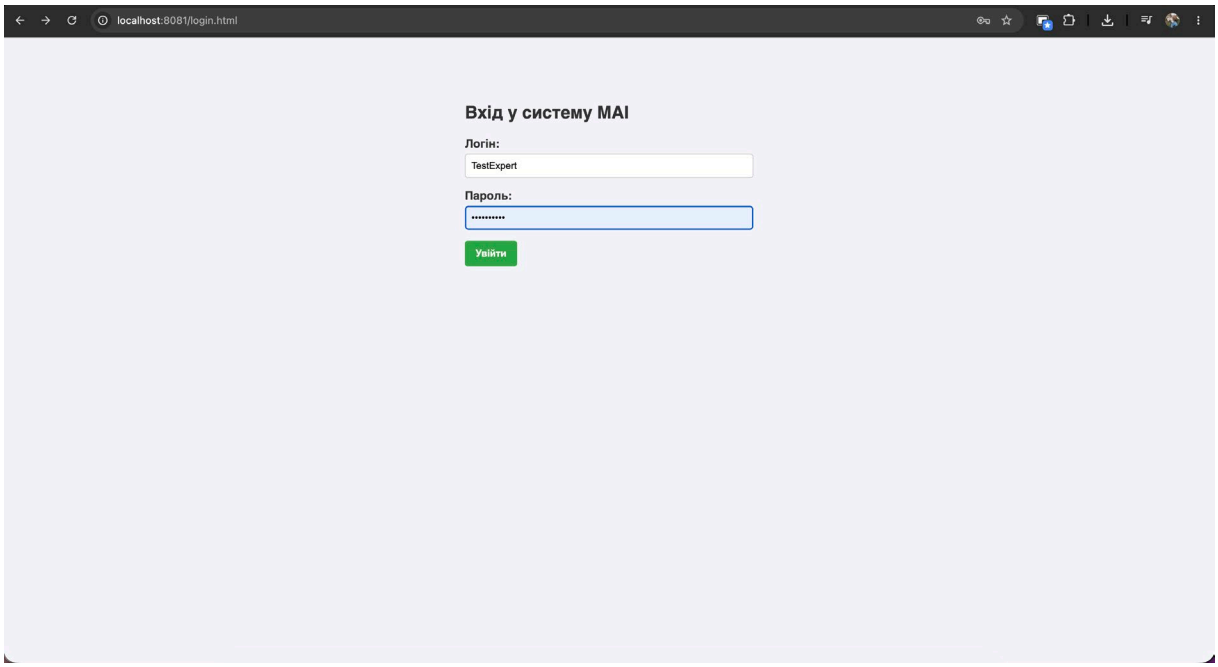


Рисунок 3.14 - Сторінка логіну

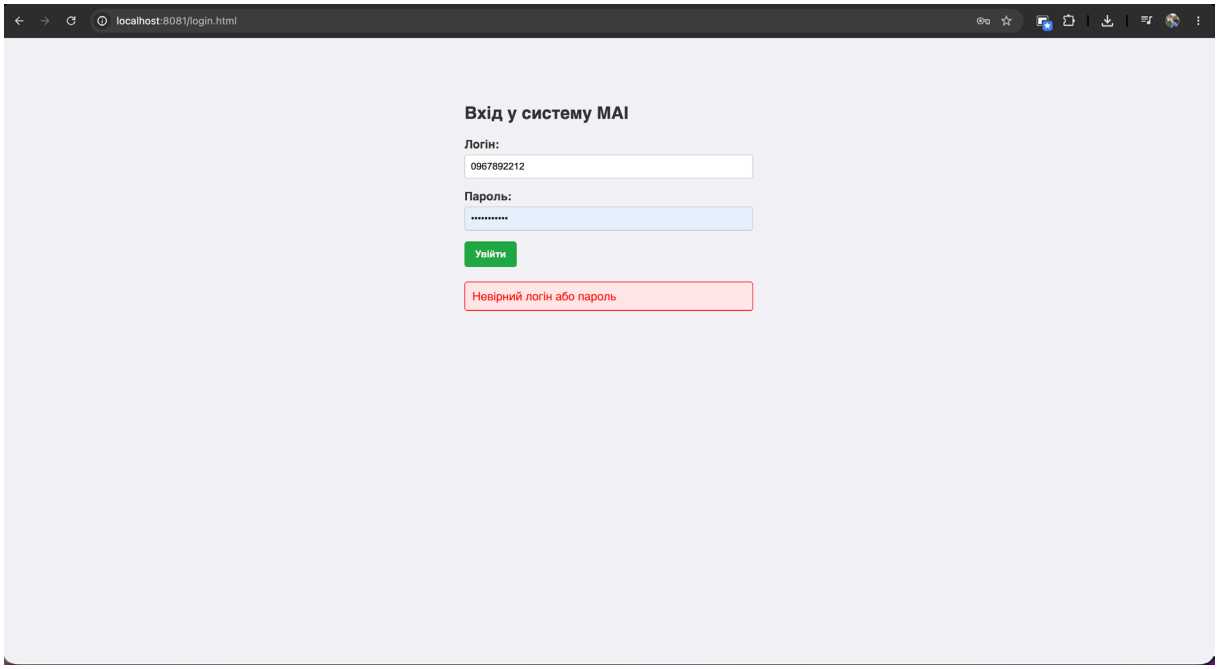


Рисунок 3.15 - Помилка входу

На сторінці експерта (Рис. 3.16) додаємо програмне забезпечення та критерії, яке ми будемо використовувати в розрахунках (Рис. 3.17).

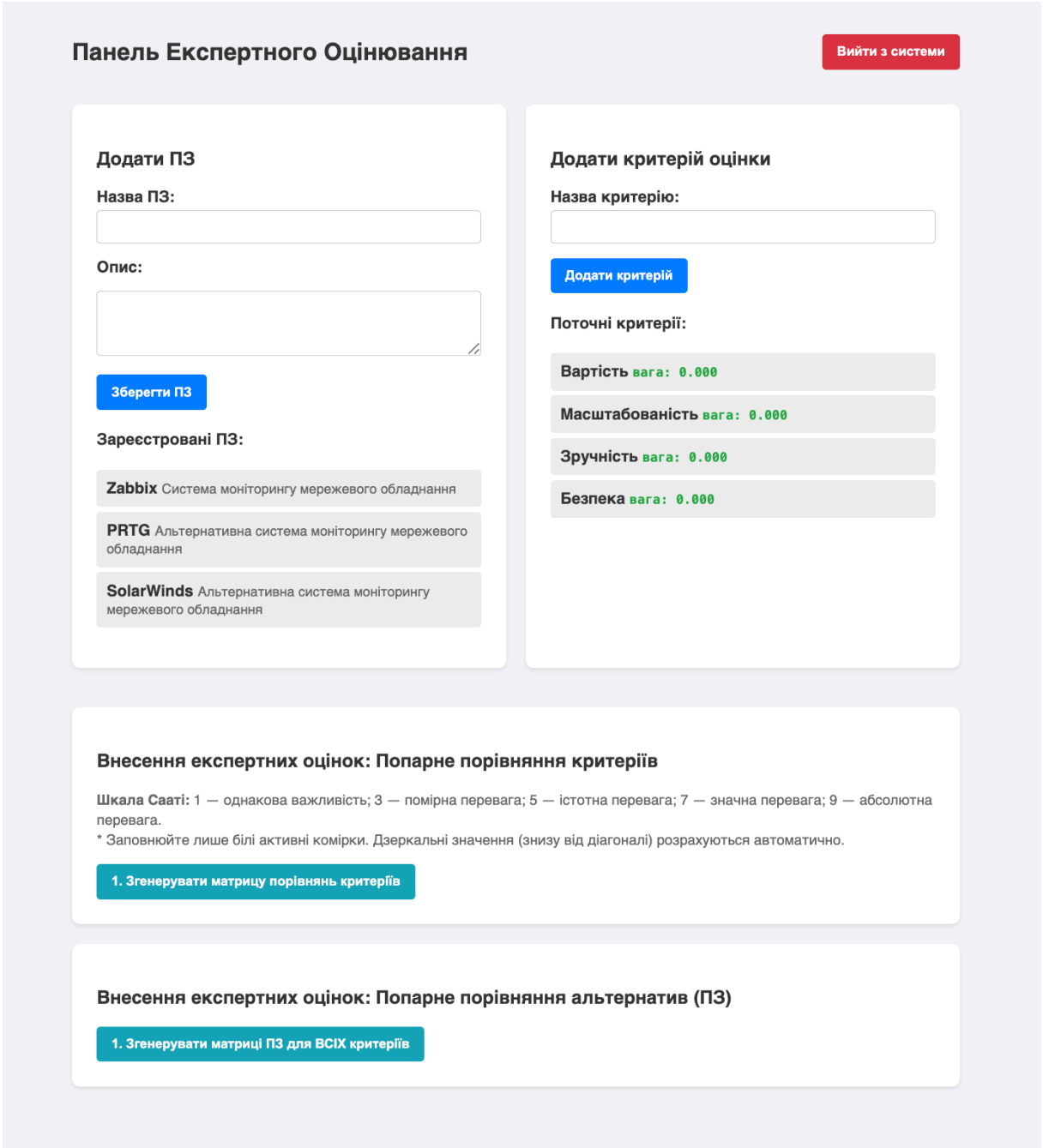


Рисунок 3.16 - Сторінка експерта

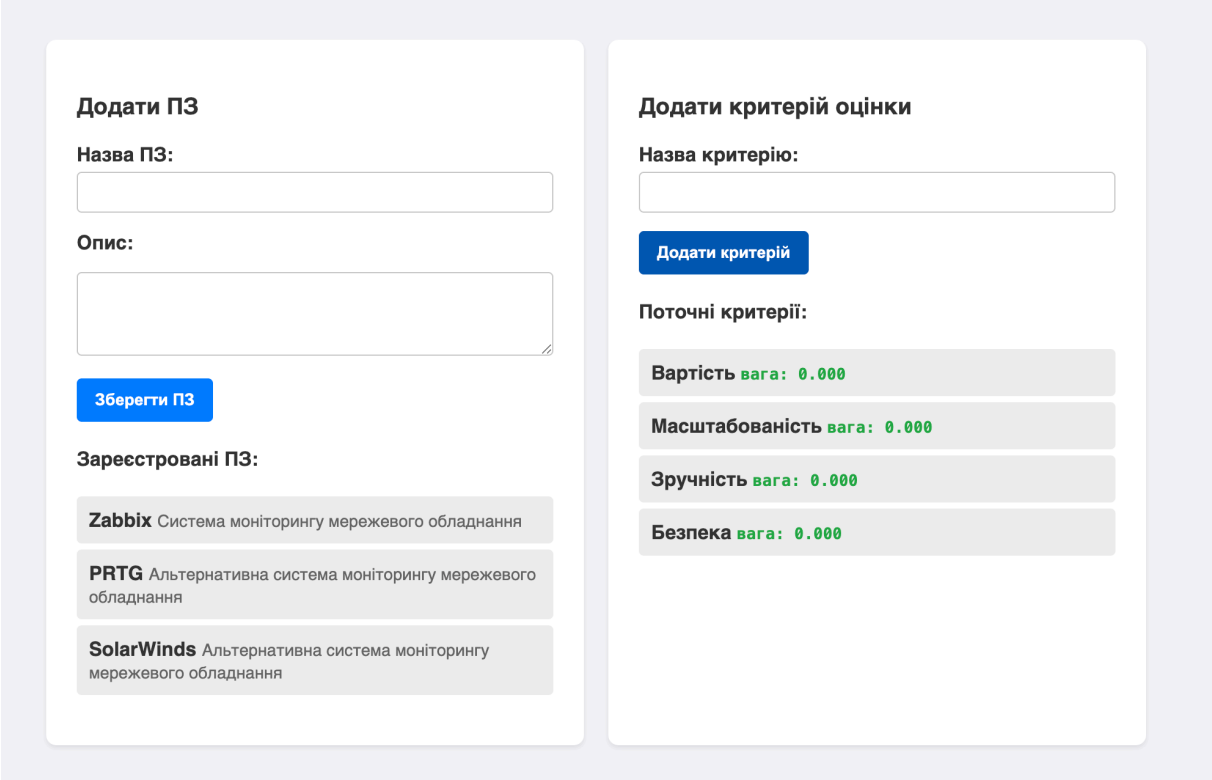


Рисунок 3.17 - Заповнення даних про програмне забезпечення та критерії

Гортаємо вниз та знаходимо меню для заповнення матриці попарного порівняння, та натискаємо «Згенерувати матрицю попарних порівнянь критеріїв».

З’являється матриця (Рис. 3.18) в яку ми заносимо оцінку за шкалою Сааті. Після занесення оцінок натискаємо «Розрахувати та зберегти ваги критеріїв».

У випадку якщо ВУ ≤ 0.1, то сайт виведе повідомлення що оцінки збережено, і покаже обчислене ВУ(Рис. 3.19).

Якщо ВУ > 0.1, з’явиться повідомлення що ВУ неузгоджене, і поверне вас назад до матриці (Рис. 3.20).

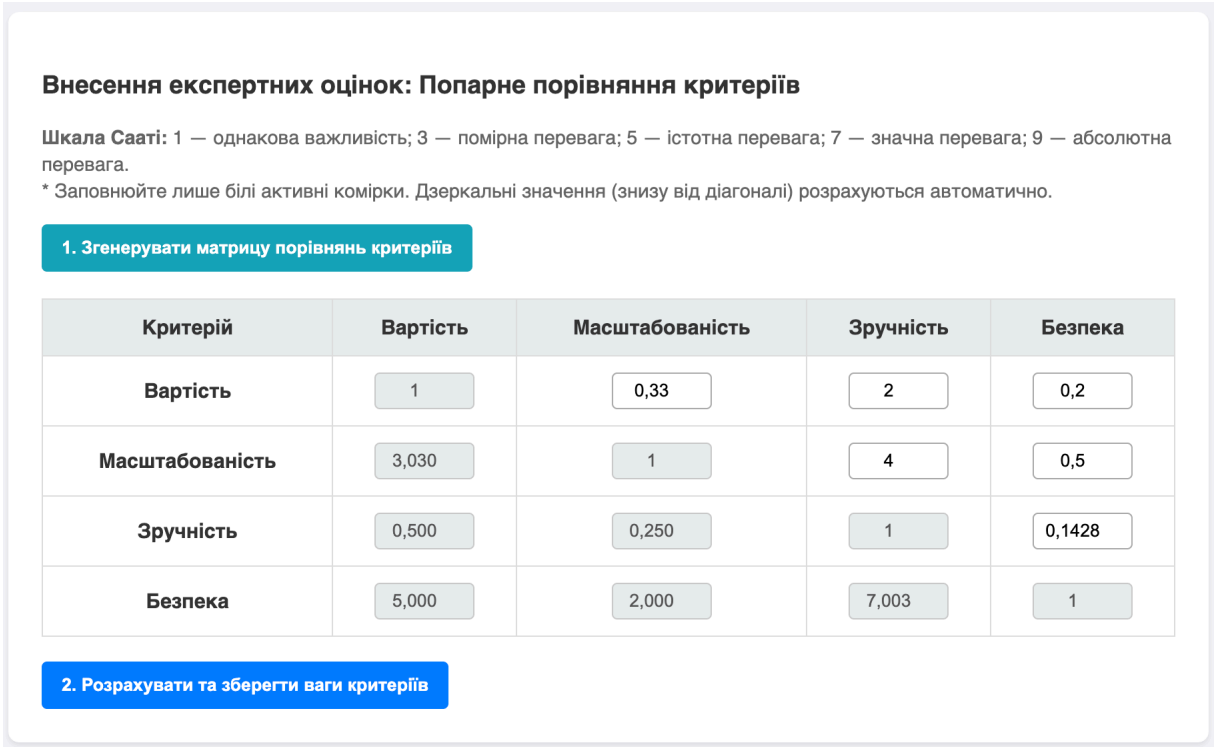


Рисунок 3.18 - Заповнена матриця попарного порівняння критеріїв

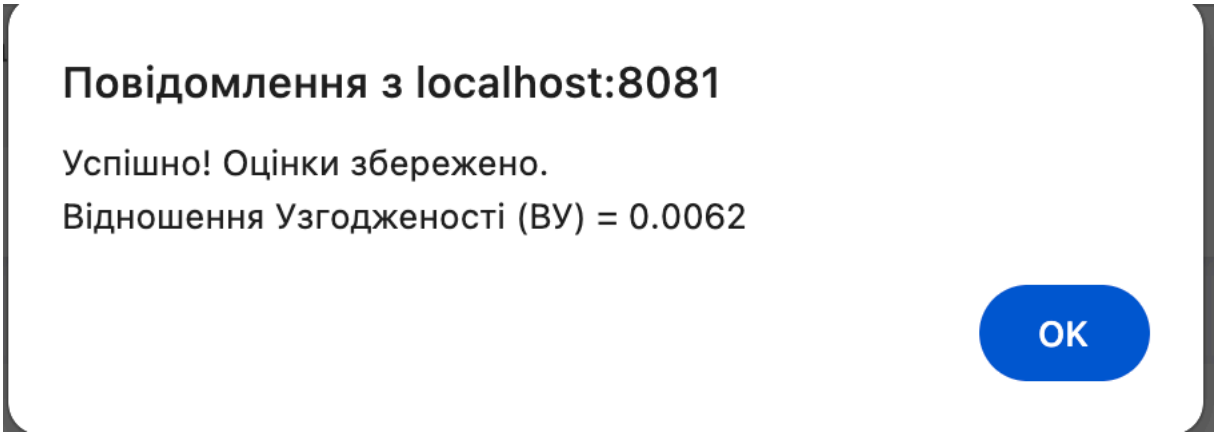


Рисунок 3.19 - Повідомлення про збереження оцінок

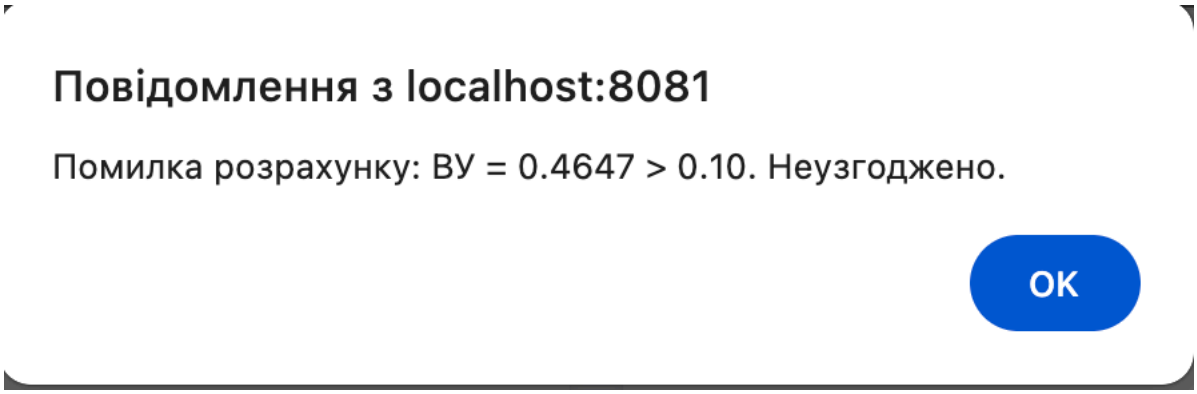


Рисунок 3.20 - Повідомлення про неузгоджене ВУ

Після успішного занесення оцінок в матрицю порівнянь критеріїв, переходимо до матриці порівняння програмного забезпечення (Рис. 3.21).

Заносимо оцінки та натискаємо «Розрахувати та зберегти оцінки для всіх матриць». Якщо ВУ узгоджене з'явиться повідомлення про успішне збереження оцінок. В іншому випадку виведеться повідомлення про неузгоджене ВУ.

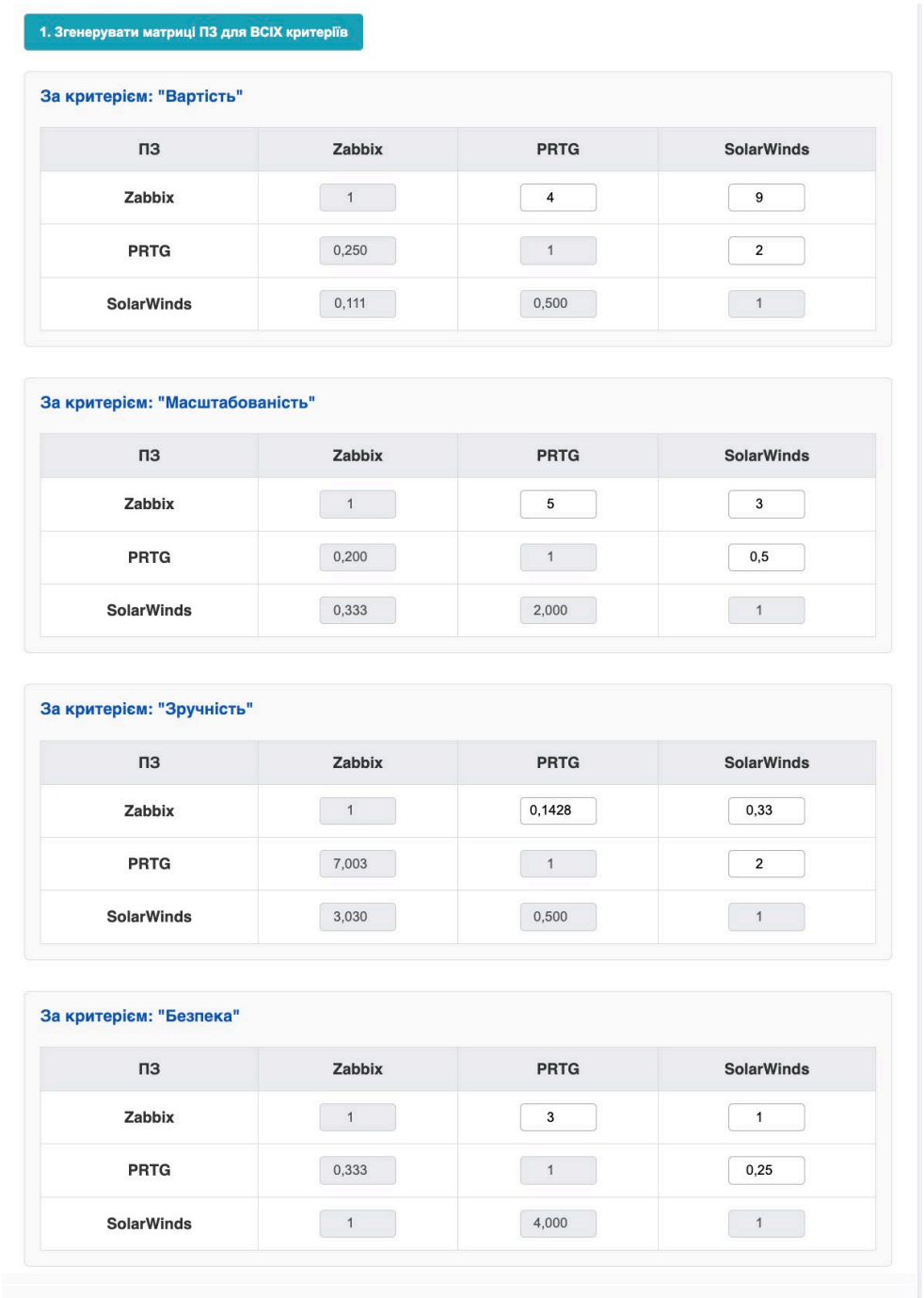


Рисунок 3.21 - Заповнені матриці попарних порівнянь програмного забезпечення

Спробуємо зайти на сторінку адміністратора під акаунтом експерта. Бачимо, що на сторінку не переходить, тому що у експерта недостатньо прав. (Рис. 3.22).

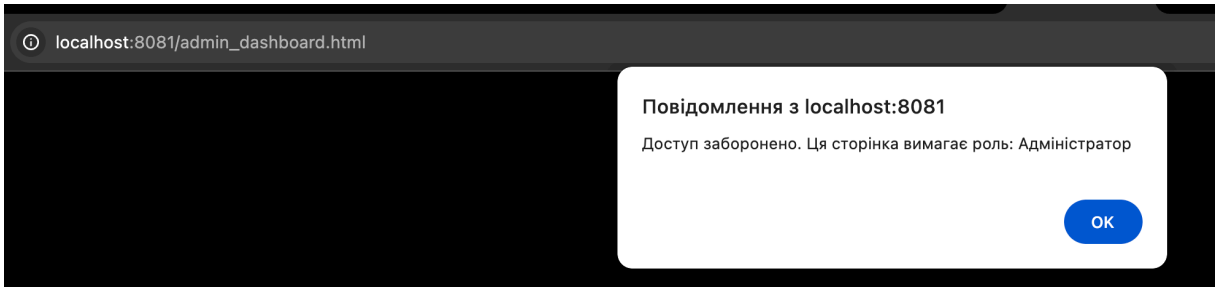


Рисунок 3.22 - Повідомлення про заборону доступу для експерта

Заходимо через профіль користувача(ОПР), і відкривається меню панелі ієрархічного синтезу, та таблиці локальних пріоритетів (Рис. 3.23).

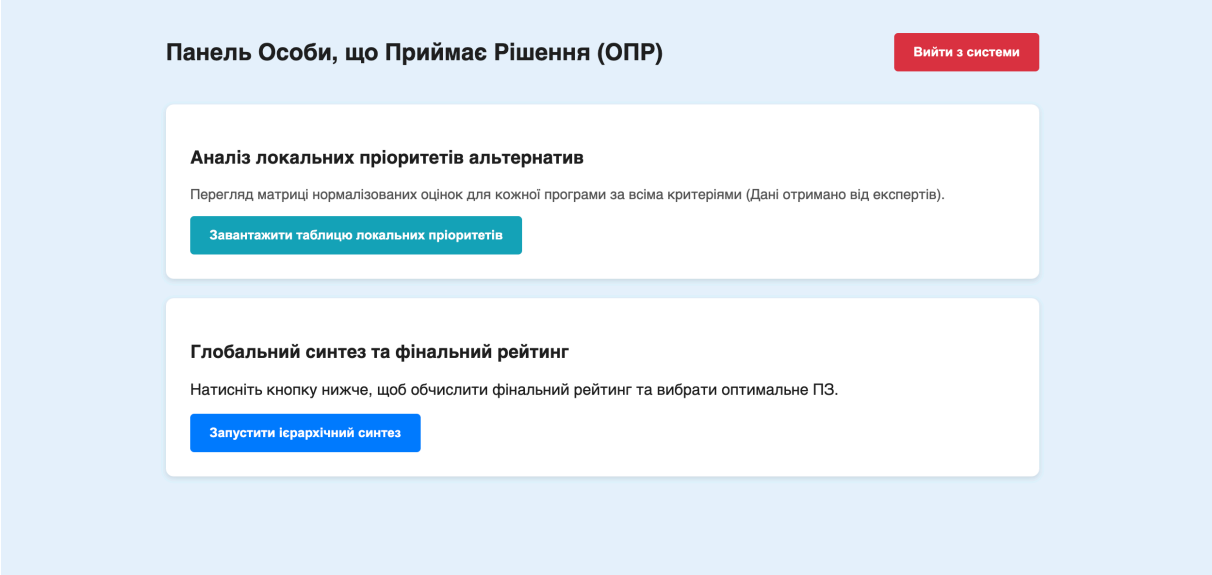


Рисунок 3.23 - Панель ієрархічного синтезу та таблиці локальних пріоритетів

Натискаємо «Завантажити таблицю локальних пріоритетів», з’являється готова матриця локальних пріоритетів (Рис. 3.24).

Аналіз локальних пріоритетів альтернатив

Перегляд матриці нормалізованих оцінок для кожної програми за всіма критеріями (Дані отримано від експертів).

Завантажити таблицю локальних пріоритетів

Критерій	Zabbix	PRTG	SolarWinds	ВУ
Вартість	0.738	0.177	0.085	≤ 0.10
Масштабованість	0.648	0.122	0.230	≤ 0.10
Зручність	0.092	0.615	0.293	≤ 0.10
Безпека	0.416	0.126	0.458	≤ 0.10

Рисунок 3.24 - Готова матриця локальних пріоритетів

Після цього гортаємо вниз та натискаємо «Запустити ієрархічний синтез», з’являється таблиця з найкращим рішенням (Рис. 3.25).

Глобальний синтез та фінальний рейтинг

Натисніть кнопку нижче, щоб обчислити фінальний рейтинг та вибрати оптимальне ПЗ.

Запустити ієрархічний синтез | Завантажити звіт (CSV)

Ранг	ПЗ	Глобальний пріоритет	Дії
1	Zabbix Система моніторингу мережевого обладнання	0.4971	Відгуки
2	SolarWinds Альтернативна система моніторингу мережевого обладнання	0.3395	Відгуки
3	PRTG Альтернативна система моніторингу мережевого обладнання	0.1634	Відгуки

Рис. 3.25 - Глобальний синтез та фінальний рейтинг

Натискаємо кнопку «Відгуки», та залишаємо відгук на потрібне програмне забезпечення (Рис. 3.26).

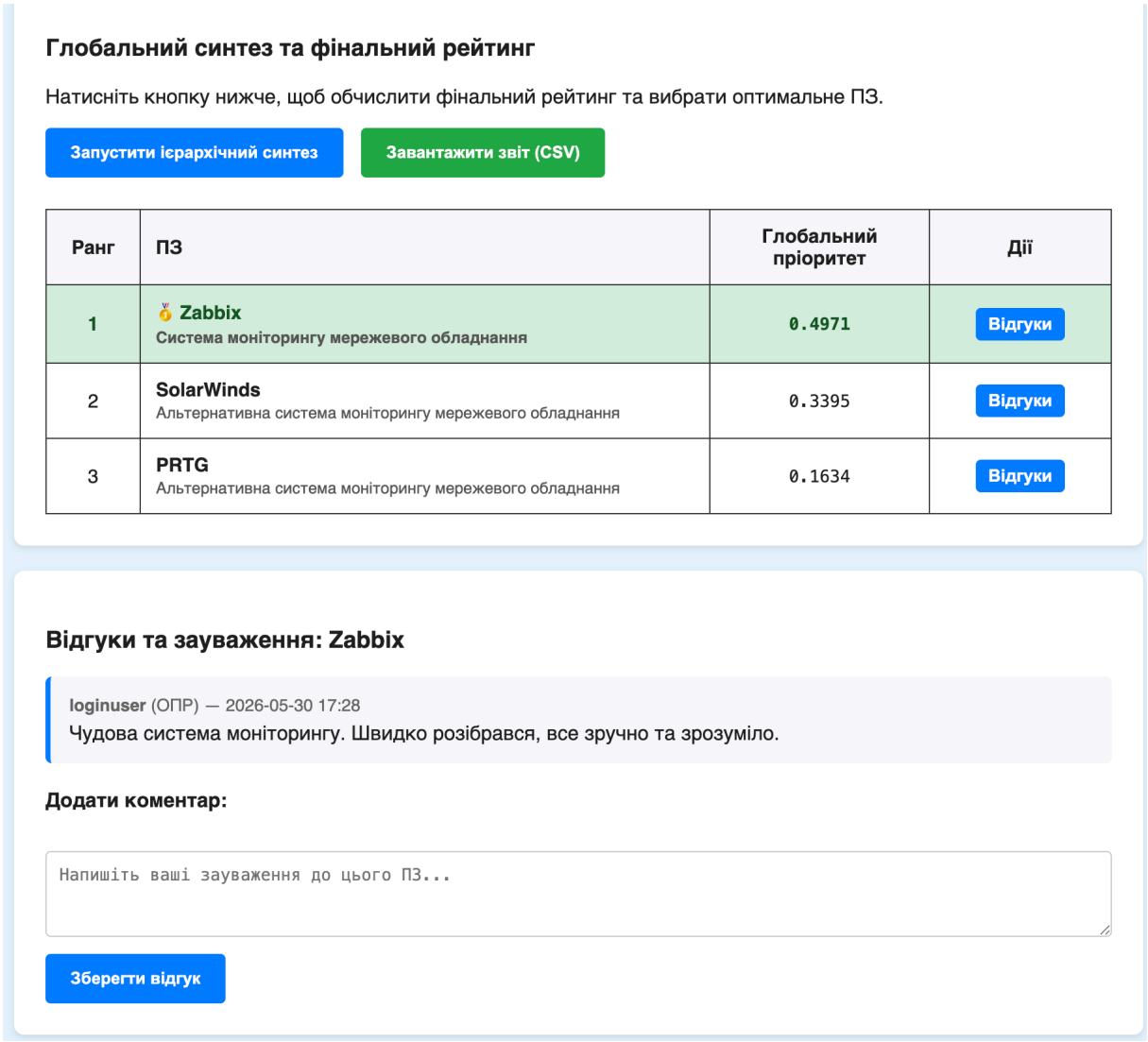


Рисунок 3.26 - Система відгуків

Натискаємо «Завантажити звіт (CSV)». Бачимо, що файл завантажився (Рис 3.27). Якщо відкрити файл, побачимо записані дані по глобальному синтезу (Рис. 3.28).

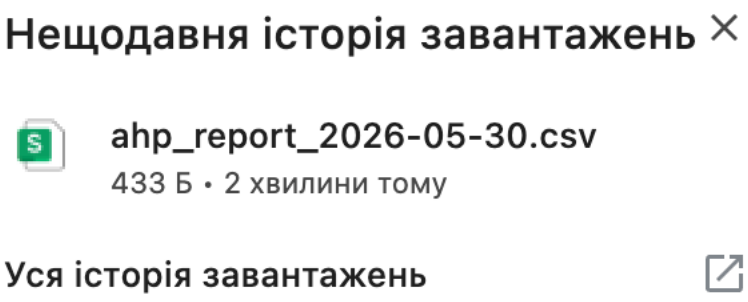


Рисунок 3.27 - Успішне завантаження файлу

A
Ранг,Назва ПЗ,Опис,Глобальний пріоритет
1,"Zabbix","Система моніторингу мережевого обладнання",0.4971
2,"SolarWinds","Альтернативна система моніторингу мережевого обладнання",0.3395
3,"PRTG","Альтернативна система моніторингу мережевого обладнання",0.1634

Рисунок 3.28 - Запис у CSV файлі

Після успішної роботи програми, натискаємо «Вийти з системи».

Після того як натиснули кнопку, нас повертає у меню логіну.

Висновки до Розділу 3

Для тестування працездатності створеного програмного продукту було сформовано репрезентативну вибірку альтернатив, до якої увійшли три концептуально різні платформи: відкрита система з розподіленою архітектурою Zabbix 7.0 LTS, комерційне монолітне рішення Paessler PRTG Network Monitor та модульна платформа корпоративного рівня SolarWinds NPM. Такий вибір дозволив перевірити гнучкість системи при роботі з альтернативами, що мають полярно різні характеристики.

В інтерфейсі розробленої системи було побудовано ієрархію критеріїв оцінки, яка відображає сучасні потреби управління IT-інфраструктурою, зокрема: сукупну вартість володіння, масштабованість та швидкодію, зручність розгортання і використання, а також надійність і безпеку. Під час симуляції реальної роботи системи в умовах, наближених до експлуатаційних, у ролі експерта було здійснено покрокове заповнення матриць попарних порівнянь Сааті безпосередньо в програмному середовищі.

На прикладі оцінювання масштабованості та швидкодії було математично обґрунтовано перевагу архітектури Zabbix над комерційними аналогами. Програмний алгоритм системи успішно обробив введені експертні дані, здійснив розрахунок власних векторів матриць, автоматично перевірив логічну узгодженість суджень експерта та виконав алгоритм ієрархічного синтезу. У результаті система згенерувала підсумковий звіт із розрахованими глобальними ваговими коефіцієнтами для кожної альтернативи.

Таким чином, практична апробація повністю підтвердила працездатність розробленого програмного забезпечення, довівши його здатність автоматизувати складні математичні обчислення, нівелювати суб'єктивність людського фактора та надавати особі, що приймає рішення, прозорий і математично обґрунтований результат для вибору оптимального мережевого програмного забезпечення.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну науково-практичну задачу, яка полягає у розробці та реалізації інформаційної системи підтримки прийняття рішень для об'єктивного багатокритеріального оцінювання та вибору програмного забезпечення управління корпоративними комп'ютерними мережами. Проведений на початку дослідження аналіз предметної області підтвердив, що сучасні IT-інфраструктури характеризуються високим ступенем гетерогенності, через що ручне адміністрування стає фізично неможливим і призводить до високого рівня аварійності через людський фактор. Наявність на ринку великої кількості архітектурно та комерційно різноманітних рішень унеможливлює вибір оптимального програмного забезпечення без застосування строгих математичних інструментів.

Для вирішення цієї проблеми було спроєктовано об'єктно-орієнтовану архітектуру інформаційної системи з використанням інструментарію UML. Побудовані діаграми прецедентів, діяльності, послідовностей та класів дозволили сформувати логічну структуру програмного забезпечення, а розроблена рольова модель доступу забезпечила чітке розмежування функцій між системним адміністратором, експертом та користувачем. Фундаментальним математичним ядром системи було обрано метод аналізу ієрархій Томаса Сааті. В межах програмного продукту успішно реалізовано алгоритми обчислення векторів локальних та глобальних пріоритетів на основі матриць попарних порівнянь, а також впроваджено важливу функцію автоматичної перевірки відношення узгодженості для контролю логічності експертних суджень.

Практичну реалізацію та апробацію розробленого програмного забезпечення було здійснено на реальній задачі вибору системи моніторингу серед таких альтернатив, як Zabbix 7.0, PRTG Network Monitor та SolarWinds NPM. Під час тестування програмний продукт безпомилково виконав усі етапи математичного розрахунку, від валідації введених експертних матриць до ієрархічного синтезу, та згенерував підсумковий рейтинг. Отримані результати підтверджують, що

розроблене програмне забезпечення є готовим дієвим інструментом, здатним перетворювати суб'єктивні технічні оцінки на строгий, кількісно вимірюваний результат. Таким чином, мету дипломної роботи досягнуто повністю, поставлені завдання виконано в повному обсязі, а створена інформаційна система має високу практичну цінність і може бути впроваджена в IT-департаментах компаній для оптимізації та стандартизації процесів закупівлі мережевого програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Suresh Reddy Thati. Configuration and compliance automation in modern networks: a framework for enhanced security and operational efficiency. *World journal of advanced engineering technology and sciences*. 2025. Т. 15, № 2. С. 2513–2519. URL: https://doi.org/10.30574/wjaets.2025.15.2.0613.

2. Goel A., Rahulamathavan Y. A comparative survey of centralised and decentralised identity management systems: analysing scalability, security, and feasibility. *Future internet*. 2024. Т. 17, № 1. С. 1. URL: https://doi.org/10.3390/fi17010001.

3. Network Configuration Management (NCM) Tools - ManageEngine Network Configuration Manager. *ManageEngine*. URL: https://www.manageengine.com/network-configuration-manager/.

4. What is FCAPS?. *ZPE Systems*. URL: https://zpesystems.com/what-is-fcaps/.

5. Single Point of Failure (SPOF) | System Design. *AlgoMaster.io - Master Software Engineering Interviews*. URL: https://algomaster.io/learn/system-design/single-point-of-failure-spof.

6. Resource Center | SolarWinds. *Observability, Database, and IT Service Management | SolarWinds*. URL: https://www.solarwinds.com/resources.

7. Cisco Catalyst Center User Guide, Release 3.2.x. *Cisco*. URL: https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/network-automation-and-management/catalyst-center/3-2-x/user-guide/cisco-catalyst-center-user-guide-3-2-x.html.

8. Shrubbery Networks, Inc. - RANCID. *Shrubbery Networks*. URL: https://www.shrubbery.net/rancid/.

9. GitHub - ytti/oxidized: Oxidized is a network device configuration backup tool. It's a RANCID replacement!. *GitHub*. URL: https://github.com/ytti/oxidized.

10. Unimus. *Unimus by NetCore j.s.a. | Network Automation, Configuration Backup & Change Management*. URL: https://unimus.net/features.html.

11. Ansible Community | Ansible documentation. URL: https://docs.ansible.com/.

12. *Welcome to NAPALM's documentation! – NAPALM 3 documentation*. URL: https://napalm.readthedocs.io/en/latest/.

13. Chawla S. Active Directory Series: Active Directory Fundamentals. *Cobalt: Offensive Security Services*. URL: https://www.cobalt.io/blog/active-directory-series-active-directory-fundamentals.

14. OpenLDAP vs. FreeIPA vs. Active Directory: Choosing the Right Solution. *DoHost - Affordable Premium Hosting Services*. URL: https://dohost.us/index.php/2025/10/10/openldap-vs-freeipa-vs-active-directory-choosing-the-right-solution/.

15. Leal M. Implementing Samba 4. Packt Publishing, Limited, 2014.

16. Shukla S., Jain K. Rise of Identity and Access Management with Microsoft Security. *International Journal on Advances in Engineering, Technology and Science (IJAETS)*. 2024. Т. 5, № 1. С. 1–7. URL: https://doi.org/10.5281/zenodo.10621038.

17. JumpCloud vs Okta Comparison Guide 2026. *Siit - The AI Service Desk*. URL: https://www.siit.io/tools/comparison/jumpcloud-vs-okta.

18. Introducing Zabbix 7.0 LTS: Enhanced Performance and Scalability for Enterprise Monitoring. *ATS Group*. URL: https://theatsgroup.com/ats-blog/introducing-zabbix-7-0-lts/.

19. What's new in Zabbix 7.0 LTS. *Zabbix: The enterprise-class open source observability solution*. URL: https://www.zabbix.com/whats_new_7_0.

20. PRTG Network Monitor – All-in-one network monitoring tool. *Paessler - The Monitoring Experts*. URL: https://www.paessler.com/prtg/prtg-network-monitor.

21. Математичні моделі задач. Метод аналізу ієрархій / І. К.-Д. О. та ін. 2021. URL: https://doi.org/10.5281/zenodo.5769999.

22. The Use Case Diagram / M. Seidl та ін. *UML @ Classroom*. Cham, 2015. С. 23–47. URL: https://doi.org/10.1007/978-3-319-12742-2_3.

23. Osadcha K. P., Khromyshev O. V. Розв’язання математичних задач засобами мови програмування Python. *Ukrainian Journal of Educational Studies and Information Technology*. 2017. Т. 5, № 1. С. 231–235. URL: https://doi.org/10.32919/10.32919/uesit.2017.01.231-235.

24. DevDocs – FastAPI documentation. *DevDocs API Documentation*. URL: https://devdocs.io/fastapi/.

25. Mayer G. E. T., Awesomeness J. JavaScript: JavaScript Awesomeness Book. CreateSpace Independent Publishing Platform, 2017. 68 с.

26. Douglas K., Douglas S. PostgreSQL. Sams, 2003. 816 с.

Додаток А

Файл main.py

```
import os
import math
import jwt
from datetime import datetime, timedelta
from typing import Optional, List, Tuple
from fastapi import FastAPI, HTTPException, Depends
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from passlib.context import CryptContext
import psycopg2
from psycopg2.extras import RealDictCursor
from dotenv import load_dotenv

load_dotenv()

class UserRegister(BaseModel):
    login: str
    password: str
    role: str
    qualification: Optional[str] = None

class UserLogin(BaseModel):
    login: str
    password: str

class SoftwareCreate(BaseModel):
    name: str
    description: Optional[str] = None

class CriteriaCreate(BaseModel):
    name: str
    parent_id: Optional[int] = None

class CriteriaMatrixInput(BaseModel):
    criteria_ids: List[int]
    matrix: List[List[float]]

class SoftwareMatrixInput(BaseModel):
    software_ids: List[int]
    criteria_id: int
    matrix: List[List[float]]

class ReviewCreate(BaseModel):
    review_text: str

class DatabaseManager:
    def __init__(self):
        self.config = {
```

```
            "dbname": os.getenv("DB_NAME", "diplomaa_DB"),
            "user": os.getenv("DB_USER", "postgres"),
            "password": os.getenv("DB_PASSWORD", "156325"),
            "host": os.getenv("DB_HOST", "localhost"),
            "port": os.getenv("DB_PORT", "5432")
        }

    def get_connection(self):
        return psycopg2.connect(**self.config, cursor_factory=RealDictCursor)

class UserRepository:
    def __init__(self, db: DatabaseManager):
        self.db = db

    def get_user_by_login(self, login: str) -> Optional[dict]:
        with self.db.get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT * FROM users WHERE login = %s;", (login,))
                return cur.fetchone()

    def create_user(self, user_data: UserRegister, hashed_password: str):
        with self.db.get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute(
                    "INSERT INTO users (login, password_hash, role, qualification) VALUES (%s, %s, %s, %s);",
                    (user_data.login, hashed_password, user_data.role, user_data.qualification)
                )
            conn.commit()

class AHPRepository:
    def __init__(self, db: DatabaseManager):
        self.db = db

    def update_criteria_weights(self, criteria_ids: List[int], weights: List[float]):
        with self.db.get_connection() as conn:
            with conn.cursor() as cur:
                for idx, cr_id in enumerate(criteria_ids):
                    cur.execute("UPDATE criteria SET weight = %s WHERE id = %s;", (weights[idx], cr_id))
            conn.commit()

    def update_software_assessments(self, software_ids: List[int], criteria_id: int, weights: List[float], expert_id: int):
        with self.db.get_connection() as conn:
            with conn.cursor() as cur:
                for idx, sw_id in enumerate(software_ids):
                    cur.execute("""
                        INSERT INTO software_assessments (software_id, criteria_id, expert_id, local_weight)
                        VALUES (%s, %s, %s, %s)
                        ON CONFLICT (software_id, criteria_id, expert_id)
                        DO UPDATE SET local_weight = EXCLUDED.local_weight;
                    """, (sw_id, criteria_id, expert_id, weights[idx]))
            conn.commit()

class AuthService:
```

```
    SECRET_KEY = os.getenv("JWT_SECRET_KEY", "super_secure_diploma_key")
    ALGORITHM = "HS256"
    EXPIRE_MINUTES = 120

    def __init__(self, user_repo: UserRepository):
        self.pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
        self.user_repo = user_repo

    def hash_password(self, password: str) -> str:
        return self.pwd_context.hash(password)

    def verify_password(self, plain_password: str, hashed_password: str) -> bool:
        return self.pwd_context.verify(plain_password, hashed_password)

    def create_token(self, user_id: int, role: str) -> str:
        expire = datetime.utcnow() + timedelta(minutes=self.EXPIRE_MINUTES)
        payload = {"sub": str(user_id), "role": role, "exp": expire}
        return jwt.encode(payload, self.SECRET_KEY, algorithm=self.ALGORITHM)

    def register_user(self, user: UserRegister):
        if user.role not in ['Експерт', 'ОПР', 'Адміністратор']:
            raise HTTPException(status_code=400, detail="Некоректна роль.")
        if user.role == 'Експерт' and not user.qualification:
            raise HTTPException(status_code=400, detail="Для Експерта необхідна кваліфікація.")

        if self.user_repo.get_user_by_login(user.login):
            raise HTTPException(status_code=400, detail="Користувач вже існує.")

        self.user_repo.create_user(user, self.hash_password(user.password))

class AHPService:
    RI_VALUES = {1: 0.0, 2: 0.0, 3: 0.58, 4: 0.90, 5: 1.12, 6: 1.24, 7: 1.32, 8: 1.41, 9: 1.45, 10: 1.49}

    @classmethod
    def calculate_weights(cls, matrix: List[List[float]]) -> Tuple[List[float], float]:
        n = len(matrix)
        if n <= 2: return [1.0 / n] * n, 0.0

        row_products = [pow(math.prod(row), 1.0/n) for row in matrix]
        sum_products = sum(row_products)
        weights = [p / sum_products for p in row_products]

        col_sums = [sum(matrix[i][j] for i in range(n)) for j in range(n)]
        lambda_max = sum(col_sums[j] * weights[j] for j in range(n))

        ci = (lambda_max - n) / (n - 1)
        ri = cls.RI_VALUES.get(n, 1.49)
        cr = ci / ri if ri > 0 else 0.0

        return weights, cr

db_manager = DatabaseManager()
security = HTTPBearer()
```

```
def get_user_repo() -> UserRepository: return UserRepository(db_manager)
def get_ahp_repo() -> AHPRepository: return AHPRepository(db_manager)
def get_auth_service(repo: UserRepository = Depends(get_user_repo)) -> AuthService: return AuthService(repo)

def get_current_user(credentials: HTTPAuthorizationCredentials = Depends(security),
                     auth_service: AuthService = Depends(get_auth_service)):
    try:
        payload = jwt.decode(credentials.credentials, auth_service.SECRET_KEY, algorithms=[auth_service.ALGORITHM])
        return {"user_id": int(payload.get("sub")), "role": payload.get("role")}
    except jwt.PyJWTError:
        raise HTTPException(status_code=401, detail="Невалідний токен")

app = FastAPI(title="СППР ООП Версія")
app.add_middleware(CORSMiddleware, allow_origins=["*"], allow_credentials=True, allow_methods=["*"],
allow_headers=["*"])

@app.post("/api/register", status_code=201)
def register(user: UserRegister, auth: AuthService = Depends(get_auth_service)):
    auth.register_user(user)
    return {"message": "Успішно"}

@app.post("/api/login")
def login(user: UserLogin, auth: AuthService = Depends(get_auth_service), repo: UserRepository =
Depends(get_user_repo)):
    db_user = repo.get_user_by_login(user.login)
    if not db_user or not auth.verify_password(user.password, db_user["password_hash"]):
        raise HTTPException(status_code=401, detail="Невірний логін або пароль")

    token = auth.create_token(db_user["id"], db_user["role"])
    return {"access_token": token, "token_type": "bearer", "role": db_user["role"]}

@app.post("/api/logout")
def logout():
    return {"message": "Успішно вийшли з системи"}

@app.get("/api/admin/users")
def get_all_users(user: dict = Depends(get_current_user)):
    if user["role"] != "Адміністратор":
        raise HTTPException(status_code=403, detail="Доступ заборонено")

    with db_manager.get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute("SELECT id, login, role, qualification FROM users ORDER BY id ASC;")
            return cur.fetchall()

@app.get("/api/software")
def get_software(user: dict = Depends(get_current_user)):
    with db_manager.get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute("SELECT id, name, description FROM software ORDER BY id ASC;")
            return cur.fetchall()
```

```
@app.post("/api/software")
def add_software(sw: SoftwareCreate, user: dict = Depends(get_current_user)):
    if user["role"] != "Експерт": raise HTTPException(status_code=403)
    try:
        with db_manager.get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("INSERT INTO software (name, description) VALUES (%s, %s) RETURNING id;", (sw.name,
sw.description))
                new_id = cur.fetchone()['id']
                conn.commit()
                return {"id": new_id}
    except psycopg2.errors.UniqueViolation:
        raise HTTPException(status_code=400, detail="ПЗ з такою назвою вже існує!")

@app.get("/api/criteria")
def get_criteria(user: dict = Depends(get_current_user)):
    with db_manager.get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute("SELECT id, name, weight FROM criteria ORDER BY id ASC;")
            return cur.fetchall()

@app.post("/api/criteria")
def add_criteria(cr: CriteriaCreate, user: dict = Depends(get_current_user)):
    if user["role"] != "Експерт": raise HTTPException(status_code=403)
    try:
        with db_manager.get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("INSERT INTO criteria (name, parent_id) VALUES (%s, %s) RETURNING id;", (cr.name,
cr.parent_id))
                conn.commit()
                return {"message": "Критерій додано"}
    except psycopg2.errors.UniqueViolation:
        raise HTTPException(status_code=400, detail="Критерій з такою назвою вже існує!")

@app.post("/api/ahp/criteria-weights")
def save_criteria_weights(
    data: CriteriaMatrixInput,
    user: dict = Depends(get_current_user),
    repo: AHPRepository = Depends(get_ahp_repo)
):
    if user["role"] != "Експерт":
        raise HTTPException(status_code=403, detail="Тільки для експертів")

    weights, cr = AHPService.calculate_weights(data.matrix)
    if cr > 0.10:
        raise HTTPException(status_code=400, detail=f"ВУ = {cr:.4f} > 0.10. Неузгоджено")

    repo.update_criteria_weights(data.criteria_ids, weights)
    return {"message": "Ваги критерії в збережено", "consistency_ratio": cr}

@app.post("/api/ahp/software-weights")
def save_software_weights(
    data: SoftwareMatrixInput,
```

```
    user: dict = Depends(get_current_user),
    repo: AHPRepository = Depends(get_ahp_repo)
):
    if user["role"] != "Експерт":
        raise HTTPException(status_code=403, detail="Тільки для експертів")

    weights, cr = AHPService.calculate_weights(data.matrix)
    if cr > 0.10:
        raise HTTPException(status_code=400, detail=f"ВУ = {cr:.4f} > 0.10. Неузгоджено")

    repo.update_software_assessments(data.software_ids, data.criteria_id, weights, user["user_id"])
    return {"message": "Збережено", "consistency_ratio": cr}

@app.get("/api/ahp/local-weights-report")
def get_local_weights_report(user: dict = Depends(get_current_user)):
    if user["role"] != "ОПР":
        raise HTTPException(status_code=403)

    with db_manager.get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute("SELECT id, name FROM criteria ORDER BY id ASC;")
            criteria = cur.fetchall()
            cur.execute("SELECT id, name FROM software ORDER BY id ASC;")
            software = cur.fetchall()

            report = []
            for cr in criteria:
                row = {"criterion": cr['name'], "weights": {}}
                for sw in software:
                    cur.execute("""
                        SELECT AVG(local_weight) as avg_weight
                        FROM software_assessments
                        WHERE criteria_id = %s AND software_id = %s;
                    """, (cr['id'], sw['id']))
                    res = cur.fetchone()
                    row["weights"][sw['name']] = float(res['avg_weight']) if res and res['avg_weight'] is not None else 0.0
                report.append(row)

            return {
                "software_names": [s['name'] for s in software],
                "report": report
            }

@app.post("/api/ahp/synthesize")
def run_hierarchical_synthesis(user: dict = Depends(get_current_user)):
    if user["role"] != "ОПР":
        raise HTTPException(status_code=403)

    with db_manager.get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute("SELECT id, weight FROM criteria;")
            criteria = cur.fetchall()
            cur.execute("SELECT id, name, description FROM software;")
```

```
        software_list = cur.fetchall()

        results = []
        for sw in software_list:
            sw_id = sw['id']
            global_priority = 0.0

            for cr in criteria:
                cur.execute("""
                    SELECT AVG(local_weight) as avg_weight
                    FROM software_assessments
                    WHERE software_id = %s AND criteria_id = %s;
                """, (sw_id, cr['id']))
                assessment = cur.fetchone()
                local_weight = float(assessment['avg_weight']) if assessment and assessment['avg_weight'] is not None else 0.0

                cr_weight = cr['weight'] if cr['weight'] is not None else 0.0
                global_priority += (cr_weight * local_weight)

            cur.execute("UPDATE software SET global_priority = %s WHERE id = %s;", (global_priority, sw_id))
            results.append({
                "id": sw_id,
                "name": sw['name'],
                "description": sw['description'],
                "global_priority": global_priority
            })

        conn.commit()
        results.sort(key=lambda x: x["global_priority"], reverse=True)
        return results

@app.post("/api/software/{sw_id}/reviews")
def add_review(sw_id: int, review: ReviewCreate, user: dict = Depends(get_current_user)):
    with db_manager.get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute(
                "INSERT INTO reviews (software_id, author_id, review_text) VALUES (%s, %s, %s);",
                (sw_id, user["user_id"], review.review_text)
            )
            conn.commit()
    return {"message": "Відгук успішно додано"     }

@app.get("/api/software/{sw_id}/reviews")
def get_reviews(sw_id: int, user: dict = Depends(get_current_user)):
    with db_manager.get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute("""
                SELECT r.review_id, r.review_text, r.post_date, u.login, u.role
                FROM reviews r
                LEFT JOIN users u ON r.author_id = u.id
                WHERE r.software_id = %s
                ORDER BY r.post_date DESC;
            """, (sw_id,))
```

```
        rows = cur.fetchall()
        return [
            {
                "id": r['review_id'],
                "text": r['review_text'],
                "date": r['post_date'].strftime("%Y-%m-%d %H:%M"),
                "author": r['login'] or "Невідомий",
                "role": r['role'] or ""
            } for r in rows
        ]
```

Додаток Б

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Аналіз та оцінка варіантів програмного забезпечення компонентів комп'ютерної мережі

ВИКОНАВ ЗДОБУВАЧ:

Чорний Юрій Володимирович

Група КСМ-22, Спеціальність 123

НАУКОВИЙ КЕРІВНИК:

Ізмайлова О. В.

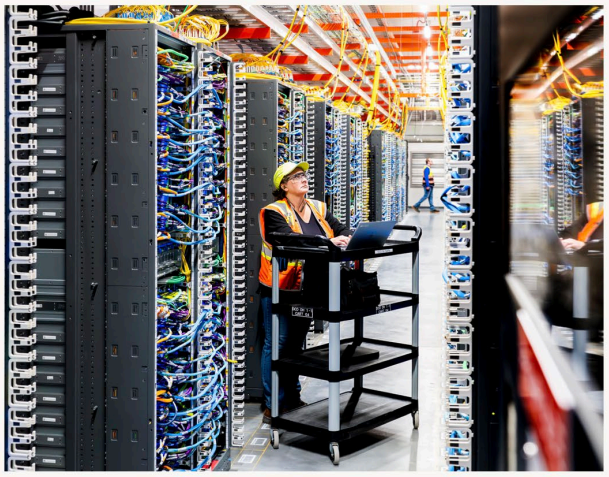
к.т.н., доцент

Слайд 1

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

Сучасна масштабна мережа (Intranet) характеризується високою **гетерогенністю** — використанням обладнання від 6–9 різних вендорів.

Це унеможливлює ефективне ручне адміністрування та потребує впровадження автоматизованих СППР.



Слайд 2

МЕТА ТА ОСНОВНІ ЗАВДАННЯ

- **Мета:** Автоматизація процесу об'єктивного вибору мережевого ПЗ за допомогою СППР на базі методу аналізу ієрархій.
- Проаналізувати проблеми управління корпоративними мережами.
- Здійснити UML-проєктування архітектури системи.
- Адаптувати математичний апарат MAI для розрахунку рейтингів.
- Виконати програмну реалізацію та апробацію системи.

Слайд 3

ОБ'ЄКТ ТА ПРЕДМЕТ

Об'єкт дослідження

Процес багатокритеріального оцінювання та вибору мережевого ПЗ для управління корпоративною IT-інфраструктурою.

Предмет дослідження

Математичні моделі та інформаційна система підтримки прийняття рішень на базі методу аналізу ієрархій (MAI).

Слайд 4

ПРОБЛЕМА ЛЮДСЬКОГО ФАКТОРА

74%

Усіх мережевих збоїв

спричинені помилками персоналу при ручному налаштуванні CLI. Це обґрунтовує потребу в автоматизації за еталонною моделлю **FCAPS**.

Слайд 5

МЕТОДОЛОГІЯ: МЕТОД САТІ

Ієрархія

Декомпозиція проблеми: Мета → Критерії → Альтернативи.

Матриці

Попарне порівняння за 9-бальною шкалою значущості.

Валідація

Розрахунок відношення узгодженості (ВУ ≤ 0.10).

Слайд 6

ПРОЄКТУВАННЯ АРХІТЕКТУРИ

UML-моделювання

- **Use Case:** взаємодія Акторів із системою.
- **Activity:** алгоритм внесення оцінок експертом.
- **Class Diagram:** структура бази даних ПЗ.

Рольова модель (RBAC)

- **Адміністратор:** управління правами доступу.
- **Експерт:** наповнення матриць оцінками.
- **Користувач (ОПР):** аналіз фінальних звітів.

Слайд 7

АЛЬТЕРНАТИВНІ РІШЕННЯ

Zabbix 7.0 LTS

Open Source, розподілена архітектура, висока масштабованість.

Paessler PRTG

Комерційна All-in-One система, орієнтація на Windows.

SolarWinds NPM

Enterprise платформа з глибокою аналітикою NetPath.

Слайд 8

СФОРМОВАНІ КРИТЕРІЇ ОЦІНКИ

Код	Критерій оцінювання	Вага (Pjc)
K1	Сукупна вартість володіння (TCO)	0.111
K2	Масштабованість та швидкодія	0.289
K3	Зручність використання (Usability)	0.067
K4	Надійність та безпека (найвищий пріоритет)	0.533

Слайд 9

ГЛОБАЛЬНИЙ СИНТЕЗ РЕЗУЛЬТАТІВ

Висновок: Zabbix є оптимальним вибором завдяки балансу безпеки та масштабованості.

Слайд 10

ПРОГРАМНА РЕАЛІЗАЦІЯ СППР

- **Backend:** Python та фреймворк FastAPI для високої швидкості обчислень.
- **База даних:** PostgreSQL для надійного збереження матриць оцінок.
- **Frontend:** JavaScript (Vanilla), HTML5 та CSS3 (адаптивний інтерфейс).
- **Безпека:** Авторизація на базі JWT-токенів та розмежування ролей.
- **Математичне ядро:** автоматизований розрахунок пріоритетів та ВУ.

Слайд 11

Запитання?

Дякую за увагу до моєї роботи!

Чорний Юрій Володимирович, 2026

Слайд 12