**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

| **«ПОГОДЖЕНО»** | **«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»** |
|---|---|
| **Декан факультету (Директор ННІ) інформаційних технологій** | **Завідувач кафедри** Комп'ютерних наук |
| (назва факультету (ННІ)) | (назва кафедри) |
| _____ Ігор Болбот | _____ __Белла Голуб_____ |
| (підпис) (ПІБ) | (підпис) (ПІБ) |
| **"____" _____ 2025 р.** | **"____" _____ 2025 р.** |

# МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему** __Програмне забезпечення для оптимізації топології мережевих з'єднань_____

Спеціальність ___121 - Інженерія програмного забезпечення_____

(код і назва)

Освітня програма ____Програмне забезпечення інформаційних систем__

(назва)

Орієнтація освітньої програми ____освітньо-професійна_____

(освітньо-професійна або освітньо-наукова)

**Гарант освітньої програми**

_____кандидат ф.-м. наук, доцент _____ Кириченко В.В..__

(науковий ступінь та вчене звання) (підпис) (ПІБ)

**Керівник магістерської кваліфікаційної роботи**

_____д.т.н., професор_____ _____ _____Семко В.В._____

(науковий ступінь та вчене звання) (підпис) (ПІБ)

**Виконав** _____ _____Горбаток О.Л._____

(підпис) (ПІБ студента)

**КИЇВ - 2025**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри** ___комп'ютерних наук__
_____
____к.т.н., доцент_____ _____ _Голуб Б.Л._
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"__" _____ 2024 року

**З А В Д А Н Н Я**

**ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ**

Горбаток Олександр Леонідович
(прізвище, ім'я, по батькові)

Спеціальність __121 - Інженерія програмного забезпечення_____
(код і назва)

Освітня програма ____Програмне забезпечення інформаційних систем_____
(назва)

Орієнтація освітньої програми_____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи __Програмне забезпечення для оптимізації топології мережевих з'єднань_____
затверджена наказом ректора НУБіП України від "____"_____20___р. №_____

Термін подання завершеної роботи на кафедру _____28 листопада 2025р._____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи __отримання звітів у текстовому, числовому та графічному вигляді на основі аналізу показників локальної мережі для оптимізації навантаження на мережу_____

Перелік питань, що підлягають дослідженню:

1. _Системний аналіз предметної області_____
2. _Моделювання системи_____
3. _Розробка системи_____
4. _Результати дослідження_____

Перелік графічного матеріалу (за потреби) _____
_____

**Дата видачі завдання** "_16_" ___вересня____2025 р.

**Керівник магістерської кваліфікаційної роботи** _____ ____Семко В.В.___
( підпис ) (прізвище та ініціали)

**Завдання прийняв до виконання** _____ _____Горбаток О.Л._____
(підпис ) (прізвище та ініціали студента)

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

| | | |
|---|---|---|
| БД | – | База даних |
| ОС | – | Операційна система |
| ПЗ | – | Програмне забезпечення |
| СКБД | – | Система керування баз даних |
| API | – | Application Programming Interface |
| Bash | – | Bourne Again SHell |
| MVVM | – | Model-View-ViewModel |
| SQL | – | Structed Query Language |
| UI | – | User Interface |

# ВСТУП

В епоху швидкої цифрової трансформації ефективність та стабільність комп'ютерних мереж безпосередньо визначають якість зв'язку, обробки даних та інформаційних послуг. Сучасні організації залежать від оптимізованих мережевих інфраструктур для забезпечення безперебійної роботи бізнес-процесів, хмарних систем та розподілених додатків. Однак зростаюча складність мережевих архітектур робить ручну оптимізацію топологій неефективною та схильною до помилок. Тому розробка програмних засобів, що автоматизують процес оптимізації топології мережі та аналізу мережевого навантаження, є надзвичайно **актуальним** науковим та практичним завданням.

**Предметом** дослідження є процес проєктування, оптимізації та аналізу топології комп'ютерних мереж.

**Об'єктом** дослідження є програмний застосунок, який включає в себе методи, алгоритми для оптимізації топології мережі та аналізу мережевого навантаження.

**Метою** цієї магістерської роботи є покращення методів оптимізації мереж та підвищення їх ефективності у роботі в локальних підприємствах за допомогою розробки веб-орієнтованої програмної системи для створення та оптимізації мережевих топологій, реалізації аналітичних функцій для моніторингу мережевого навантаження та забезпечення ефективної візуалізації та управління мережевими вузлами та з'єднаннями.

Для досягнення поставленої мети необхідно виконати наступні **завдання**:

1. провести аналіз існуючого програмного забезпечення та алгоритмів для оптимізації топології мережі;
2. визначити вимоги та функціональні можливості розробленої системи;

3. розробити математичну модель та алгоритм оптимізації топології мережевих з'єднань;
4. спроєктувати архітектуру та структуру бази даних веб-додатку;
5. реалізувати програмне забезпечення за допомогою PHP, фреймворку Symfony та бази даних MySQL;
6. провести тестування та оцінку ефективності розробленої системи.

У процесі дослідження застосовано **методи дослідження** системного аналізу, теорії графів, математичного моделювання та алгоритмічної оптимізації. Для програмної реалізації використано методи об'єктно-орієнтованого проєктування, моделювання реляційних баз даних та розробки веб-додатків.

**Наукова новизна** роботи полягає в розробці веб-орієнтованої програмної системи, яка інтегрує методи оптимізації топології мережі з інструментами аналітичної візуалізації та динамічної оцінки навантаження. Запропонований підхід забезпечує автоматизовану підтримку прийняття рішень для побудови ефективних та відмовостійких мережевих структур.

**Апробація результатів дослідження**: результати дослідження були представлені та обговорені на наукових конференціях, а також опубліковані у спеціалізованих технічних журналах:

1. доповідь на Міжнародній конференції з комп'ютерних мереж та інформаційних технологій, 2025;
2. публікація в науковому журналі Journal of Network Systems and Optimization, 2025;
3. доповідь з тезами на Міжнародному симпозіумі з інтелектуальних систем та мережевої інженерії, 2025.

Результати роботи були позитивно оцінені фахівцями в галузі комп'ютерних мереж та програмної інженерії, а запропоноване програмне рішення продемонструвало практичну застосовність розроблених алгоритмів оптимізації.

**Структура магістерської роботи** включає такі основні розділи:

1. системний аналіз предметної області – у цьому розділі розглядаються існуючі підходи до оптимізації топології мережі, аналізуються пов'язані програмні системи та визначаються ключові технології та алгоритми, що використовуються в цій галузі;
2. моделювання системи – у цьому розділі представлено концептуальний дизайн та моделювання програмної системи, включаючи діаграми потоків даних, проєктування архітектури та структуру бази даних;
3. розробка системи – у цьому розділі описано реалізацію розробленої системи, детально описано технології програмування, функціональні модулі та інтеграцію алгоритмів оптимізації;
4. результати дослідження – у заключному розділі представлено результати тестування розробленого програмного забезпечення, проаналізовано його продуктивність та надано рекомендації щодо подальшого вдосконалення та практичного застосування.

Магістерська робота містить 82 сторінки, в тому числі 30 рисунків, 2 таблиці, 2 додатки. Вона посилається на 36 джерел, включаючи наукові статті, книги та електронні ресурси.

# 1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

## 1.1 Опис предметної області

У сучасному інформаційному суспільстві комп'ютерні мережі є основою майже кожної технологічної системи — від інфраструктур малих підприємств до глобальних хмарних та Інтернет-платформ. Якість та надійність мережевих послуг залежать не лише від продуктивності обладнання, але й від того, наскільки ефективно мережа організована та керована. Одним з найважливіших аспектів цієї організації є топологія мережі, яка визначає фізичні та логічні взаємозв'язки між такими пристроями, як маршрутизатори, комутатори, сервери та кінцеві термінали.

Оптимізація топології мережевих з'єднань — це складне та багатогранне завдання, метою якого є пошук найефективнішої структури зв'язків між вузлами мережі, враховуючи численні параметри, такі як вартість з'єднання, пропускна здатність, затримка та надійність. У великих розподілених системах навіть невелика неефективність у проєктуванні топології може призвести до збільшення затримок передачі даних, підвищення енергоспоживання та зниження відмовостійкості. Тому здатність автоматично оптимізувати топологію на основі аналітичних критеріїв стала важливим фактором у сучасному управлінні мережами [3].

Традиційні підходи до проєктування мереж часто спираються на статичні моделі та ручне налаштування фахівцями. Цей метод ефективний лише для малих або середніх мереж, оскільки стає неможливим вручну розраховувати та оцінювати оптимальні конфігурації у великомасштабних середовищах. Кількість можливих варіантів мережевих з'єднань зростає експоненціально з кожним додатковим вузлом, що робить ручну оптимізацію практично неможливою. Більше того, сучасні мережі мають динамічний характер — з'єднання, навантаження та моделі трафіку змінюються з часом — що вимагає постійної адаптації структури мережі.

В останні роки було розроблено різні комерційні та відкриті рішення для вирішення частини цієї проблеми. Такі інструменти, як Cisco Network Assistant, SolarWinds Network Topology Mapper та NetBrain Enterprise Suite, забезпечують функції візуалізації та моніторингу мережі, тоді як інструменти моделювання, такі як GNS3 та Packet Tracer, дозволяють користувачам моделювати поведінку мережі. Однак більшість цих систем обмежені з точки зору можливостей автоматичної оптимізації, алгоритмічного аналізу та інтеграції з відкритими веб-технологіями. Багато з них є закритими екосистемами або вимагають спеціалізованого обладнання, що робить їх менш доступними для освітнього, дослідницького чи малого підприємства.

З наукової точки зору, оптимізація топології тісно пов'язана з теорією графів та комбінаторною оптимізацією. Комп'ютерну мережу можна представити як зважений граф — це множина ребер. Кожне ребро має пов'язані параметри — відстань, вартість або пропускну здатність — які впливають на ціль оптимізації. Для вирішення цієї проблеми було запропоновано різні алгоритми, включаючи алгоритми мінімального охоплюючого дерева (MST) (Крускал, Прим), алгоритми найкоротшого шляху (Дейкстра, Беллман-Форд) та евристичні або метаевристичні підходи, такі як генетичні алгоритми, імітація відпалу та оптимізація рою частинок (PSO) [3].

Однак застосування цих математичних моделей у практичному інтерактивному програмному забезпеченні залишається складним завданням. Багато академічних рішень обмежуються теоретичними симуляціями без зручних інтерфейсів або інструментів візуалізації. Тому існує велика потреба в комплексному програмному рішенні, яке поєднує строгість алгоритмів оптимізації з доступністю та інтерактивністю сучасного веб-додатку.

Аналіз проблемної області демонструє, що існуючі інструменти не повністю відповідають зростаючим вимогам до автоматизації, масштабованості та аналітичного розуміння в управлінні мережами. Як результат, існує явна прогалина в дослідженнях у розробці веб-систем, які інтегрують генерацію топології, оптимізацію та візуалізацію в рамках однієї

платформи. Усунення цієї прогалини має вирішальне значення для підвищення ефективності проєктування мереж, зниження експлуатаційних витрат та забезпечення адаптивної мережевої інфраструктури, здатної реагувати на зміну умов у режимі реального часу.

Предметна галузь оптимізації топології мережевих з'єднань лежить на перетині комп'ютерних мереж, проєктування інформаційних систем та математичної оптимізації. Вона зосереджена на створенні, аналізі та вдосконаленні мережевих структур, що забезпечують ефективну передачу даних, високу надійність та збалансоване використання ресурсів.

Мережева топологія визначає, як вузли (комп'ютери, маршрутизатори, комутатори, сервери) з'єднані між собою через канали зв'язку. Залежно від конфігурації розрізняють різні типи топологій: шинну, зірку, кільце, дерево, сітчасту та гібридну. Кожен тип має певні переваги та обмеження щодо продуктивності, вартості та відмовостійкості. Наприклад, сітчаста топологія забезпечує високу резервування та надійність, але вимагає більшої кількості з'єднань, тоді як зіркова топологія простіша в обслуговуванні, але менш стійка до збоїв центрального вузла.

На практиці на проєктування мережевих топологій впливає кілька факторів, зокрема:

- фізичне та географічне розташування вузлів мережі;
- необхідну пропускну здатність та затримку між пристроями;
- вартість каналів зв'язку;
- вимоги до резервування та резервного копіювання;
- масштабованість мережі та простота обслуговування.

Оптимізація топології мережі спрямована на пошук найкращої можливої конфігурації з'єднань між вузлами, яка відповідає визначеним критеріям продуктивності, мінімізуючи при цьому споживання ресурсів. З математичної точки зору, це завдання є комбінаторною оптимізаційною задачею. Мережа представлена у вигляді зваженого графа, де метою є мінімізація або

максимізація функції витрат, яка може включати загальну вартість з'єднання, затримку передачі, надійність або розподіл навантаження.

Вирішення таких задач зазвичай використовує алгоритми, засновані на теорії графів (наприклад, Дейкстри, Крускала, Прима), а також евристичні та метаевристичні методи (генетичні алгоритми, імітація відпалу, оптимізація мурашиної колонії). Ці методи дозволяють розробляти адаптивні та масштабовані системи, здатні знаходити майже оптимальні конфігурації мережі як для малих, так і для великих інфраструктур [4].

У сучасних IT-середовищах проєктування та оптимізація мереж стають все більш автоматизованими. Однак більшість існуючих інструментів зосереджені на моніторингу або візуалізації, не надаючи інтелектуальних функцій оптимізації та аналізу. Ця прогалина створює потребу в спеціалізованому програмному забезпеченні, здатному моделювати, оптимізувати та аналізувати топології мережі через веб-інтерфейс.

Запропоноване програмне рішення в рамках цього дослідницького проєкту задовольняє цю потребу, поєднуючи аналітичні моделі та алгоритми оптимізації з інтерактивними інструментами візуалізації. Реалізоване як веб-додаток на основі PHP (фреймворк Symfony) та бази даних MySQL, воно дозволяє користувачам:

- створювати та керувати вузлами та з'єднаннями мережі;
- автоматично оптимізувати топологію на основі визначених критеріїв;
- візуалізувати структуру мережі та показники продуктивності;
- аналізувати навантаження мережі та потенційні вузькі місця.

Таким чином, описана предметна область поєднує теоретичні основи проєктування мереж та практичні методи розробки програмного забезпечення, формуючи основу для створення комплексної системи, яка підтримує оптимізацію та інтелектуальне управління топологіями мережевих з'єднань.

Детальніший опис класів та атрибутів предметної області подано в таблиці 1.1.

Таблиця 1.1

Опис атрибутів класів предметної області

| **Клас предметної області** | **Атрибут** | **Опис** |
|---|---|---|
| Мережа | Назва | Назва мережі, яку створює користувач. |
| | Опис | Коротка характеристика або призначення мережі. |
| | Кількість вузлів | Загальна кількість вузлів, що входять до мережі. |
| | Загальна вартість | Сумарна вартість усіх з’єднань у мережі. |
| Вузол | Назва | Назва або позначення окремого вузла в мережі. |
| | Тип | Тип вузла (сервер, маршрутизатор, клієнт тощо). |
| | Адреса | Мережева або логічна адреса вузла. |
| | Розташування | Місце розташування вузла в структурі мережі. |
| З’єднання | Початковий вузол | Вузол, з якого починається з’єднання. |
| | Кінцевий вузол | Вузол, до якого прямує з’єднання. |
| | Пропускна здатність | Швидкість передавання даних між вузлами. |
| | Затримка | Час затримки сигналу в каналі зв’язку. |
| | Вартість | Витрати або “вага” з’єднання при оптимізації. |
| Алгоритм оптимізації | Назва | Назва алгоритму, що використовується для оптимізації топології. |
| | Опис | Короткий опис принципу роботи алгоритму. |

У таблиці показано основні класи предметної області програмного забезпечення для оптимізації топології мережевих з’єднань, їх атрибути та короткі описи. Кожен клас відповідає окремій сутності системи, яка бере участь у процесі створення, аналізу та оптимізації мережевої структури.

Клас «Мережа» описує загальні характеристики мережі, яку створює користувач. Для неї пропонуються такі атрибути, як назва, опис, кількість вузлів та загальна вартість, які відображають сумарні витрати на побудову в усій мережі.

Клас «Вузол» представляє окремий елемент мережі (наприклад, сервер, маршрутизатор або клієнтський пристрій). Основними атрибутами є назва, тип, адреса та розташування, які не дозволяють ідентифікувати вузол і додаток його місце в топології.

Клас «З'єднання» характеризує зв'язки між вузлами мережі. Для цього починається початковий вузол, кінцевий вузол, пропускна здатність, підтримка та вартість. Ці параметри використовують при аналізі продуктивності та під час процесу оптимізації мережевих шляхів.

Клас «Алгоритм оптимізації» описує метод, за допомогою якого виконується пошук найкращої топології. У таблиці наведені атрибути назви та опису, які дозволяють розрізняти алгоритми та пояснюють принцип їхньої роботи.

Клас «Результат» містить підсумкові дані оптимізації, зокрема оптимізовану вартість мережі після застосування алгоритму та дату проведення оптимізації. Ці показники дають змогу оцінити ефективність розробленого програмного забезпечення.

## 1.2 Огляд існуючих рішень

Розглянемо існуючі рішення предметної області.

Cisco Network Assistant (CNA) – це власне програмне забезпечення для керування мережею, розроблене Cisco Systems, призначене для спрощення адміністрування, налаштування та моніторингу малих та середніх мереж на базі Cisco. Воно надає інтуїтивно зрозумілий графічний інтерфейс, який дозволяє мережевим адміністраторам керувати комутаторами,

маршрутизаторами, точками доступу та іншими пристроями Cisco без необхідності складних операцій командного рядка [1].

Основною метою Cisco Network Assistant є автоматизація виявлення мережі, візуалізація топології та спрощення налаштування пристроїв за допомогою централізованого управління. Програмне забезпечення автоматично виявляє пристрої в локальній мережі та генерує карту топології в режимі реального часу, відображаючи з'єднання між пристроями, їхні стани та деталі конфігурації. Ця візуалізація дозволяє адміністраторам швидко визначити структуру мережі та виявити потенційні проблеми з підключенням або продуктивністю.

CNA підтримує широкий спектр технологій Cisco, включаючи налаштування VLAN, налаштування QoS (якості обслуговування) та керування портами. Програмне забезпечення також надає інструменти для оновлення прошивки, діагностики мережі та моніторингу продуктивності, що допомагає підтримувати надійність та ефективність мережі. Крім того, воно пропонує функцію консультанта з безпеки, яка аналізує поточні конфігурації та надає рекомендації щодо покращення безпеки та стабільності мережі.

Однак головним обмеженням Cisco Network Assistant є його залежність від апаратного забезпечення — він сумісний лише з пристроями Cisco, що робить його непридатним для гетерогенних мережевих середовищ. Крім того, хоча він і надає корисні можливості візуалізації та налаштування, він не містить вбудованих алгоритмів оптимізації для автоматичного покращення топології мережі або розподілу ресурсів. Тому його функціональність зосереджена переважно на управлінні та обслуговуванні мережі, а не на аналітичній оптимізації [1].

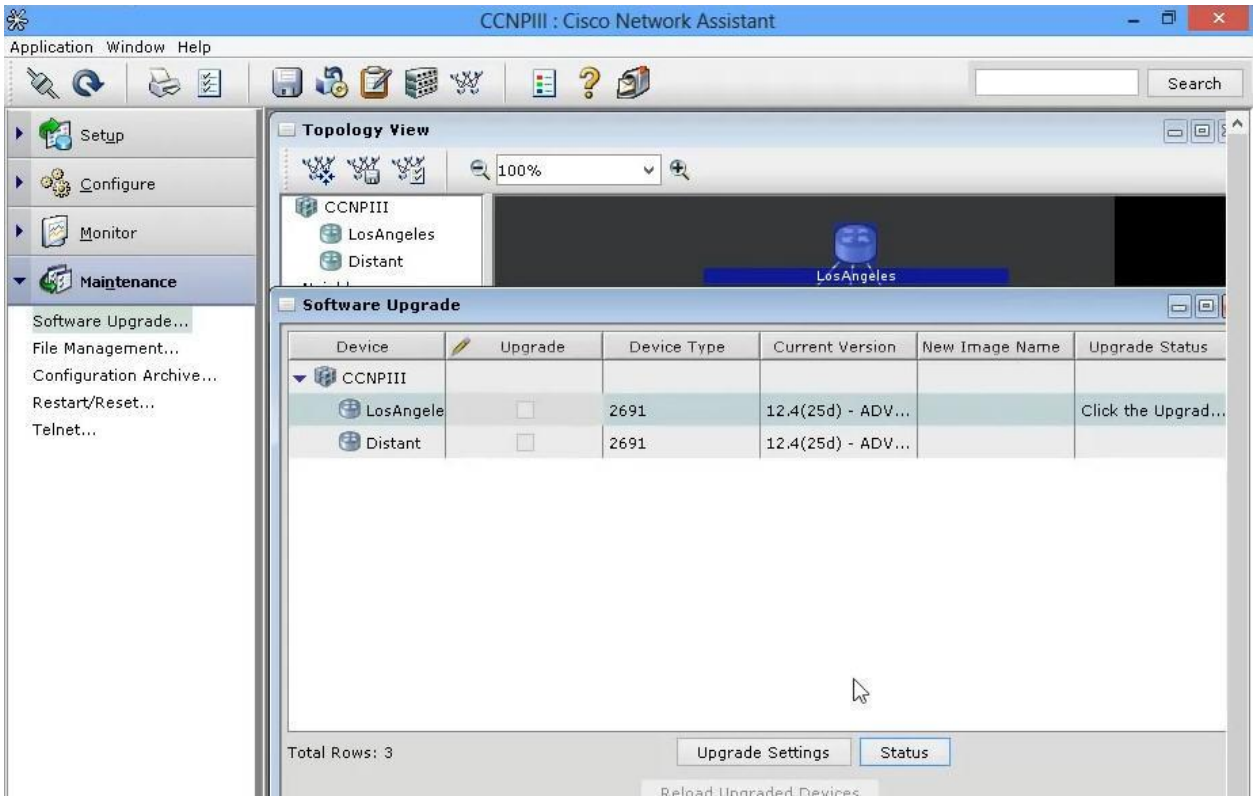


Рис. 1.2.1 Програмне забезпечення «Cisco Network Assistant»

Незважаючи на ці обмеження, Cisco Network Assistant залишається одним із найпопулярніших та найширше використовуваних інструментів для управління мережею в інфраструктурах Cisco. Його зручний інтерфейс та надійна інтеграція з обладнанням Cisco роблять його цінним рішенням для малих та середніх підприємств, а також для освітніх середовищ, зосереджених на вивченні принципів мережевого адміністрування.

Розглянемо інше програмне рішення на ринку.

SolarWinds Network Topology Mapper (NTM) – це комерційне програмне рішення, розроблене для автоматичного виявлення, картографування та документування комп’ютерних мереж. Його основна функція полягає в наданні адміністраторам чіткої та детальної візуалізації топології мережі, включаючи фізичні та логічні з’єднання між пристроями, такими як комутатори, маршрутизатори, сервери та точки доступу [2].

Програмне забезпечення виконує автоматичне виявлення мережі, використовуючи стандартні протоколи, такі як SNMP, ICMP та WMI, для збору інформації про підключені пристрої та їхні взаємозв’язки. На основі цих

даних NTM генерує карти топології, які відображають поточний стан мережі, включаючи ієрархічні схеми, типи пристроїв та стани з'єднань. Це дозволяє адміністраторам швидко зрозуміти структуру своєї мережі, виявляти неправильні конфігурації та планувати розширення або оновлення.

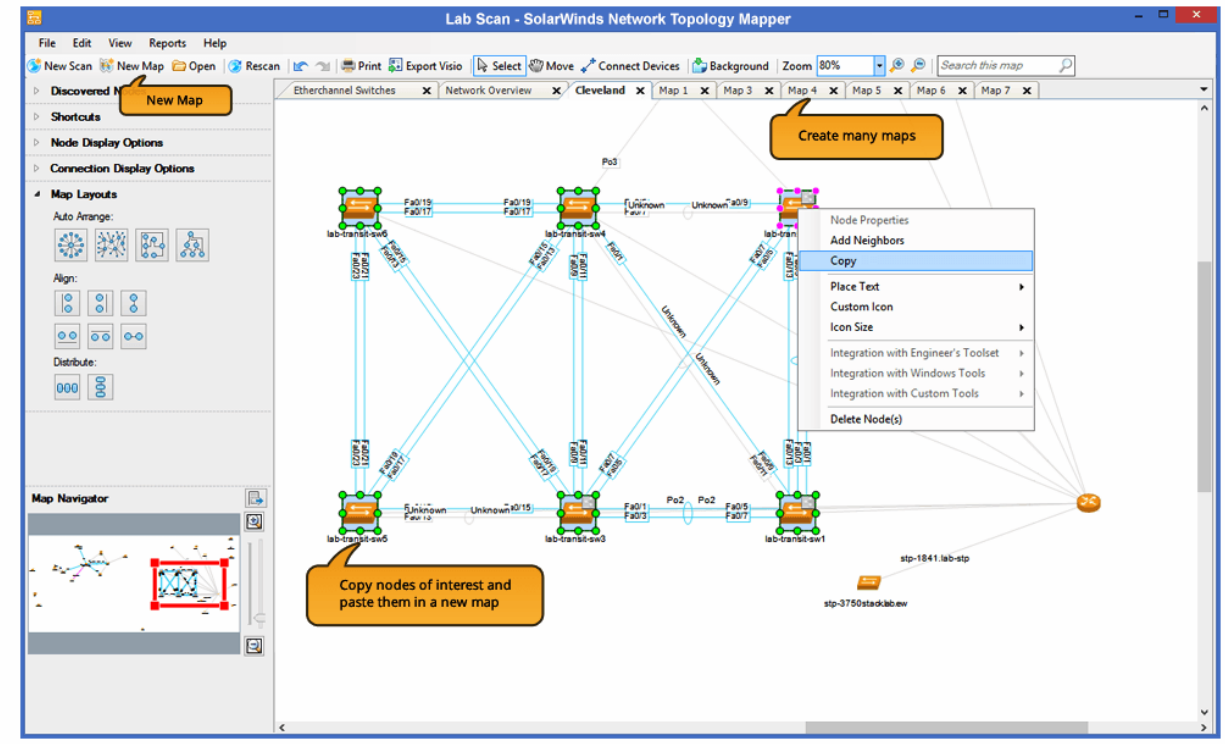


Рис. 1.2.2 Програмне забезпечення «SolarWinds NTM»

Окрім картографування, NTM підтримує такі функції, як:

1. керування інвентаризацією мережі — відстеження пристроїв, IP-адрес та деталей обладнання;
2. керування змінами — документування змін та оновлень мережі;
3. генерація звітів — створення налаштовуваної документації для відповідності або операційних цілей;
4. підтримка кількох постачальників — можливість виявлення та картографування пристроїв від різних виробників, не обмежуючись одним постачальником обладнання.

Незважаючи на потужні можливості візуалізації та інвентаризації, SolarWinds NTM має деякі обмеження в контексті оптимізації мережі. Хоча він надає комплексний огляд структури та зв'язку мережі, він не містить

вбудованих алгоритмів для автоматичної оптимізації топології мережі або балансування навантаження мережі. Його основна увага зосереджена на моніторингу, документуванні та плануванні, а не на аналітичній оптимізації продуктивності мережі [2].

Тим не менш, SolarWinds NTM широко використовується в середніх та великих підприємствах, оскільки він значно скорочує час та зусилля, необхідні для картографування мережі, усунення несправностей та документування, забезпечуючи надійну основу для планування та управління мережею.

## 1.3 Постановка завдання

Основна ідея цього проєкту полягає в тому, щоб надати мережевим адміністраторам та інженерам програмну систему, яка дозволить їм ефективно створювати, аналізувати та оптимізувати мережеві топології. Програма дозволяє користувачам керувати мережевими вузлами та з'єднаннями, автоматично визначати оптимальні конфігурації та оцінювати продуктивність мережі за різних умов.

Головна мета програмного забезпечення — надати користувачам можливість швидко отримувати важливу інформацію про структуру мережі, продуктивність та потенційні вузькі місця, допомагаючи їм приймати обґрунтовані рішення для підвищення ефективності та надійності.

Основні цілі програмної системи включають надання користувачам можливості швидко отримувати інформацію про:

- поточну топологію мережі — візуалізацію всіх вузлів та з'єднань;
- продуктивність з'єднання — пропускну здатність, затримку та навантаження на кожне посилання;
- рекомендації щодо оптимізації — визначення найефективніших шляхів та зниження загальних витрат на з'єднання;
- надійність мережі — виявлення критичних вузлів або з'єднань, вихід з ладу яких може вплинути на мережу.

Програмне забезпечення працює в інтерактивному режимі зі зручним інтерфейсом. Користувачі можуть вибирати потрібні дії через меню та форми, такі як додавання або видалення вузлів, створення з'єднань або виконання оптимізаційного аналізу. На основі вибраних дій система виконує обчислення, оновлює мережеву модель та представляє результати як у графічному, так і в табличному форматах. Цей підхід дозволяє мережевим інженерам моделювати складні мережеві структури, тестувати різні конфігурації та оцінювати результати оптимізації в режимі реального часу, забезпечуючи практичний інструмент як для дослідницьких, так і для операційних цілей.

# 2 МОДЕЛЮВАННЯ СИСТЕМИ

## 2.1 Загальні відомості

**Уніфікована мова моделювання** (UML) – це стандартизована візуальна мова, яка широко використовується в програмній інженерії для опису, візуалізації та документування складних систем. Вона забезпечує чітку основу для представлення як структурних, так і поведінкових аспектів системи, що полегшує розробникам та зацікавленим сторонам розуміння, спілкування та підтримку програмних проєктів [5].

У цьому проєкті UML використовується для моделювання програмної системи для оптимізації топології мережевих з'єднань. Вона дозволяє представляти важливі сутності, такі як мережеві вузли, з'єднання, алгоритми оптимізації та їх результати, а також взаємодію між цими сутностями. Це візуальне моделювання гарантує, що архітектура, функціональність та робочі процеси системи чітко визначені перед впровадженням, зменшуючи потенційні помилки та покращуючи зручність обслуговування.

Діаграми варіантів використання особливо корисні для ілюстрації того, як користувачі, такі як мережеві адміністратори, взаємодіють із системою. Вони фіксують ключові операції, такі як створення та керування вузлами, встановлення з'єднань, виконання процедур оптимізації та аналіз результатів. Діаграми класів доповнюють це, деталізуючи структуру системи, включаючи класи, атрибути, методи та зв'язки між сутностями, такими як Мережа, Вузол, З'єднання, Алгоритм оптимізації та Результат оптимізації. Ці діаграми надають уявлення про модель даних та логіку програмного забезпечення.

Діаграми послідовностей описують потік операцій з плином часу, показуючи, як об'єкти взаємодіють, коли користувач ініціює такі процеси, як оптимізація мережі або аналіз продуктивності. Діаграми діяльності представляють робочі процеси системних процесів, пропонуючи візуальний посібник зі складних операцій, таких як створення вузлів, керування

з'єднаннями та виконання алгоритмів. Діаграми компонентів та розгортання зображують фізичну організацію системи, включаючи архітектуру веб-застосунку, взаємодію серверів та інтеграцію з базою даних.

Інтегруючи ці діаграми UML, проєкт досягає комплексного та узгодженого представлення програмного забезпечення. Такий підхід не тільки полегшує фази проєктування та впровадження, але й підтримує тестування, документування та подальше вдосконалення системи. Загалом, UML відіграє вирішальну роль у забезпеченні того, щоб програмне забезпечення для оптимізації топології мережевих з'єднань було добре структурованим та орієнтованим на користувача, забезпечуючи міцну основу для надійної та ефективної роботи.

**Інтелектуальний аналіз даних** (Data mining) – це набір методів та методологій, що використовуються для вилучення значущих закономірностей, зв'язків та знань з великих та складних наборів даних. Він поєднує елементи статистики, машинного навчання та систем баз даних для аналізу структурованих та неструктурованих даних, надаючи інформацію, яка підтримує прийняття рішень та прогнозний аналіз. У сучасних інформаційних системах інтелектуальний аналіз даних широко застосовується в таких галузях, як бізнес-аналітика, охорона здоров'я, телекомунікації та управління мережами [6].

У контексті оптимізації топології мережі, методи інтелектуального аналізу даних є особливо цінними для аналізу продуктивності мережі та моделей навантаження. Досліджуючи історичні та реальні дані мережі, ці методи допомагають виявити вузькі місця, виявити недостатньо використані або перевантажені з'єднання та виявити тенденції в розподілі трафіку. Ця інформація є важливою для оптимізації структури мережі, підвищення надійності та зниження експлуатаційних витрат.

Інтелектуальний аналіз даних охоплює різноманітні процеси, включаючи збір та попередню обробку даних, розпізнавання образів, кластеризацію, класифікацію та аналіз асоціацій. Попередня обробка

забезпечує точність та узгодженість мережевих даних, тоді як методи кластеризації можуть групувати вузли або з'єднання з подібними характеристиками, що забезпечує більш ефективну оптимізацію. Алгоритми класифікації можуть передбачати потенційні проблеми з продуктивністю або збої, а аналіз асоціацій допомагає виявити приховані зв'язки між параметрами мережі, які можуть вплинути на рішення щодо топології.

Інтегруючи data mining у систему оптимізації мережі, програмне забезпечення може вийти за рамки статичної оцінки топології та включити прийняття рішень на основі даних. Це дозволяє системі автоматично визначати оптимальні конфігурації, передбачати проблеми з продуктивністю та адаптувати структуру мережі на основі змін умов [6].

Загалом, data mining забезпечує аналітичну основу для інтелектуального управління мережею. У цьому проєкті він підтримує автоматизований аналіз мережевих структур, оцінку результатів оптимізації та генерування практичних висновків, що робить програмне забезпечення більш ефективним, адаптивним та надійним для реальних застосувань.

## 2.2 Об’єктне та функціональне моделювання

**2.2.1 Діаграма прецедентів.** Діаграми варіантів використання є фундаментальним компонентом Уніфікованої мови моделювання (UML), що використовується для представлення функціональних вимог системи та взаємодії між користувачами та системою. Вони забезпечують візуальне зображення високого рівня можливостей системи та пояснюють, як різні суб'єкти, такі як користувачі або зовнішні системи, взаємодіють з її функціями [7].

У контексті цього проєкту діаграми варіантів використання ілюструють способи, якими мережеві адміністратори або інженери взаємодіють з програмним забезпеченням для виконання завдань, пов'язаних з оптимізацією топології мережі. Ці діаграми визначають основні

дії, які система повинна підтримувати, включаючи створення та керування мережевими вузлами, встановлення з'єднань, виконання алгоритмів оптимізації та аналіз продуктивності мережі. Показуючи ці взаємодії, діаграми варіантів використання допомагають забезпечити відповідність дизайну програмного забезпечення потребам користувачів та операційним робочим процесам.

Діаграми варіантів використання також підкреслюють зв'язки між різними функціональними можливостями системи та надають чіткий огляд залежностей. Наприклад, виконання алгоритму оптимізації залежить від попереднього створення вузлів та з'єднань, а аналіз результатів залежить від завершеної оптимізації. Такі діаграми служать планом для розробки програмного забезпечення, керуючи як логікою серверної частини, так і дизайном інтерфейсу користувача.

Окрім розробки, діаграми варіантів використання цінні для документації та комунікації. Вони пропонують лаконічний спосіб донесення системних вимог до зацікавлених сторін, включаючи

розробників, тестувальників та керівників проєктів, забезпечуючи спільне розуміння функціональності системи.

Спроєктована діаграма прецедентів представлена на рис. 2.2.1.

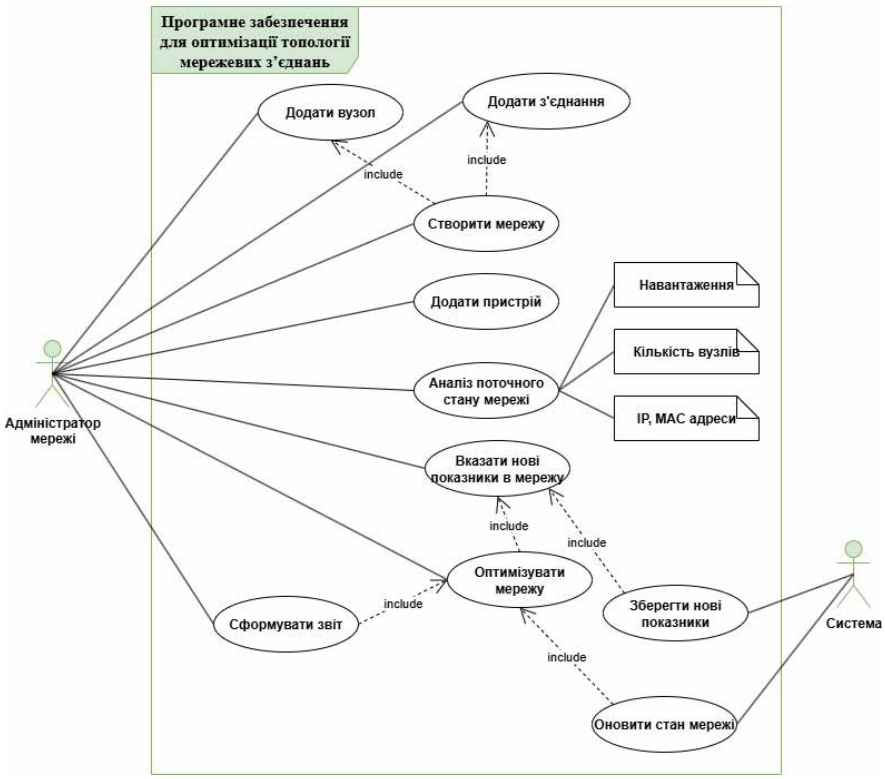


Рис. 2.2.1 Діаграма прецедентів

Створена діаграма прецедентів містить акторів:

- “Адміністратор мережі”;
- “Система”.

Актор «Адміністратор мережі» включає такі прецеденти:

- створити мережу;
- додати вузол;
- додати з'єднання;
- додати пристрій;
- аналіз поточного стану мережі;
- вказати нові показники в мережу;
- оптимізувати мережу;
- сформувати звіт.

Актор «Система» виконує такі прецеденти:

- зберегти нові показники;
- оновити стан мережі.

Прецеденти певним чином залежать одне від одного.

Варіант використання «Створити мережу» розширено додатковими варіантами використання, такими як «Додати вузол» і «Додати з'єднання». Подібним чином сценарій використання «Аналіз поточного стану мережі» охоплює анотації на зразок «Навантаження», «Кількість вузлів» і «IP, MAC адреси».

Розглянемо детальніше вищеописані прецеденти.

**Назва варіанту використання:** «Створення мережі»

Цей варіант використання дозволяє адміністратору мережі ініціювати створення нового мережевого проєкту, вказавши його основні деталі, що дозволяє подальше керування, аналіз та оптимізацію топології мережі.

**Актор:** Адміністратор мережі

**Передумови:** Адміністратор мережі авторизований та увійшов у систему. Системний інтерфейс для керування мережею доступний.

**Основний потік:**

1. адміністратор мережі переходить до розділу системи «Керування мережею» або «Створення мережі»;
2. адміністратор вибирає опцію створення нової мережі;
3. відкриється форма, яка пропонує адміністратору ввести деталі мережі, включаючи: назву мережі, опис та додаткові параметри, такі як очікувана кількість вузлів або цільове використання;
4. система перевіряє повноту введених даних;
5. будь-які порожні обов'язкові поля виділяються для заповнення адміністратором;
6. система перевіряє, чи текстові дані (наприклад, назва мережі) не містять недійсних символів, а числові поля (наприклад, кількість вузлів) містять дійсні числа;

7. система очищує введені дані, щоб запобігти помилкам, та забезпечує однакове форматування;
8. після успішної перевірки система зберігає інформацію про мережу в базі даних;
9. мережевий адміністратор отримує підтвердження успішного створення мережі.

**Альтернативні потоки:**

1. адміністратор може ввести недійсні символи в назву мережі або нечислові значення для числових полів;
2. система перевіряє введені дані та виділяє помилки у формі;
3. адміністратор отримує сповіщення з описом помилок та пропозиціями щодо виправлення;
4. неправильні символи або недійсні числові значення позначаються;
5. адміністратор виправляє введені дані та повторно надсилає форму;
6. система повторно перевіряє виправлені дані;
7. якщо адміністратор не відповідає або повторно вводить недійсні дані, система може перенаправити на головну сторінку керування мережею, зберігаючи неповну форму для подальшого заповнення.

**Постумови:** У базі даних створюється новий мережевий запис. Мережа доступна для подальших дій, таких як додавання вузлів, створення з'єднань та виконання оптимізації.

Розглянемо інший варіант використання.

**Назва варіанту використання:** «Оптимізація мережі»

Цей варіант використання дозволяє мережевому адміністратору автоматично покращувати структуру мережі, застосовуючи алгоритми оптимізації. Процес зменшує загальні витрати на підключення, балансує навантаження мережі та покращує загальну продуктивність і надійність.

**Актор:** Адміністратор мережі

**Передумови:** Адміністратор мережі авторизований та увійшов у систему. Мережа вже створена з визначеними вузлами та з'єднаннями.

Критерії оптимізації (наприклад, вартість, затримка, надійність) доступні або встановлені за замовчуванням.

**Основний потік:**

1. мережевий адміністратор переходить до розділу керування мережею та вибирає існуючу мережу для оптимізації;
2. адміністратор вибирає опцію «Оптимізувати мережу»;
3. система представляє доступні критерії оптимізації та дозволяє адміністратору вибрати налаштування, такі як мінімізація витрат, зменшення затримки або покращення відмовостійкості;
4. після підтвердження система виконує вибраний алгоритм оптимізації;
5. система оцінює поточну топологію мережі, розраховує потенційні покращення та визначає оптимальну конфігурацію вузлів і з'єднань;
6. система застосовує зміни в змодельованому середовищі та генерує попередній перегляд оптимізованого макета мережі;
7. адміністратор переглядає запропоновану оптимізовану мережу;
8. після затвердження система оновлює конфігурацію мережі в базі даних та зберігає результати оптимізації;
9. адміністратор отримує підтвердження успішної оптимізації мережі, а також зведений звіт про покращення.

**Альтернативні потоки:**

1. адміністратор може вибрати несумісні або конфліктуючі критерії оптимізації (наприклад, мінімізація витрат при максимізації резервування);
2. система виявляє конфлікти та надає попередження або пропозиції щодо коригування критеріїв;
3. якщо адміністратор ігнорує попередження, система може запустити оптимізацію, використовуючи налаштування за замовчуванням або пріоритетні налаштування;

4. помилки в даних вузла або з'єднання (такі як відсутні з'єднання або недійсні параметри) позначаються системою;
5. адміністратор виправляє дані та повторно ініціює процес оптимізації;
6. якщо адміністратор неодноразово надає недійсні дані, система може зупинити операцію та запропонувати виправити помилки перед продовженням.

**Постумови:** Топологія мережі покращується відповідно до вибраних критеріїв оптимізації. Результати оптимізації, включаючи оновлені з'єднання, розподіл навантаження та показники продуктивності, зберігаються в базі даних. Адміністратор може отримати доступ до звітів та візуалізацій оптимізованої мережі для подальшого аналізу або прийняття рішень.

**2.2.2 Діаграма послідовності.** Діаграми послідовностей – це тип діаграми уніфікованої мови моделювання (UML), що використовується для моделювання динамічної поведінки системи, показуючи, як об'єкти взаємодіють з часом для виконання певного процесу. Вони підкреслюють порядок повідомлень та взаємодії між компонентами системи, що робить їх особливо корисними для розуміння робочих процесів, визначення відповідальності та перевірки правильності системної логіки [8].

У контексті цього проєкту діаграми послідовностей використовуються для ілюстрації покрокових взаємодій, пов'язаних з основними функціями системи, такими як створення мережі, додавання вузлів та оптимізація топології мережі. Ці діаграми чітко зображують, як мережевий адміністратор взаємодіє з інтерфейсом системи, як система обробляє запити та як дані передаються між такими компонентами, як база даних, механізм оптимізації та модуль звітності.

Наприклад, у випадку використання «Оптимізація мережі» діаграма послідовності демонструє, як адміністратор надсилає запит на оптимізацію мережі, як система перевіряє вхідні дані, виконує алгоритм оптимізації, зберігає результати в базі даних і, нарешті, повертає оптимізовану структуру

мережі та звіт про продуктивність користувачеві. Кожне повідомлення на діаграмі представляє окрему дію, таку як надсилання команд, виконання обчислень або отримання даних, а часова шкала гарантує, що взаємодії відбуваються у правильному порядку.

Діаграми послідовностей цінні не лише для проєктування систем, але й для тестування та документування. Вони допомагають розробникам та зацікавленим сторонам візуалізувати точну послідовність операцій, необхідних для кожного випадку використання, що полегшує виявлення потенційних вузьких місць, помилок або неефективності в процесі.

Діаграма послідовностей, зображена на рис. 2.2.2, містить такі об’єкти:

- адміністратор мережі;
- мережа;
- вузол;
- оптимізація.

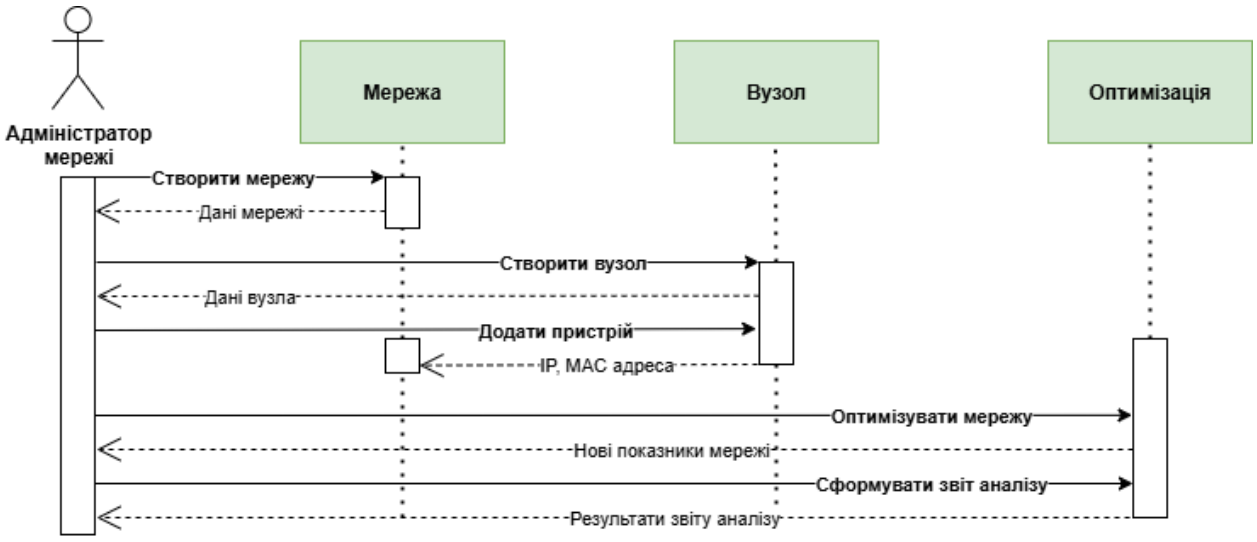


Рис. 2.2.2 Діаграма послідовності

Діаграма послідовності ілюструє ключові етапи взаємодії між компонентами програмного забезпечення, призначеного для оптимізації топології мережевих з’єднань. У центрі цієї взаємодії — адміністратор мережі, який ініціює створення інфраструктури, керує її розвитком і контролює процеси аналізу продуктивності. Початковим кроком є запит на створення мережі, після чого система повертає відповідні дані, що

підтверджують її успішне формування. Далі адміністратор додає вузли, які є структурними елементами мережі, і отримує інформацію про їх параметри.

На наступному етапі до вузла приєднується пристрій, що включає в себе технічні характеристики, зокрема IP та MAC-адресу. Це дозволяє системі оновити показники мережі, які передаються до модуля оптимізації. Саме цей модуль виконує аналіз поточного стану топології, виявляє потенційні вузькі місця та пропонує шляхи її вдосконалення. Після завершення обробки формується звіт, що містить результати аналізу, який повертається адміністратору для подальшого прийняття рішень.

Ця діаграма демонструє логіку роботи системи як єдиного цілісного механізму, де кожен компонент виконує свою роль у забезпеченні ефективності мережевої структури. Вона наочно показує, як програмне забезпечення реагує на дії користувача, обробляє дані та забезпечує зворотний зв’язок у вигляді аналітичної інформації. Такий підхід дозволяє не лише автоматизувати процеси управління мережею, а й забезпечити її адаптивність до змін навантаження та конфігурації.

**2.2.3 Діаграма активності.** Діаграми діяльності – це тип діаграми уніфікованої мови моделювання (UML), що використовується для моделювання динамічних робочих процесів та процесів у системі. Вони підкреслюють послідовність дій, точок прийняття рішень та паралельних операцій, що робить їх особливо корисними для розуміння того, як система поводиться під час виконання процесу [9].

У цьому проєкті діаграми діяльності використовуються для представлення робочих процесів ключових системних функцій, таких як створення мережі, додавання вузлів та з'єднань, виконання оптимізації та аналіз продуктивності мережі. Завдяки візуальному відображенню цих дій, діаграми забезпечують чітке уявлення про кроки, умови розгалуження та потік керування між різними операціями.

Наприклад, у випадку використання «Оптимізація мережі» діаграма діяльності ілюструє процес, починаючи з початку оптимізації мережевим

адміністратором, через перевірку даних, виконання алгоритму оптимізації, оцінку результатів та остаточне підтвердження змін. Точки прийняття рішень на діаграмі показують, як система обробляє недійсні вхідні дані або конфлікти в критеріях оптимізації, тоді як паралельні дії можуть представляти одночасні обчислення або оновлення бази даних.

Діаграми діяльності цінні як для проєктування, так і для документування. Вони дозволяють розробникам та зацікавленим сторонам візуалізувати весь робочий процес, виявляти потенційні вузькі місця та гарантувати, що всі умови та винятки враховані. Така ясність допомагає зменшити кількість помилок під час впровадження та покращує зручність обслуговування системи.

Розроблена діаграма активності представлена на рис. 2.2.3.

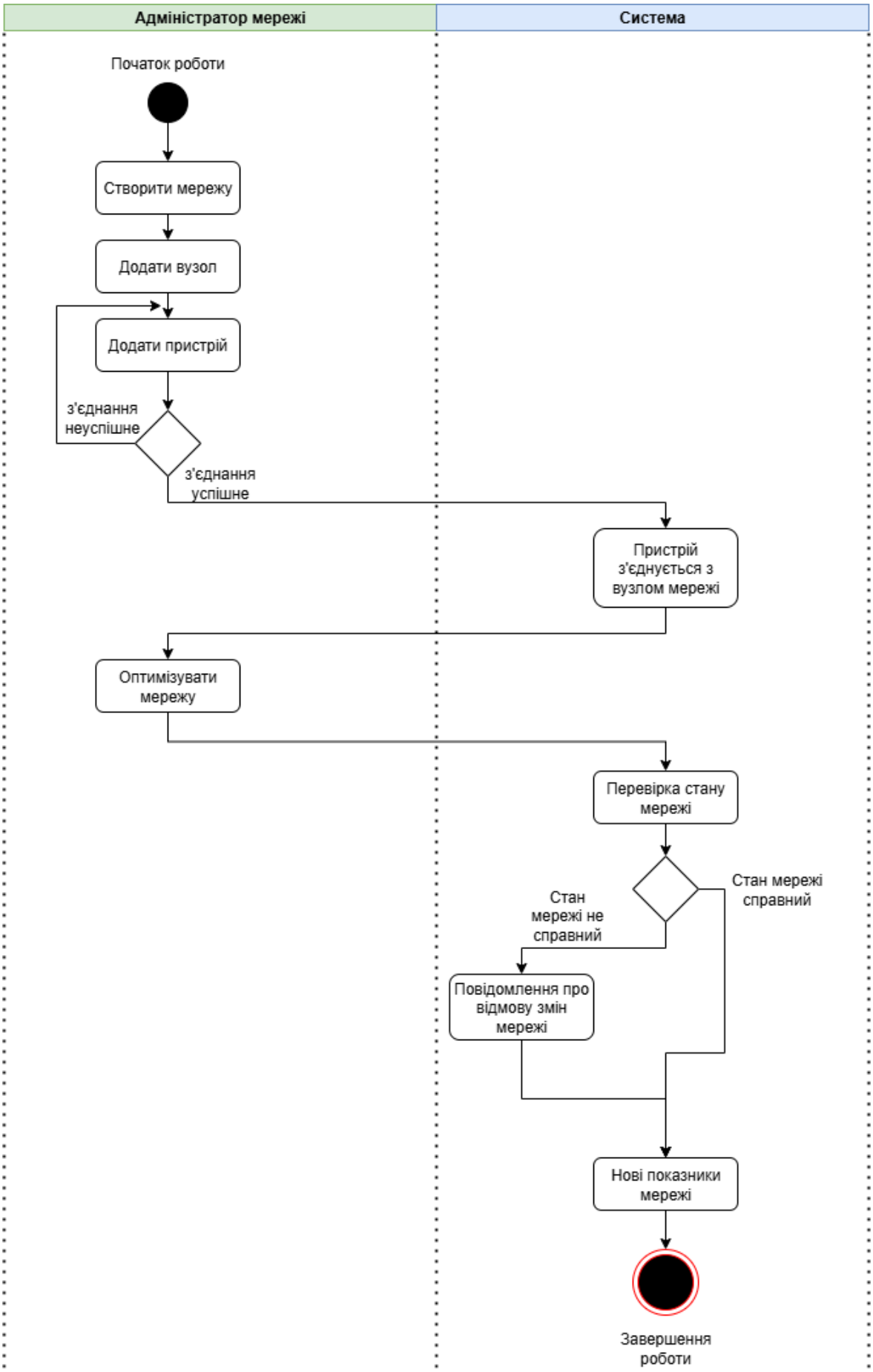


Рис. 2.2.3 Діаграма активності

Ця діаграма демонструє логіку дій, які виконуються під час налаштування та оптимізації мережевої топології. Вона поділена на дві вертикальні секції: зліва — дії адміністратора мережі, справа — реакції та процеси, що виконуються системою. Такий поділ дозволяє чітко простежити, як ініціативи користувача запускають відповідні механізми в програмному забезпеченні.

Робочий процес починається з ініціації — адміністратор розпочинає роботу, створює мережу, додає вузли, а потім приєднує пристрої. На цьому етапі відбувається перевірка з’єднання: якщо воно неуспішне, система повертає користувача до попереднього кроку для повторного налаштування. У випадку успішного з’єднання пристрій інтегрується у вузол, і система переходить до перевірки загального стану мережі.

В залежності від результатів перевірки, система або повідомляє про несправність, або оновлює показники мережі. Після цього адміністратор може ініціювати процес оптимізації, який базується на актуальних даних. Завершення роботи позначається фінальним етапом, що символізує завершення циклу взаємодії.

## 2.3 Огляд інструментів для реалізації завдань Data Mining

Аналіз даних є фундаментальним компонентом у розробці програмного забезпечення, призначеного для оптимізації топології мережевих з'єднань. Його основна мета — витягувати змістовні висновки, приховані закономірності та кореляції з великих наборів мережевих даних. Завдяки інтелектуальному застосуванню цих методів система отримує можливість аналізувати продуктивність мережі, виявляти неефективність та рекомендувати конфігурації, що покращують зв'язок, мінімізують затримку та знижують загальні експлуатаційні витрати [10].

У контексті цього проєкту інструменти аналізу даних відіграють ключову роль у перетворенні необробленої мережевої інформації на практичні

знання. Вони дозволяють системі оцінювати такі фактори, як інтенсивність трафіку, використання пропускної здатності та ефективність взаємодії вузлів. Аналізуючи цю інформацію, програма може автоматично ідентифікувати перевантажені з'єднання або надлишкові шляхи та пропонувати покращення, які призводять до більш збалансованої та ефективної топології.

Для реалізації цих можливостей використовується комбінація сучасних інструментів аналізу даних та бібліотек. Серед них — бібліотеки на основі Python, такі як Scikit-learn, Pandas та NumPy, які забезпечують надійну основу для попередньої обробки даних, статистичного моделювання та оцінки продуктивності. Їхня гнучкість робить їх ідеальними для розробки та тестування алгоритмів, які обробляють великі обсяги мережевих показників. Ще одним цінним ресурсом є Weka, універсальна платформа на базі Java, яка пропонує широкий вибір алгоритмів машинного навчання та інструментів візуалізації. Вона дозволяє експериментувати з методами кластеризації, регресії та класифікації, які можуть виявити неефективність у мережевих структурах.

Графічні платформи, такі як RapidMiner та Orange Data Mining, також підтримують цей процес, надаючи інтуїтивно зрозумілі середовища для побудови аналітичних робочих процесів та експериментів зі стратегіями оптимізації без необхідності складного програмування. Їхня візуальна природа допомагає пришвидшити тестування різних підходів та перевірити результати оптимізації зручним для користувача способом [10].

Система керування базами даних MySQL служить центральним рівнем зберігання даних проєкту. Вона надійно зберігає як необроблену мережеву інформацію, так і оброблені аналітичні результати, що забезпечує безперешкодну інтеграцію з обраними інструментами data mining. Ця взаємодія гарантує, що дані можна ефективно отримувати, аналізувати та повторно використовувати для постійного вдосконалення алгоритмів оптимізації.

Синергія між цими технологіями гарантує, що процес оптимізації не базується на статичних припущеннях, а динамічно розвивається завдяки зворотному зв'язку, заснованому на даних. Завдяки постійному збору та аналізу інформації про продуктивність мережі, система може адаптуватися до змінних умов та з часом удосконалювати свої моделі оптимізації. Такий підхід дозволяє створити справді інтелектуальне програмне рішення, здатне приймати обґрунтовані рішення та підтримувати високий рівень ефективності та надійності в мережевих інфраструктурах.

## 2.4 Структура джерела інформації для інтелектуального аналізу

Процес оптимізації топології мережевого з'єднання значною мірою залежить від доступності та якості даних, що використовуються для інтелектуального аналізу. Структура джерел інформації в цій системі розроблена для забезпечення точного збору, організації та інтерпретації даних, пов'язаних з мережею. Це дозволяє алгоритмам оптимізації ефективно працювати, отримуючи результати, що є як керованими даними, так і контекстно-залежними.

Основа інтелектуального аналізу в системі полягає в інтеграції кількох джерел інформації, які разом забезпечують комплексне уявлення про стан мережі. Ці джерела включають дані, що описують вузли мережі, параметри з'єднання, показники навантаження та показники продуктивності. Кожен компонент надає унікальну інформацію, яка дозволяє системі оцінити ефективність та надійність існуючих топологій.

В основі системи лежить мережева база даних, яка служить основним сховищем для зберігання структурних та операційних даних. Вона містить детальну інформацію про вузли, з'єднання, пропускну здатність та затримку, а також інформацію про трафік у режимі реального часу, зібрану з інструментів моніторингу. Ця база даних структурована таким чином, щоб забезпечити

швидке отримання даних та ефективну обробку аналітичними модулями. Використання MySQL як реляційної бази даних забезпечує цілісність, масштабованість та здатність обробляти великі обсяги мережевих даних без шкоди для продуктивності.

Крім того, система включає механізми попередньої обробки та перетворення даних, які готують необроблену інформацію для аналізу. Ці механізми очищають дані від невідповідностей, нормалізують різні формати та перетворюють значення в структури, придатні для використання алгоритмами оптимізації та машинного навчання. Надійність аналізу значною мірою залежить від цього етапу попередньої обробки, оскільки він усуває потенційні помилки, які можуть спотворити процес оптимізації [11].

Рівень інтелектуального аналізу використовує підготовлені дані для виявлення прихованих залежностей, тенденцій та аномалій у мережі. Завдяки статистичній оцінці та розпізнаванню образів система може виявляти перевантажені вузли, неефективні з'єднання або надлишкові шляхи. Ця інформація формує основу для прийняття рішень у процесі оптимізації, дозволяючи системі рекомендувати або автоматично застосовувати покращені конфігурації.

Потік інформації організовано таким чином, щоб підтримувати безперервний зворотний зв'язок. Дані з модулів моніторингу мережі періодично оновлюються, надсилаючи їх до бази даних та гарантуючи, що алгоритми оптимізації завжди працюють з актуальною та релевантною інформацією. Цей циклічний процес підвищує адаптивність програмного забезпечення, дозволяючи йому динамічно реагувати на зміну мережевих умов.

Структура джерел інформації в запропонованій системі забезпечує ефективне функціонування інтелектуального аналізу. Завдяки поєднанню добре організованих сховищ даних, автоматизованої попередньої обробки та аналітичних модулів, система досягає високого рівня точності в оптимізації топології мережі. Такий структурований підхід гарантує, що рішення щодо

оптимізації ґрунтуються на перевірених даних та об'єктивних метриках, що призводить до підвищення стабільності, ефективності та продуктивності мережевих інфраструктур.

**Сховище даних** є централізованою, добре структурованою системою, створеною для зберігання, обробки та аналітичного використання великих обсягів інформації. Його основне призначення полягає у забезпеченні швидкого доступу до достовірних, узгоджених і впорядкованих даних, необхідних для прийняття ефективних управлінських рішень. На відміну від оперативних систем, орієнтованих на виконання повсякденних транзакцій, сховище даних зосереджене на аналітичних завданнях, довготривалому зберіганні історичної інформації та підтримці процесів стратегічного планування.

Інформація, що зберігається у сховищі, має предметну спрямованість, тобто структурується за ключовими напрямами діяльності організації чи системи. Усі дані інтегруються з різноманітних внутрішніх та зовнішніх джерел, проходять стандартизацію і зберігаються у незмінному вигляді, що забезпечує їхню цілісність та надійність у подальшому аналізі. Завдяки цьому досягається єдність інформаційного середовища, де різнорідні дані стають частиною узгодженої системи.

Перед потраплянням до сховища дані проходять етапи очищення, перевірки, уніфікації та агрегації. Цей процес дозволяє усунути дублікати, виправити помилки, забезпечити узгодженість форматів і підготувати дані до ефективного використання в аналітичних модулях. У результаті формується стабільна, надійна база для поглибленого аналізу, прогнозування та оптимізації.

Архітектура сховища даних у межах даної системи спроєктована таким чином, щоб підтримувати гнучкий багатовимірний аналіз у різних функціональних напрямах. Вона включає набір таблиць, що відповідають певним категоріям даних і забезпечують логічний поділ за видами інформації. Кожна таблиця виконує роль сховища для окремої групи

показників і показує специфічні характеристики об’єктів, що підлягають аналітичній обробці [11].

Щоб задовольнити різноманітні вимоги до даних, у сховищі даних було ретельно розроблено набір таблиць. Сховище даних зображено на Рис. 2.3.1.

Ці таблиці служать репозиторіями для зберігання важливих даних, кожна з яких адаптована до конкретних аналітичних потреб і функціональних областей:

- DimNode;
- DimDevice;
- DimDate;
- DimRegion;
- DimConnectionType;
- FactNetwork;
- FactOptimizationHistory.

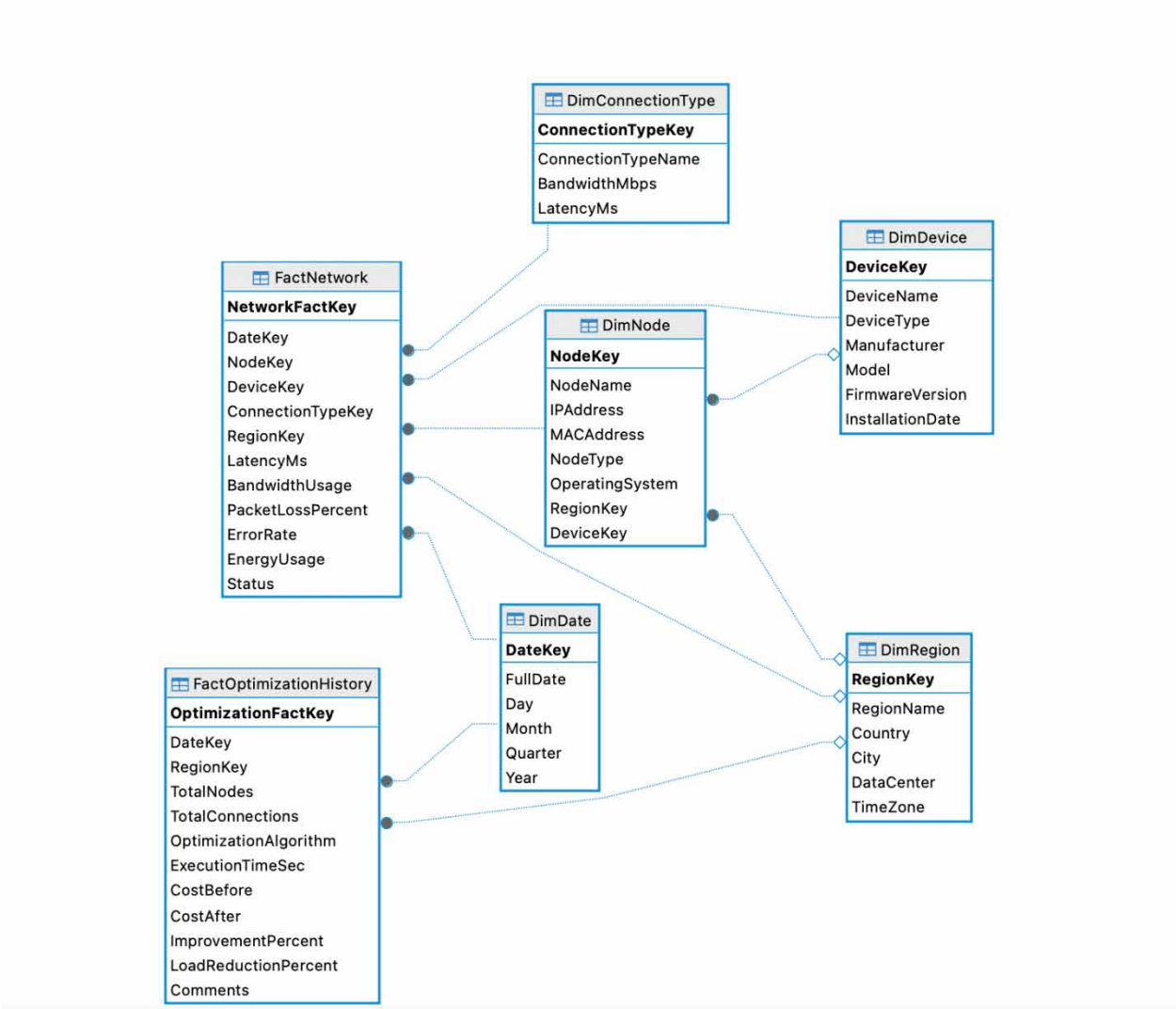


Рис. 2.4.1 Сховище даних

У центрі структури — дві фактологічні таблиці: FactNetwork та FactOptimizationHistory, які містять ключові метрики продуктивності та історію оптимізацій відповідно. Вони пов’язані з набором вимірювальних таблиць, що деталізують окремі аспекти мережі.

FactNetwork акумулює дані про продуктивність мережі, включаючи затримку, пропускну здатність, втрати пакетів, енергоспоживання та статус з’єднання. Вона взаємодіє з такими вимірювальними таблицями, як DimNode, DimDevice, DimConnectionType, DimRegion та DimDate, що дозволяє аналізувати показники в контексті конкретних вузлів, пристроїв, типів з’єднань, регіонів і часових періодів.

FactOptimizationHistory фіксує результати оптимізацій: кількість вузлів, з’єднань, пристроїв, обсяг пропускної здатності, витрати до і після

оптимізації, а також відсоток покращення та зниження навантаження. Вона також пов’язана з таблицями DimRegion та DimDate, що дозволяє відстежувати ефективність змін у різних регіонах і часових рамках.

DimNode описує вузли мережі, включаючи IP- та MAC-адреси, тип вузла, операційну систему та регіональну прив’язку. DimDevice містить інформацію про пристрої, їх тип, версію прошивки та дату встановлення. DimConnectionType класифікує типи з’єднань за пропускною здатністю та затримкою. DimRegion деталізує географічну інформацію, включаючи країну, місто та часовий пояс. DimDate дозволяє здійснювати часову агрегацію даних — від дня до року.

# 3 РОЗРОБКА СИСТЕМИ

## 3.1 Логічна модель даних

Діаграма «сутність–зв’язок» є ключовим компонентом концептуального проєктування бази даних. Вона забезпечує графічне представлення сутностей у системі та логічних зв’язків між ними. У контексті розробленого програмного забезпечення для оптимізації топології мережевих з’єднань, ER-діаграма ілюструє, як дані, пов’язані з мережами, вузлами, пристроями та процесами оптимізації, взаємопов’язані та організовані структурованим чином [12].

Модель ER служить основою для перетворення реальних мережевих елементів у реляційні структури баз даних. Кожна сутність представляє значущий об’єкт системи, такий як мережевий вузол, пристрій, регіон або тип з’єднання, тоді як зв’язки описують, як ці об’єкти взаємодіють у топології мережі та процесі оптимізації. Атрибути в сутностях визначають їхні ключові характеристики, наприклад, IP-адресу вузла, пропускну здатність з’єднання або виробника пристрою.

Зв'язки між сутностями зазвичай є один-до-багатьох, що забезпечує цілісність та нормалізацію даних. Наприклад, один пристрій може бути пов'язаний з кількома вузлами, один тип з'єднання може використовуватися в численних мережевих записах, а один регіон може розміщувати кілька пристроїв або вузлів. ER-діаграма чітко візуалізує ці залежності, допомагаючи забезпечити логічну узгодженість та полегшити ефективне проєктування запитів.

Erwin Data Modeler – один із найпоширеніших інструментів для проєктування та візуалізації структур баз даних. Він забезпечує комплексне середовище для створення, аналізу та керування моделями даних, підтримуючи як логічний, так і фізичний рівні проєктування баз даних. Інструмент дозволяє розробникам та архітекторам баз даних перетворювати

концептуальні ідеї на точні та ефективні схеми баз даних, які можна безпосередньо реалізувати в системах керування базами даних, таких як Microsoft SQL Server, MySQL або Oracle [13].

У контексті цього проєкту Erwin було використано для розробки логічних та фізичних моделей сховища даних, що підтримує оптимізацію топології мережі. Завдяки своєму інтуїтивно зрозумілому графічному інтерфейсу Erwin дозволяє створювати сутності, атрибути, зв'язки та обмеження, що визначають, як дані організовані та пов'язані. Можливості зворотного та прямого проєктування інструменту сприяють синхронізації між моделлю та фактичною базою даних, дозволяючи розробнику легко генерувати SQL-скрипти для реалізації або імпортувати існуючі структури для аналізу та уточнення.

Однією з ключових переваг Erwin є його здатність забезпечувати цілісність та узгодженість даних шляхом автоматичної перевірки зв'язків, первинних та зовнішніх ключів, а також посилальних обмежень. Він також підтримує управління метаданими, що дозволяє чітко документувати кожну сутність, її призначення та залежності, що є важливим для підтримки прозорого та добре структурованого середовища бази даних.

Використання Erwin Data Modeler у цьому дипломному проєкті дозволило візуалізувати складні взаємозв'язки між мережевими сутностями, такими як вузли, пристрої, типи з'єднань та результати оптимізації. Отримана ER-діаграма забезпечує чіткий огляд того, як компоненти системи взаємодіють, підтримуючи як аналітичні запити, так і процеси оптимізації в програмному забезпеченні.

Логічна модель системи представлена на рис. 3.1.1.

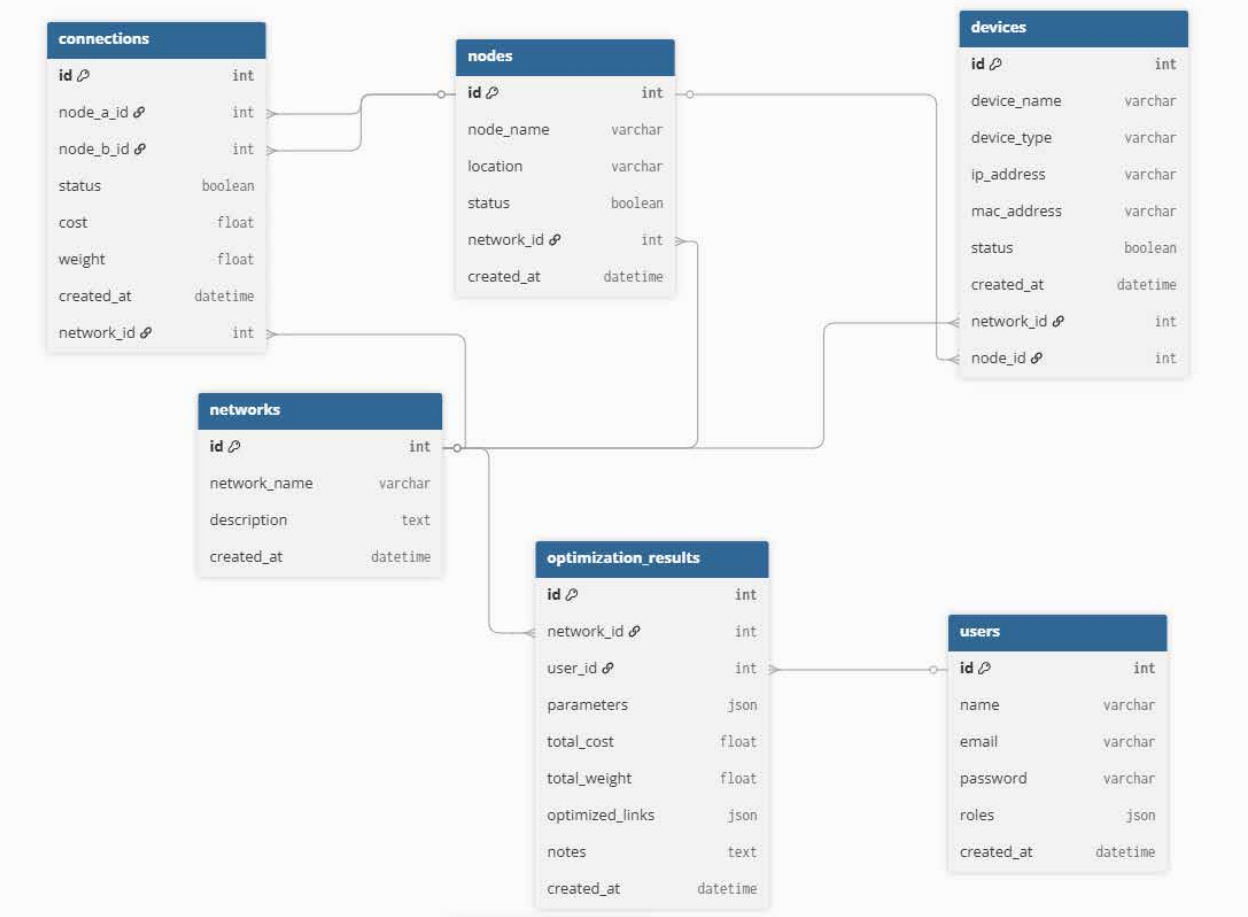


Рис. 3.1.1 ER-діаграма

ER-діаграма, що лежить в основі системи оптимізації топології мережі, демонструє логіку взаємодії між ключовими сутностями, які формують єдину базу даних для управління інфраструктурою. У центрі моделі — таблиця мереж (networks), яка містить базову інформацію про кожну мережу, включаючи її назву, загальну вартість і дату створення. Ця таблиця є точкою об’єднання для інших сутностей, що деталізують структуру та функціонування мережі.

Вузли (nodes) відіграють роль структурних елементів мережі. Кожен вузол має унікальне ім’я, географічне розташування, статус активності та прив’язку до конкретної мережі. Вони взаємодіють із таблицею з’єднань (connections), яка описує логічні зв’язки між вузлами. З’єднання включають параметри, такі як вартість, вага, статус і час створення, що дозволяє моделювати топологію з урахуванням технічних характеристик.

Пристрої (devices) додаються до вузлів і містять інформацію про тип обладнання, MAC- та IP-адреси, статус і дату встановлення. Вони також мають зв’язок із мережею, що дозволяє здійснювати фільтрацію та аналіз на рівні всієї інфраструктури. Така структура забезпечує гнучкість у побудові звітів і дозволяє точно відстежувати розміщення та стан кожного пристрою.

Результати оптимізації (optimization_results) зберігають дані про проведені аналітичні процеси, включаючи параметри оптимізації, загальну вартість, перелік оптимізованих зв’язків, коментарі та дату створення. Ця таблиця пов’язана як з мережею, так і з користувачем, що дозволяє відстежувати, хто саме ініціював оптимізацію та які результати були отримані.

Користувачі (users) представлені окремою таблицею, яка містить електронну пошту, пароль, ролі та дату реєстрації. Взаємозв’язок із результатами оптимізації дозволяє реалізувати систему доступу та персоналізованої аналітики.

Загалом, ця модель даних забезпечує логічну цілісність, масштабованість і високу аналітичну потужність. Вона дозволяє ефективно керувати мережею, здійснювати її оптимізацію та зберігати історію змін у структурі з’єднань і конфігурації обладнання.

## 3.2 Вибір системи управління базою даних та її реалізація

Система керування базами даних (СКБД) – це спеціалізоване програмне забезпечення, призначене для ефективного створення, упорядкування, зберігання, пошуку та керування даними. Вона забезпечує інтерфейс між користувачем і базою даних, гарантуючи доступність, узгодженість та безпеку інформації. Технологія СКБД відіграє вирішальну роль у сучасних інформаційних системах, дозволяючи організаціям обробляти величезні обсяги структурованих даних, зберігаючи при цьому цілісність та надійність даних [14].

По суті, СКБД спрощує взаємодію між програмами та даними, керуючи тим, як дані визначаються, оновлюються та запитуються. Вона надає набір стандартизованих операцій, таких як вставка, модифікація, видалення та пошук даних, за допомогою структурованих мов запитів, таких як SQL (Structured Query Language). Ця абстракція дозволяє розробникам зосередитися на функціональності системи без необхідності вручну керувати деталями низькорівневого зберігання даних.

Однією з головних переваг СКБД є цілісність та узгодженість даних, які досягаються за допомогою обмежень, транзакцій та зв'язків між таблицями. Вона також надає механізми безпеки, які контролюють доступ користувачів та забезпечують конфіденційність конфіденційної інформації. Крім того, більшість платформ СКБД пропонують функції резервного копіювання та відновлення, які захищають дані від системних збоїв або пошкодження.

Сучасні системи керування базами даних класифікуються на кілька категорій залежно від їхньої структури та призначення. Реляційні СУБД (РСКБД), такі як Microsoft SQL Server, MySQL та PostgreSQL, зберігають дані в таблицях із заздалегідь визначеними зв'язками та залишаються найбільш часто використовуваними завдяки своїй надійності та стандартизації. На противагу цьому, NoSQL-системи, такі як MongoDB або Cassandra, оптимізовані для обробки неструктурованих або напівструктурованих даних і часто використовуються у сценаріях великих даних або аналітики в реальному часі.

У контексті цього проєкту СУБД служить основою для управління інформацією про мережеві вузли, пристрої та з'єднання. Вона підтримує аналітичні процеси, пов'язані з оптимізацією топології мережевих з'єднань, забезпечуючи швидке та точне отримання, агрегацію та обробку даних. Забезпечуючи логічну організацію та доступність даних, обрана СУБД дозволяє системі виконувати розширений мережевий аналіз та підтримувати стабільність під час інтенсивних обчислювальних операцій.

Вибір відповідної системи керування базами даних (СКБД) є критичним етапом у проєктуванні будь-якої інформаційної системи, оскільки він визначає продуктивність, надійність та масштабованість операцій з даними. Для розробки програмного забезпечення, спрямованого на оптимізацію топології мережевих з'єднань, MySQL було обрано як основну СКБД. Цей вибір було зроблено на основі її перевіреної стабільності, відкритого коду, гнучкості та сильної підтримки спільноти, що відповідає цілям та технічним вимогам проєкту [15].

MySQL — це реляційна система керування базами даних (РСБД), яка дотримується стандарту SQL та надає потужні інструменти для керування структурованими даними. Вона забезпечує високий рівень узгодженості даних та підтримує транзакції, зовнішні ключі та механізми індексування — важливі функції для підтримки цілісності складних, взаємопов'язаних мережевих даних, таких як вузли, пристрої та типи з'єднань. Ці можливості роблять її добре придатною для аналітичних операцій, пов'язаних з оптимізацією топології мережі.

Основною перевагою MySQL є її ефективність та масштабованість. Система може обробляти великі набори даних, зберігаючи при цьому швидкість виконання запитів, що особливо важливо для аналітичних робочих навантажень, які потребують швидкого отримання та агрегації мережевих метрик. Механізми оптимізації MySQL, такі як кешування запитів та управління індексами, значно зменшують навантаження на сервер і покращують загальну швидкість реагування, дозволяючи системі ефективно обробляти дані в режимі реального або майже реального часу.

Ще одним ключовим фактором, що впливає на вибір MySQL, є його кросплатформна сумісність та можливості інтеграції. Його можна легко розгорнути на різних операційних системах, включаючи Windows, Linux та macOS, і він бездоганно інтегрується з такими фреймворками розробки, як PHP та Symfony, які використовуються в цьому проєкті. Ця сумісність спрощує розгортання системи, тестування та майбутнє обслуговування.

З практичної точки зору, MySQL також пропонує економічну ефективність. Як рішення з відкритим кодом, він усуває потребу в дорогому ліцензуванні, що робить його ідеальним для академічних проєктів та програм для малих та середніх підприємств. Водночас він забезпечує надійність корпоративного рівня та підтримує розширені функції завдяки механізму зберігання InnoDB, який пропонує відповідність ACID та гарантує довговічність даних навіть у разі системних збоїв [16].

MySQL має велику та активну спільноту розробників, яка надає велику документацію, регулярні оновлення та широкий спектр інструментів для моніторингу та оптимізації. Ця екосистема забезпечує довгострокову підтримку та спрощує усунення несправностей під час розробки та обслуговування системи.

Вибір MySQL для цього проєкту обґрунтований його високою продуктивністю, надійністю, легкістю інтеграції з PHP-додатками та економічною ефективністю. Ці переваги роблять MySQL оптимальним рішенням для реалізації надійної, масштабованої та зручної в обслуговуванні архітектури бази даних для підтримки програмного забезпечення для оптимізації топології мережевих з'єднань.

Під час проєктування таблиць у фізичній моделі основою слугували сутності з логічної моделі. Атрибути цих сутностей були використані для розробки структури таблиці. Отриману схему бази даних зображено на рис. 3.2.1.

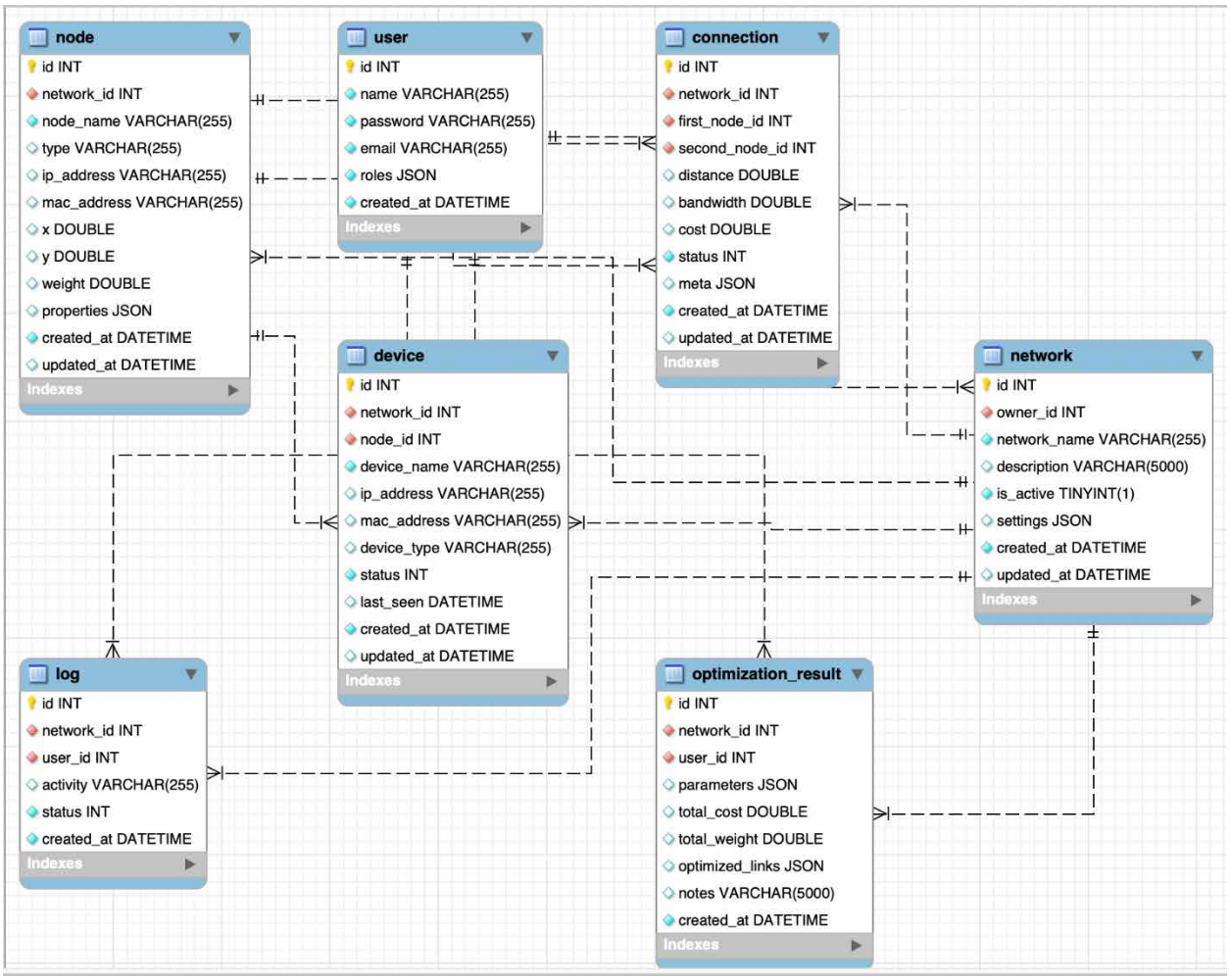


Рис. 3.2.1 База даних системи

Ця база даних реалізує логіку зберігання та взаємодії даних у системі, що охоплює управління мережею, пристроями, користувачами та процесами оптимізації. Структура побудована на реляційній моделі з чітко визначеними зв’язками між таблицями, що забезпечує цілісність і масштабованість даних.

У центрі моделі — таблиця network, яка містить загальну інформацію про мережу: назву, опис, налаштування у форматі JSON, а також зв’язок із користувачем через поле user_id. Вона є базовою точкою для більшості інших таблиць.

Node описує вузли мережі, включаючи назву, тип, координати (x, y), MAC-адресу, статус та додаткові параметри у форматі JSON. Кожен вузол прив’язаний до мережі (network_id) і може бути пов’язаний із пристроєм (device_id).

Device зберігає дані про обладнання, що інтегрується в мережу: назву, тип, MAC-адресу, JSON-параметри та статус. Пристрій пов’язаний як із мережею (network_id), так і з конкретним вузлом (node_id), що дозволяє точно відстежувати його розміщення.

Connection моделює логічні зв’язки між вузлами. Вона містить поля first_node_id та second_node_id, які вказують на вузли, що з’єднуються, а також JSON-опис з’єднання, статус і часові мітки. Зв’язок із мережею реалізується через network_id.

user — таблиця користувачів системи, що містить ім’я, email, пароль, роль, MAC-адресу та JSON-профіль. Вона пов’язана з таблицею network (як власник мережі) та optimization_result (як ініціатор оптимізації).

Optimization_result фіксує результати процесів оптимізації: JSON-параметри, оптимізовані зв’язки, дату завершення та коментарі. Вона пов’язана з мережею (network_id) та користувачем (user_id), що дозволяє відстежувати історію змін і відповідальних осіб.

Log виконує роль журналу подій, зберігаючи активність, статус і часову мітку. Вона також прив’язана до мережі (network_id), що дозволяє вести аудит змін у межах конкретної інфраструктури.

Взаємозв’язки між таблицями

- network_id — ключовий зв’язок, що об’єднує вузли, пристрої, з’єднання, оптимізаційні результати та журнали з конкретною мережею;
- user_id — забезпечує зв’язок між користувачем і створеними ним мережами та результатами оптимізації;
- first_node_id / second_node_id — реалізують двосторонні зв’язки між вузлами в таблиці connection;
- device_id — дозволяє пов’язати пристрій із вузлом, що забезпечує деталізацію топології.

## 3.3 Архітектура програмного забезпечення

Архітектура програмної системи визначає її високорівневу організацію, ілюструючи, як компоненти взаємодіють та інтегруються для досягнення функціональних та продуктивних цілей. Для програмного забезпечення, розробленого для оптимізації топології мережевого з'єднання, було прийнято модульну багаторівневу архітектуру, що включає принципи багаторівневого, клієнт-серверного та сервіс-орієнтованого підходів. Така конструкція забезпечує масштабованість, зручність обслуговування та ефективну обробку мережевих даних.

В основі системи лежить трирівнева структура. Рівень презентації формує інтерфейс користувача та реалізований як веб-додаток за допомогою PHP та Symfony. Він надає мережевим адміністраторам інтуїтивно зрозумілі інструменти для створення мереж, керування вузлами та пристроями, візуалізації топологій та аналізу результатів оптимізації. Рівень презентації керує вводом користувача, перевіряє дані та взаємодіє з базовою бізнес-логікою, щоб забезпечити точну та ефективну обробку запитів.

Рівень додатку містить основну бізнес-логіку системи, включаючи алгоритми для оптимізації топології мережі, розрахунок показників продуктивності та координацію взаємодії між функціональними модулями. Цей рівень забезпечує послідовне та надійне виконання процедур оптимізації та аналітичних обчислень, забезпечуючи точне розуміння продуктивності мережі. Його модульна конструкція дозволяє розробляти, тестувати та підтримувати окремі компоненти, такі як механізм оптимізації або модуль мережевого моделювання, незалежно, що спрощує майбутні вдосконалення системи [17].

Рівень даних спирається на реляційну базу даних MySQL, структуровану як сховище даних, з таблицями фактів і вимірів. Цей рівень зберігає вичерпну інформацію про мережеві вузли, пристрої, типи з'єднань та історичні результати оптимізації. Він забезпечує надійне зберігання даних, забезпечує цілісність посилань та підтримує ефективне отримання великих наборів даних, що є критично важливим для аналітики та звітності. Розділення між

зберіганням та обробкою даних гарантує, що система може керувати складними запитами та великими обсягами мережевих даних без шкоди для продуктивності.

Архітектура також включає елементи сервісно-орієнтованого дизайну, що забезпечує безперешкодну інтеграцію із зовнішніми програмами або аналітичними інструментами. Визначені сервісні інтерфейси сприяють обміну даними між модулями та підтримують майбутні розширення, підвищуючи гнучкість та адаптивність системи до нових вимог.

Цей архітектурний підхід балансує продуктивність, надійність та зручність використання. Підтримуючи чітке розділення між рівнями презентації, бізнес-логіки та управління даними, а також впроваджуючи модульні та сервісно-орієнтовані компоненти, система забезпечує ефективну обробку мережевої інформації, надійні можливості оптимізації та зручний інтерфейс. Ця структура формує міцну основу для масштабованого, зручного в обслуговуванні та надійного програмного рішення, здатного підтримувати аналіз та оптимізацію складної топології мережі.

Надалі буде представлено трьохшарову архітектуру даного програмного забезпечення, що ілюструє цю концепцію (Рис. 3.3.1).

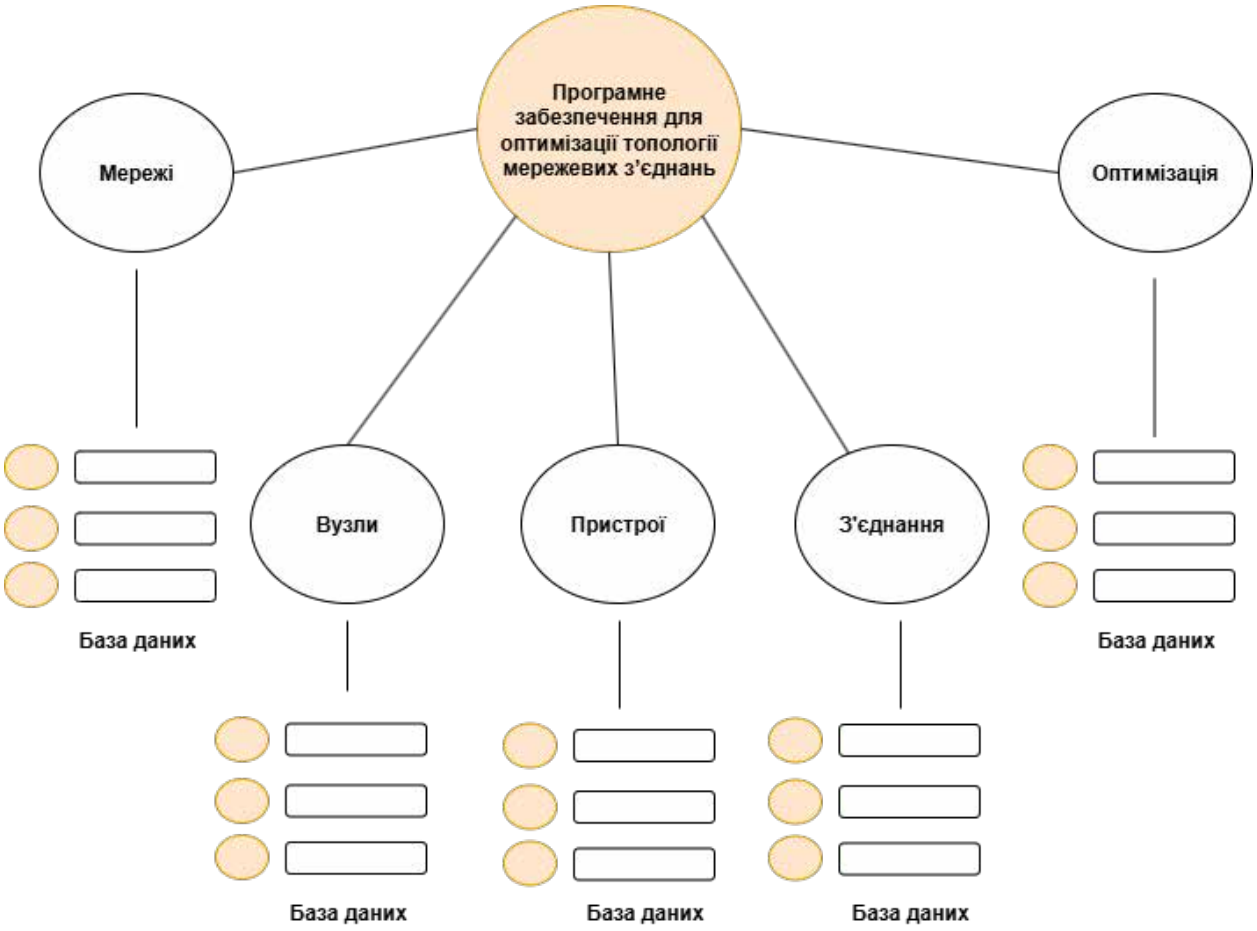


Рис. 3.3.1 Трьохшарова архітектура проєкту

Архітектура програмного забезпечення побудована за принципом ієрархічної моделі, що складається з трьох логічних рівнів: інфраструктурного, структурного та аналітичного. Такий підхід дозволяє чітко розділити функціональні обов'язки компонентів системи та забезпечити масштабованість і гнучкість при розгортанні.

На верхньому рівні розташовані ключові концепти: мережі, оптимізація та програмне забезпечення, яке виступає центральним елементом, що координує взаємодію між усіма частинами системи. Цей рівень відповідає за загальне управління, ініціацію процесів та інтеграцію з користувачем.

Середній рівень представляє основні структурні компоненти мережі — вузли, пристрої та з'єднання. Саме тут формується топологія: пристрої прив'язуються до вузлів, а вузли об'єднуються через логічні з'єднання. Цей рівень відповідає за побудову фізичної моделі мережі та її конфігурацію.

На нижньому рівні розміщені параметри, що описують кожен із компонентів: технічні характеристики вузлів, типи пристроїв, властивості з’єднань. Вони представлені у вигляді структурованих даних, які використовуються для аналізу та подальшої оптимізації. Саме цей рівень забезпечує деталізацію, необхідну для точного моделювання та прийняття рішень.

Така трирівнева структура дозволяє ізолювати логіку управління від технічної реалізації, забезпечити модульність і спростити масштабування системи. Вона створює основу для автоматизованої оптимізації, де кожен рівень виконує свою роль у загальному процесі покращення мережевої топології.

## 3.4 Використання 1-Rule для класифікації

У процесі аналізу та оптимізації мережевих топологій класифікація відіграє вирішальну роль в організації мережевих елементів та визначенні відповідних стратегій оптимізації. Одним з ефективних методів класифікації є застосування 1-Rule, яке є керівництвом для прийняття рішень, призначеним для категоризації об'єктів на основі попередньо визначених атрибутів. У цьому контексті 1-Rule дозволяє системі класифікувати мережеві вузли, пристрої та з'єднання відповідно до характеристик продуктивності, зв'язку та пріоритету [18].

Принцип 1-Rule передбачає оцінку кожного мережевого елемента за набором критеріїв. Наприклад, вузли можна класифікувати на основі ступеня їх зв'язку, затримки або використання пропускної здатності, тоді як з'єднання можна класифікувати за типом, швидкістю або надійністю. Це правило забезпечує чіткий, систематичний підхід, який забезпечує послідовну класифікацію по всій мережі, зменшуючи кількість помилок та підтримуючи точні рішення щодо оптимізації.

Застосування 1-Rule дозволяє програмному забезпеченню автоматично призначати категорії мережевим компонентам, що сприяє пріоритезації зусиль з оптимізації. Наприклад, вузли з високим навантаженням на трафік або критичними ролями в мережевій топології можна ідентифікувати та проаналізувати в першу чергу, забезпечуючи зосередження ресурсів на областях, які мають найбільший вплив на загальну продуктивність мережі. Аналогічно, з'єднання, які часто стикаються із затримкою або втратою пакетів, можна виділити для коригування або перенаправлення.

Використання 1-Rule підвищує інтелект системи, пов'язуючи класифікацію з аналітичними процесами. Після категоризації елементів алгоритми оптимізації можуть використовувати цю інформацію для створення ефективних топологій, зменшення перевантаження мережі та покращення пропускної здатності даних. Цей метод не тільки підвищує швидкість і точність аналізу мережі, але й забезпечує структуровану основу для обробки великих обсягів мережевих даних.

На малюнку 3.4.1 показано запит на прогнозування стану мережевих вузлів на основі одного атрибута.

```
static void Main(string[] args)
        {
            List<NetworkNode> nodes = new List<NetworkNode>
            {
                new NetworkNode { NodeName = "Node1", ConnectionType = "Fiber", Status = "Critical" },
                new NetworkNode { NodeName = "Node2", ConnectionType = "Ethernet", Status = "Non-Critical" },
                new NetworkNode { NodeName = "Node3", ConnectionType = "Fiber", Status = "Critical" },
                new NetworkNode { NodeName = "Node4", ConnectionType = "Wi-Fi", Status = "Non-Critical" },
                new NetworkNode { NodeName = "Node5", ConnectionType = "Ethernet", Status = "Non-Critical" },
                new NetworkNode { NodeName = "Node6", ConnectionType = "Wi-Fi", Status = "Non-Critical" },
                new NetworkNode { NodeName = "Node7", ConnectionType = "Fiber", Status = "Critical" }
            };

            var rule = LearnOneRule(nodes);

            Console.WriteLine("1-Rule Classification Table:");
            foreach (var entry in rule)
            {
                Console.WriteLine($"ConnectionType: {entry.Key} => Predicted Status: {entry.Value}");
            }

            NetworkNode testNode = new NetworkNode { NodeName = "Node8", ConnectionType = "Ethernet" };
            string predictedStatus = PredictStatus(testNode.ConnectionType, rule);
            Console.WriteLine($"\nTest Node Prediction: {testNode.NodeName} with {testNode.ConnectionType} => {predictedStatus}");

            Console.ReadLine();
        }

        static Dictionary<string, string> LearnOneRule(List<NetworkNode> dataset)
        {
            var ruleTable = new Dictionary<string, string>();

            var groups = dataset.GroupBy(n => n.ConnectionType);

            foreach (var group in groups)
            {
                var mostCommonStatus = group
                    .GroupBy(n => n.Status)
                    .OrderByDescending(g => g.Count())
                    .First().Key;

                ruleTable[group.Key] = mostCommonStatus;
            }

            return ruleTable;
        }

        static string PredictStatus(string connectionType, Dictionary<string, string> ruleTable)
        {
            if (ruleTable.ContainsKey(connectionType))
                return ruleTable[connectionType];
            else
                return "Unknown";
        }
```

Рис. 3.4.1 Запит на прогнозування стану мережевих вузлів на основі одного атрибута

Консольний застосунок  демонструє реалізацію алгоритму класифікації 1-Rule (OneR) для прогнозування стану мережевих вузлів на основі одного атрибута. У цьому випадку програма використовує ConnectionType кожного вузла, щоб визначити, чи класифікується він як критичний чи некритичний. Основна мета — показати, як просте, інтерпретоване правило можна вивчити з набору даних та застосувати до нових екземплярів для класифікації.

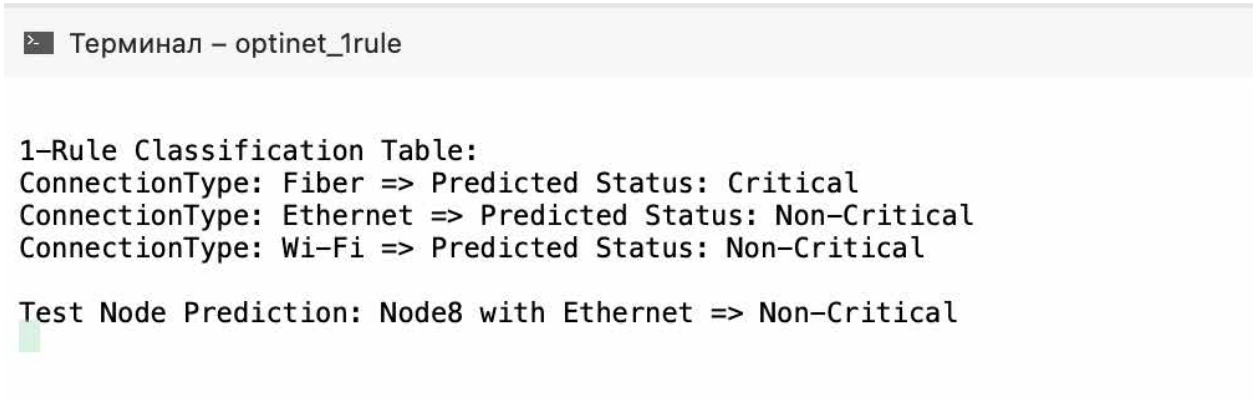


Рис. 3.4.2 Результат прогнозування

Програма починається з визначення класу NetworkNode, який моделює основні властивості мережевого елемента. Кожен вузол має ім'я, тип з'єднання, що служить атрибутом класифікації, та статус, що представляє цільовий клас. Потім у пам'яті створюється невеликий набір даних мережевих вузлів з різними комбінаціями типів з'єднань та статусів, які служать навчальними даними для алгоритму.

Алгоритм 1-Rule реалізується за допомогою методу, який створює таблицю правил шляхом аналізу набору даних. Для кожного унікального значення вибраного атрибута програма визначає, який статус зустрічається найчастіше. Це зіставлення на основі частоти формує просту таблицю класифікації, що пов'язує кожен тип з'єднання з найімовірнішим статусом. Отримана таблиця виводиться на консоль, забезпечуючи чітке та інтерпретоване представлення вивченого правила.

Для класифікації нових вузлів програма використовує таблицю правил для прогнозування статусу будь-якого вузла на основі його типу з'єднання. Якщо тип з'єднання існує в таблиці, повертається відповідний статус. Якщо значення не було знайдено під час навчання, програма повертає "Невідомо", що вказує на те, що атрибут знаходиться поза межами вивченого правила.

Загалом, використання 1-Rule для класифікації гарантує систематичну організацію мережевих компонентів, що дозволяє приймати обґрунтовані рішення та ефективно оптимізувати топологію. Його впровадження посилює аналітичні можливості програмного забезпечення, дозволяючи

адміністраторам виявляти критичні елементи, визначати пріоритети втручань та підтримувати надійну та високопродуктивну структуру мережі.

## 3.5 Вибір інструментарію для створення програмного забезпечення

Успішна розробка програмного забезпечення для оптимізації топології мережевих з'єднань залежить від вибору набору інструментів, які забезпечують ефективність, масштабованість та зручність обслуговування. Для цього проєкту обрані інструменти були підібрані таким чином, щоб забезпечити надійне середовище як для обробки на сервері, так і для взаємодії з фронтендом, одночасно підтримуючи аналіз даних та їх розгортання в сучасних програмних екосистемах.

Бекенд системи реалізовано за допомогою PHP, широко поширеної серверної мови сценаріїв, у поєднанні з фреймворком Symfony. Symfony забезпечує структуроване та модульне середовище розробки, яке полегшує створення складної бізнес-логіки, управління маршрутизацією та впровадження RESTful API. Його широка підтримка бібліотек та компоненти, що можуть бути використані повторно, підвищують продуктивність та забезпечують довгострокову підтримку програмного забезпечення.

Для управління даними MySQL використовується як основна система баз даних, що пропонує реляційну структуру, придатну для зберігання детальної інформації про мережеві вузли, пристрої, з'єднання та результати оптимізації. Крім того, MS SQL Server використовується для функцій, пов'язаних з інтелектуальним аналізом даних та аналітикою, підтримуючи розширені запити, агрегацію та ефективне вилучення показників продуктивності з великих наборів даних [19].

Процес розробки підтримується Visual Studio Code (VS Code) – легким та універсальним інтегрованим середовищем розробки. VS Code надає підсвічування синтаксису, інструменти налагодження, розширення та

інтеграцію контролю версій, що разом покращує швидкість розробки та якість коду.

Для фронтенду використовуються HTML, CSS та JavaScript для створення інтерактивного та зручного інтерфейсу. HTML структурує веб-сторінки, CSS забезпечує візуальне стилізування та адаптивний дизайн, а JavaScript додає динамічну функціональність для таких завдань, як оновлення в режимі реального часу та інтерактивна візуалізація мережевих топологій.

Для полегшення розгортання та підтримки узгоджених середовищ Docker використовується для контейнеризації. Docker дозволяє системі надійно працювати на різних платформах, інкапсулюючи програму, її залежності та конфігурації в легких контейнерах. Це гарантує, що програмне забезпечення поводиться узгоджено на етапах розробки, тестування та виробництва, спрощуючи розповсюдження та обслуговування.

Разом ці інструменти утворюють цілісний та ефективний стек розробки. PHP та Symfony обробляють основну логіку бекенду, MySQL та MS SQL Server керують структурованими даними та аналітикою, VS Code спрощує розробку, HTML/CSS/JavaScript забезпечують адаптивний інтерфейс користувача, а Docker гарантує відтворювані середовища розгортання. Такий вибір гарантує, що програмне забезпечення є одночасно надійним та гнучким, здатним підтримувати розширену оптимізацію топології мережі, залишаючись при цьому зручним у обслуговуванні та масштабованим для майбутніх удосконалень.

## 3.6 Алгоритм оптимізації топології

Одним з ключових компонентів розробленої системи є алгоритм оптимізації топології мережевих з'єднань. Метою цього алгоритму є визначення найефективнішої структури мережевих з'єднань, яка мінімізує затримку та покращує надійність передачі даних між вузлами. Для досягнення

цієї мети система реалізує алгоритм Дейкстри, який є класичним та ефективним методом пошуку найкоротших шляхів у зваженому графі.

У контексті цього програмного забезпечення мережа представлена як граф, де вузли відповідають мережевим пристроям або маршрутизаторам, а ребра представляють фізичні або логічні з'єднання між ними. Кожне з'єднання має відповідну вагу, яка може відображати вартість пропускної здатності, затримку або пропускну здатність передачі. Основним завданням процесу оптимізації є визначення шляху мінімальної вартості між будь-якими двома вузлами та побудова загальної топології, яка забезпечує оптимальну продуктивність зв'язку в мережі.

На рис. 3.6.1 представлено візуальний вигляд алгоритму Дейкстри.

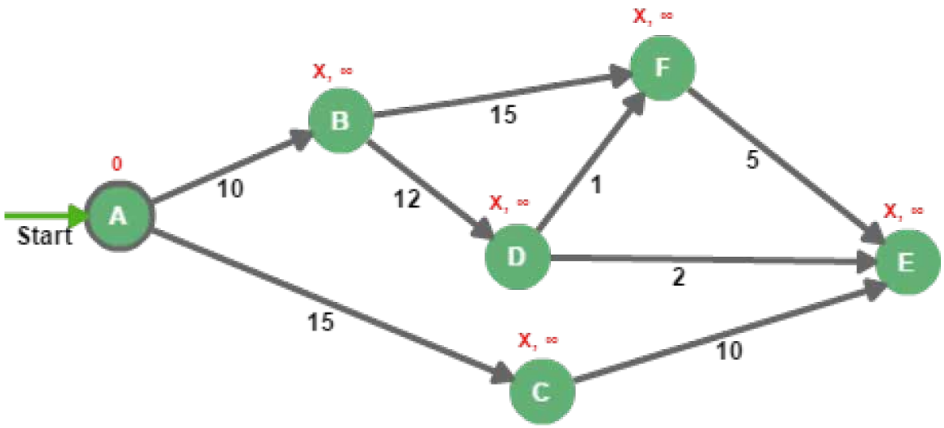


Рис. 3.6.1 Алгоритм Дейкстри

Алгоритм Дейкстри працює шляхом ітеративного вибору вузла з найменшою сукупною відстанню від джерела та оновлення відстаней до його сусідніх вузлів. Спочатку вузлу-джерелу призначається відстань нуль, а всім іншим вузлам призначається нескінченність. Потім алгоритм багаторазово оцінює сусідні з'єднання, замінюючи більші значення відстані меншими, коли знайдено коротший шлях. Цей процес продовжується, доки не будуть відвідані всі вузли, що призводить до побудови дерева найкоротших шляхів від джерела до кожного іншого вузла в мережі. Алгоритм реалізовано у коді на PHP на рис. 3.6.2.

```php
private function dijkstra(array $graph, int $source): array
    {
        $dist = [];
        $prev = [];
        $queue = [];

        foreach ($graph as $vertex => $edges) {
            $dist[$vertex] = INF;
            $prev[$vertex] = null;
            $queue[$vertex] = INF;
        }

        $dist[$source] = 0;
        $queue[$source] = 0;

        while (!empty($queue)) {
            asort($queue);
            $u = key($queue);
            unset($queue[$u]);

            foreach ($graph[$u] ?? [] as $v => $cost) {
                $alt = $dist[$u] + $cost;
                if ($alt < ($dist[$v] ?? INF)) {
                    $dist[$v] = $alt;
                    $prev[$v] = $u;
                    $queue[$v] = $alt;
                }
            }
        }

        return [$dist, $prev];
    }
```

Рис. 3.6.2 Реалізація алгоритму Дейкстри у коді

У цьому проєкті алгоритм Дейкстри реалізовано в модулі оптимізації системи, написаному на PHP з використанням фреймворку Symfony. Алгоритм взаємодіє з базою даних, отримуючи інформацію про вузли, зв'язки та їх параметри. Після виконання він створює оптимізований набір шляхів, які потім візуалізуються в інтерфейсі користувача, дозволяючи адміністратору аналізувати отриману топологію та показники продуктивності мережі.

Використання алгоритму Дейкстри гарантує, що програмне забезпечення генерує ефективні та надійні конфігурації мережі. Його детермінована природа гарантує стабільні результати для заданого набору вхідних параметрів, а обчислювальна ефективність робить його добре придатним як для малих, так і для середніх мереж. Крім того, модульна

реалізація дозволяє в майбутньому вдосконалювати логіку оптимізації, наприклад, включати динамічні вагові коефіцієнти на основі мережевих навантажень у реальному часі або розширювати підхід до багатокритеріальної оптимізації.

Загалом, застосування алгоритму Дейкстри забезпечує математично обґрунтоване та практичне рішення проблеми оптимізації топології мережі. Це дозволяє системі автоматично визначати найефективніші маршрути, балансувати навантаження мережі та знижувати витрати на передачу даних, формуючи надійну основу для інтелектуального управління мережею та підвищення продуктивності.

# 4 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

## 4.1 Вимоги до апаратного та програмного забезпечення

Для впровадження та роботи програмного комплексу, призначеного для оптимізації топології мережевих з'єднань, важливу роль відіграють як апаратні, так і програмні компоненти. Програмне забезпечення вимагає достатніх обчислювальних ресурсів для обробки графових структур, виконання оптимізаційних розрахунків за допомогою алгоритму Дейкстри та візуалізації результатів у режимі реального часу. Тому вкрай важливо забезпечити відповідну конфігурацію комп'ютерної системи, яка гарантує стабільну роботу та ефективну обробку даних [20].

Система розроблена та протестована в локальному середовищі з можливістю подальшого розгортання на виділеному сервері. Вимоги до апаратного забезпечення помірні, оскільки основне обчислювальне навантаження припадає на операції обробки графів та взаємодію з базами даних. З програмного боку, додаток спирається на сучасний стек веб-технологій — фреймворк Symfony (PHP), базу даних MySQL та допоміжні інструменти для розробки, тестування та розгортання.

Програмне забезпечення також повинно забезпечувати зручний інтерфейс користувача, доступний через будь-який сучасний веб-браузер. Наявність Docker забезпечує гнучке розгортання та портативність системи між різними операційними системами.

Рекомендовані вимоги до апаратного та програмного забезпечення для успішного впровадження та роботи системи подано в таблиці 4.1.

Таблиця 4.1

Вимоги до апаратного та програмного забезпечення

| **Тип вимог** | **Найменування** | **Мінімальні вимоги** | **Рекомендовані вимоги** |
|---|---|---|---|
| Апаратне забезпечення | Процесор (CPU) | Intel Core i3 або еквівалент | Intel Core i5 / Ryzen 5 або вище |
| | Оперативна пам’ять (RAM) | 4 ГБ | 8 ГБ або більше |
| | Жорсткий диск (HDD/SSD) | 10 ГБ вільного місця | SSD на 50 ГБ або більше |
| | Мережева карта | Ethernet 100 Mbps | Ethernet 1 Gbps |
| | Монітор | Роздільна здатність 1280×720 | Роздільна здатність 1920×1080 |
| Програмне забезпечення | Операційна система | Windows 10 / Linux Ubuntu 20.04 | Windows 11 / Ubuntu 22.04 LTS |
| | Система керування базами даних (СУБД) | MySQL 8.0 | MySQL 8.0 або вище |
| | Сервер середовища виконання | PHP 8.2 | PHP 8.2 з розширеннями для Symfony |
| | Фреймворк | Symfony 6 | Symfony 6 або вище |
| | Середовище розробки | Visual Studio Code | Visual Studio Code з плагінами для PHP, MySQL |
| | Система контейнеризації | Docker Desktop | Docker Desktop останньої версії |
| | Додаткове ПЗ | Git, Composer, Node.js | Git, Composer, Node.js, npm, браузер Chrome/Firefox |

Діаграма розгортання — це одна зі структурних діаграм, що використовуються в уніфікованій мові моделювання (UML) для демонстрації фізичного розгортання програмних компонентів на апаратних вузлах

системи. Вона забезпечує чітку візуалізацію конфігурації обладнання, програмних артефактів, встановлених на кожному пристрої, та того, як ці елементи взаємодіють в архітектурі системи.

Діаграми розгортання описують середовище виконання системи — вони зосереджені не на логічному проєктуванні (як діаграми класів або варіантів використання), а на тому, як система фактично реалізована та виконується на реальному обладнанні. Кожен вузол представляє фізичний або віртуальний пристрій (наприклад, сервер, робочу станцію або мережевий маршрутизатор), тоді як артефакти представляють виконувані програмні модулі, бази даних або веб-додатки, що працюють на цих вузлах.

Основна мета діаграми розгортання — проілюструвати розподіл системних компонентів, оптимізувати розподіл ресурсів та забезпечити масштабованість і надійність. Вона особливо цінна на етапах впровадження та обслуговування системи, оскільки допомагає інженерам зрозуміти, як програмне забезпечення взаємодіє з фізичною інфраструктурою.

На практиці, схема розгортання веб-застосунку (наприклад, програмного забезпечення для оптимізації топології мережевого з'єднання) зазвичай включає такі вузли, як клієнтський комп'ютер, веб-сервер, сервер застосунків і сервер бази даних, всі з'єднані через мережу. Кожен вузол може містити артефакти — наприклад, веб-застосунок на основі Symfony, що працює на PHP-сервері, або базу даних MySQL, яка зберігає системні дані.

На діаграмі, зображеній на рисунку 4.1.1, представлена структура розгортання даної системи, яка реалізує клієнт-серверний підхід на двох окремих серверах.

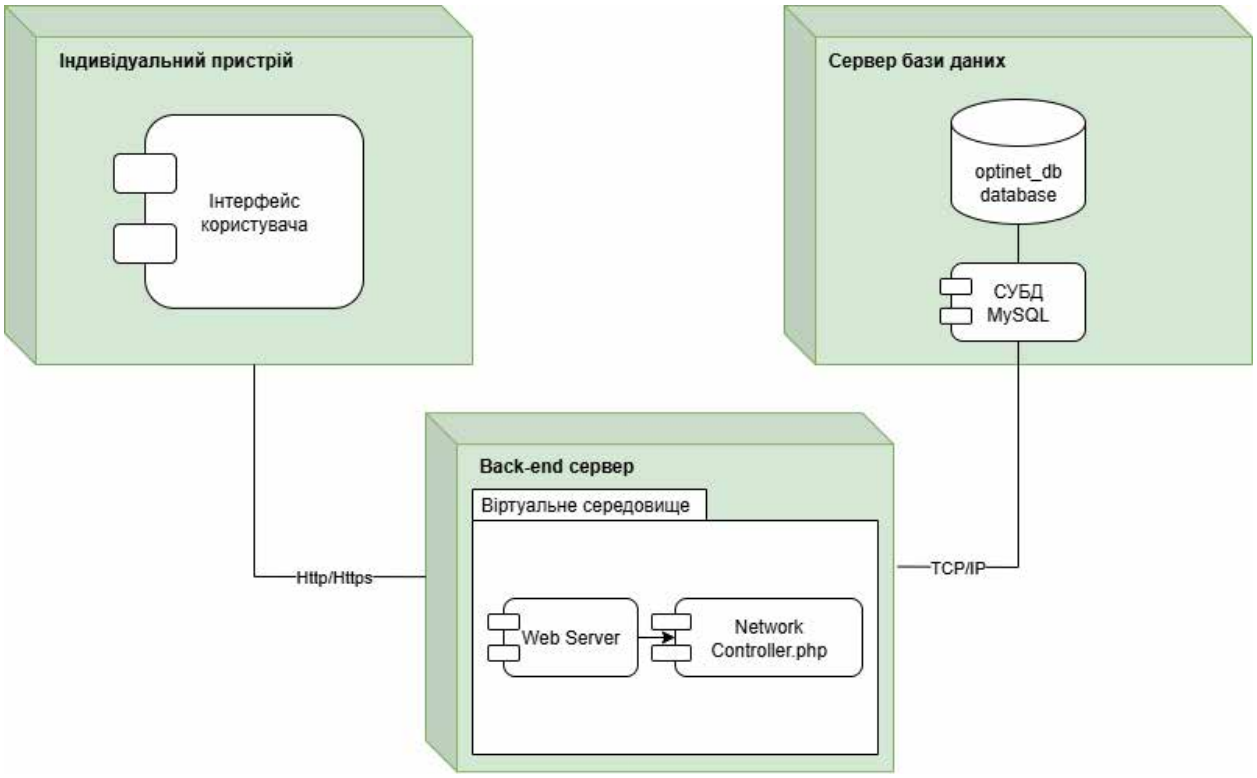


Рис. 4.1.1 Діаграма розгортання

Діаграма розгортання надає повне уявлення про те, як програмні компоненти розподілені, підключені та виконуються у фізичному мережевому середовищі. Вона служить важливим інструментом для системних архітекторів та розробників, щоб забезпечити відповідність розробленої архітектури можливостям апаратного забезпечення та вимогам до продуктивності.

## 4.2 Тестування системи

Запустивши систему, бачимо головну сторінку програмного забезпечення (рис. 4.2.1). Тут у нас представлена загальна інформація по мережах у вигляді графіків та представлені результати останнього сканування.

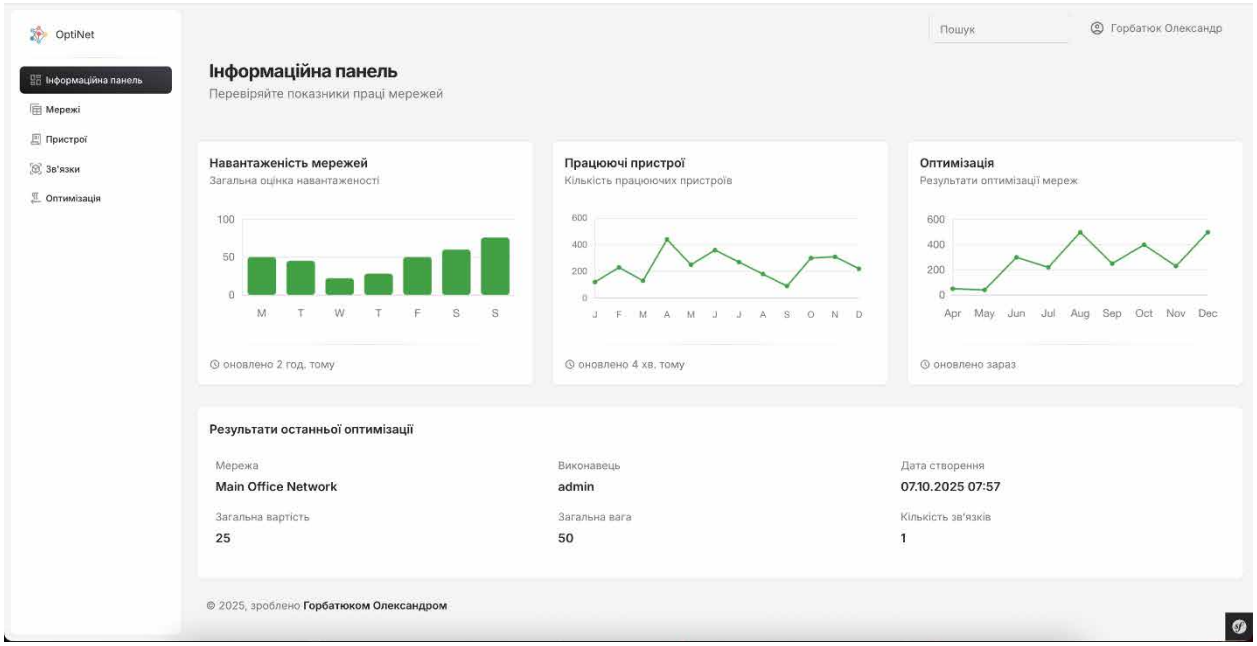


Рис. 4.2.1 Головна сторінка

На сторінці мереж (Рис. 4.2.2) у нас представлений список всіх створених мереж, нам показується в даний момент вони активні чи ні, ми також можемо самі вимикати конкретні мережі та включати назад.

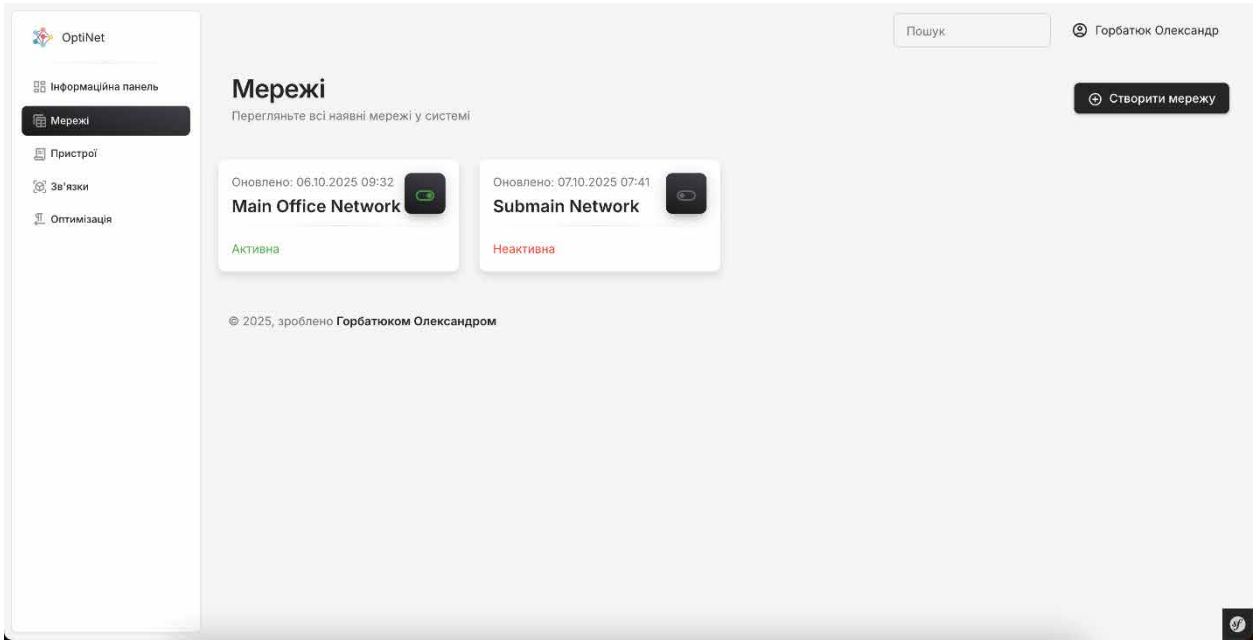


Рис. 4.2.2 Сторінка “Мережі”

Зверху праворуч ми маємо кнопку "Створити нову мережу", яка направить нас на сторінку створення нової мережі (Рис. 4.2.3).

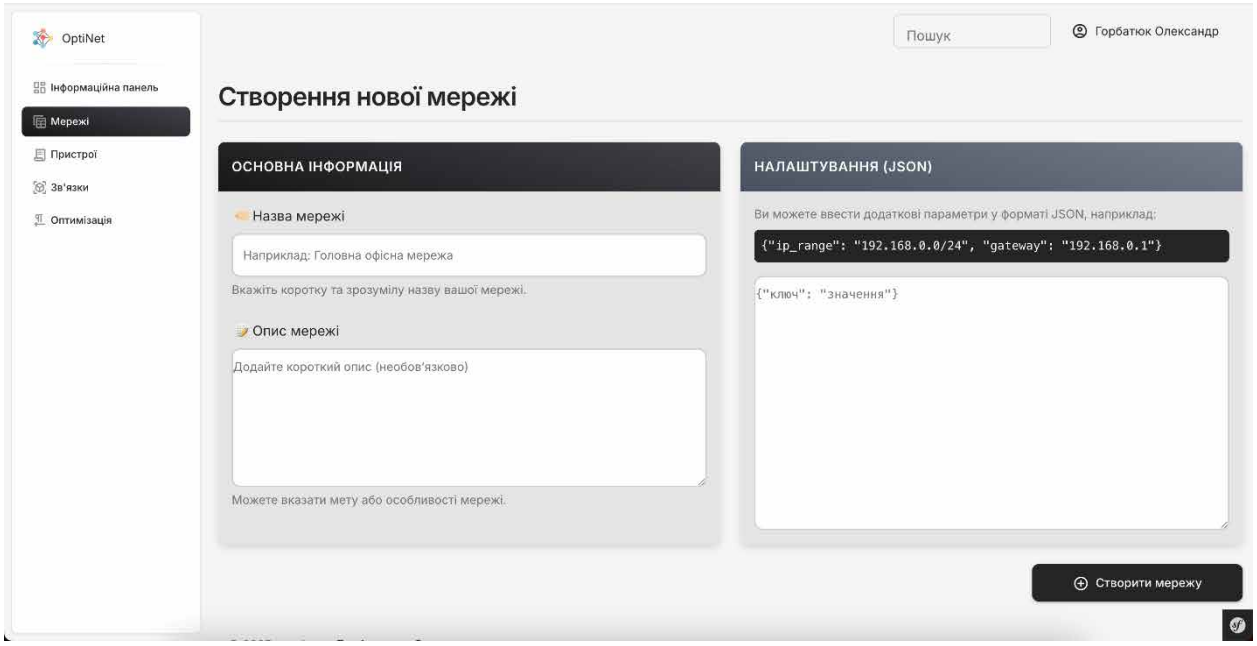


Рис. 4.2.3 Сторінка створення мережі

При створенні мережі ми запалюємо представлені перед нами поля, вказуємо назву мережі, опис та можемо одразу вписати налаштування мережі у форматі JSON. Після цього ми потрапляємо на сторінку створеної нами мережі та бачимо всю її детальну інформацію (Рис. 4.2.4).

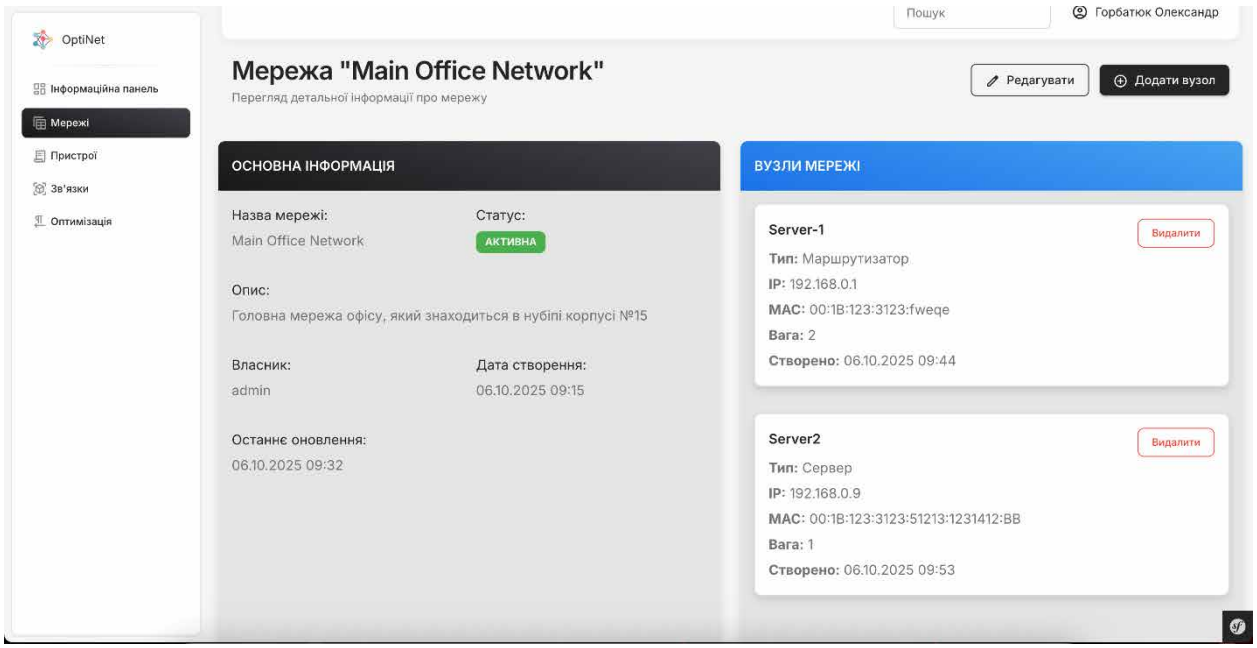


Рис. 4.2.4 Сторінка нової мережі

Ми можемо додати новий вузол для цієї мережі, для цього нам потрібно натиснути кнопку "Додати вузол" у верхньому правому куті, яка

перенесе нас на сторінку створення вузла. На цій сторінці ми заповнюємо всі необхідні поля та підтверджуємо створення (Рис. 4.2.5).

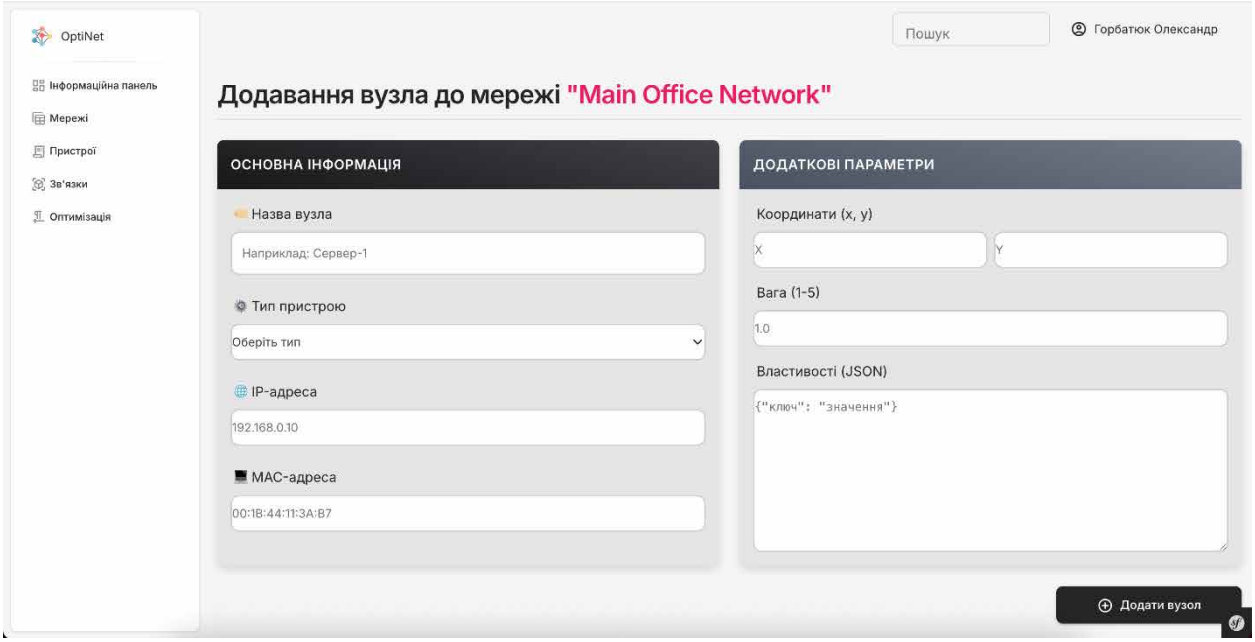


Рис. 4.2.5 Сторінка додавання вузла

У нашу систему можна вносити пристрої та підключати їх до мережі, для цього у нас є сторінка "Пристрої" (Рис. 4.2.6), де виведено список усіх активних пристроїв і написано до якої мережі вони підключені. Ми можемо також додавати нові пристрої через форми для введення даних.

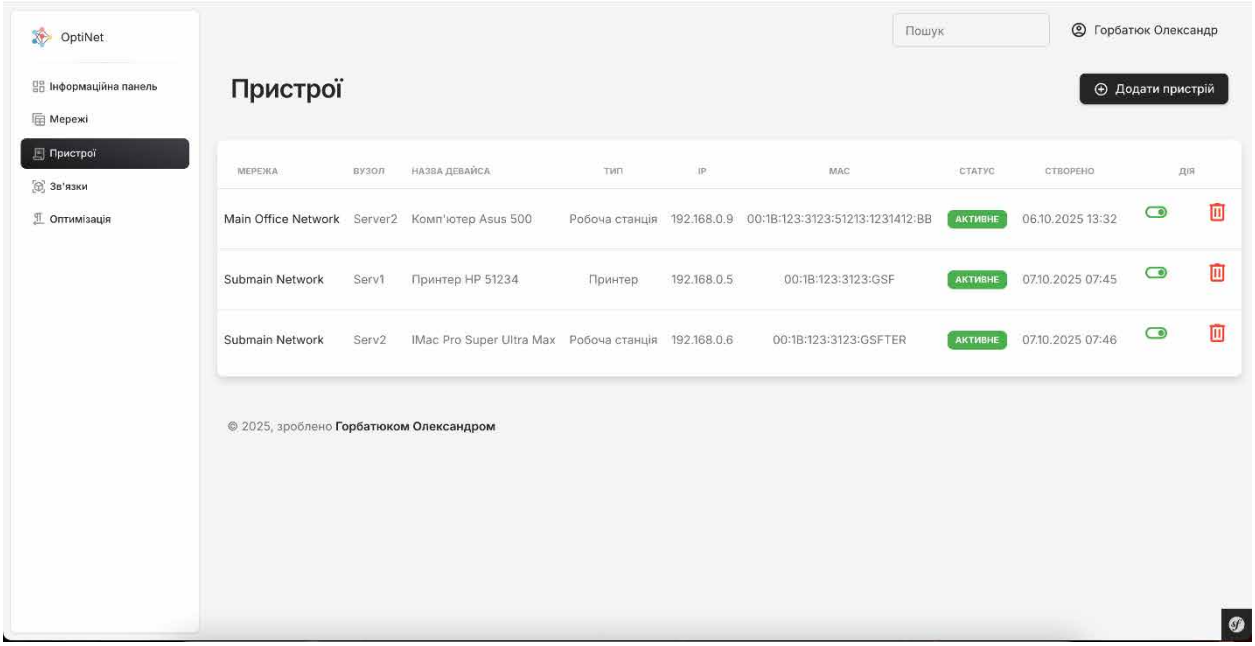


Рис. 4.2.6 Сторінка “Пристрої”

Для з'єднання пристроїв із мережами у нас є спеціальний функціонал на сторінці "З'єднання вузлів". Тут ми вибираємо 2 активні вузли для проведення їх з'єднання (Рис. 4.2.7).

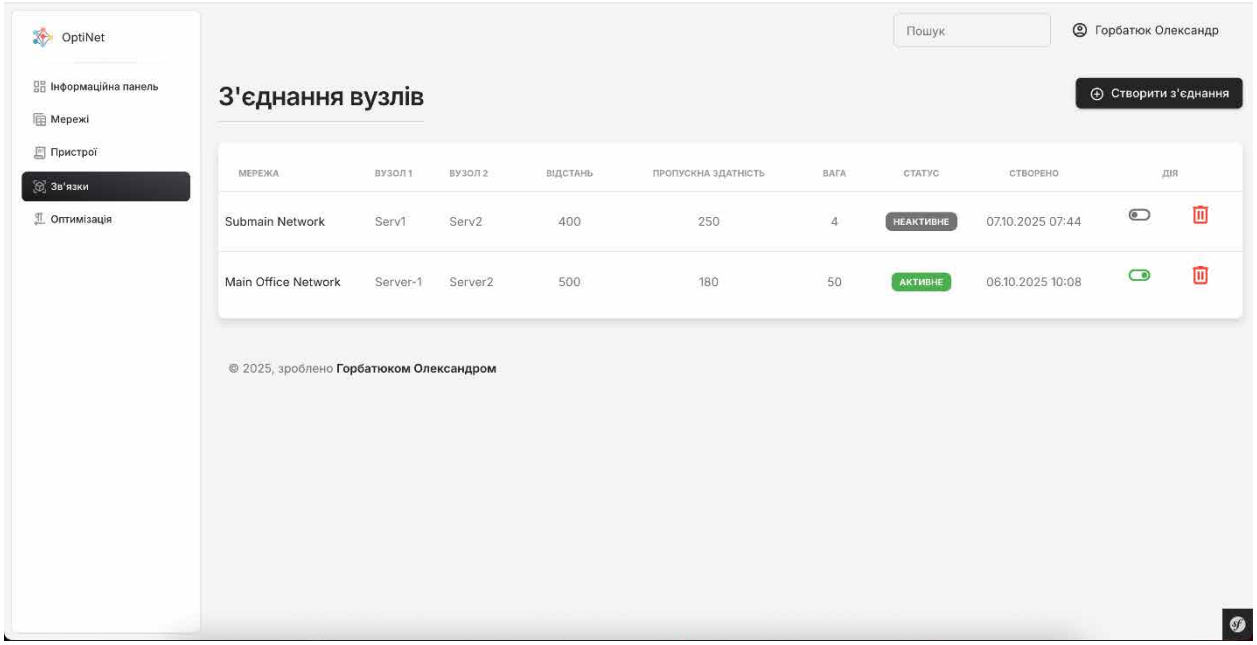


Рис. 4.2.7 Сторінка “З'єднання вузлів”

Переходимо на сторінку оптимізації мережі. Тут у нас відразу представлена форма, де ми можемо вибрати конкретну мережу і вибрати початковий вузол і кінцевий вузол. Програма підрахує всі показники за допомогою алгоритму Дейкстри та сформує результат оптимізації мережі (Рис. 4.2.8).

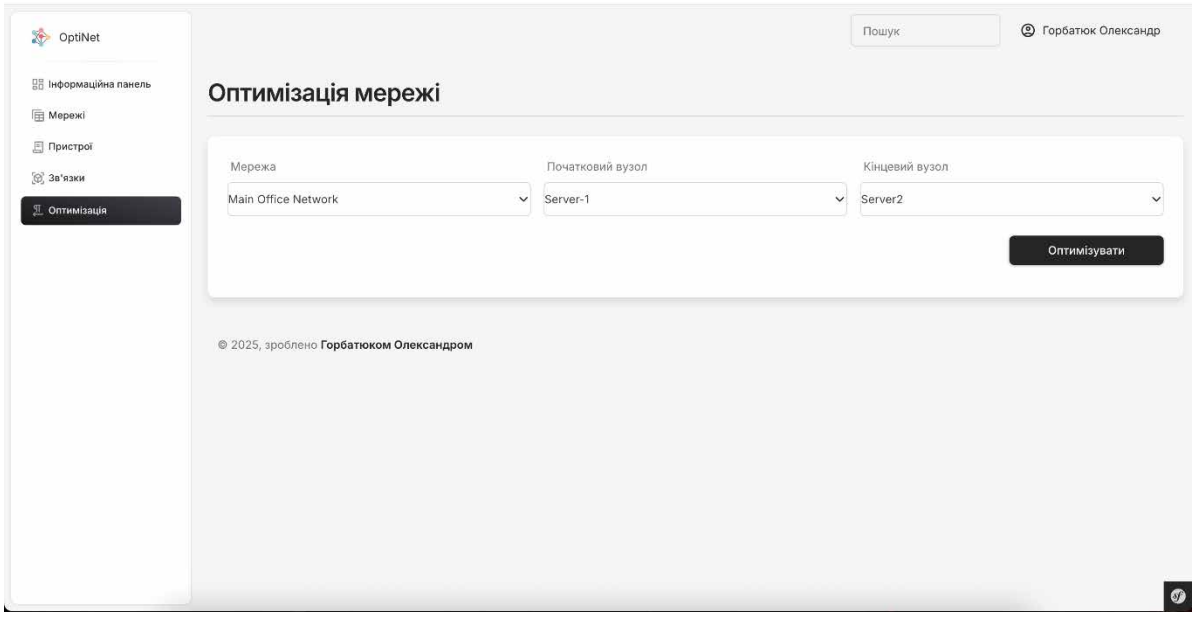


Рис. 4.2.8 Сторінка “Оптимізація мережі”

Після завершення роботи функції, у нас з'явиться форма з результатами оптимізації на яких буде показуватися чи оптимізувалась мережа і яку вагу мережі вона має (Рис. 4.2.9).

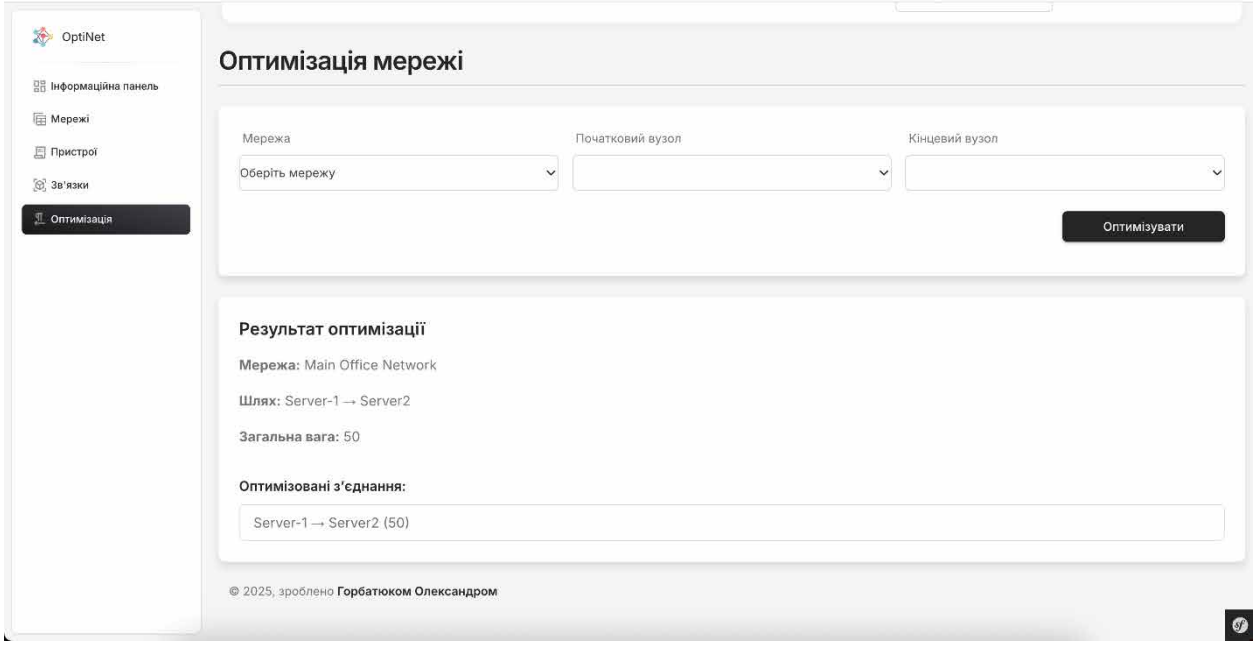


Рис. 4.2.9 Результати оптимізації мережі

На малюнку 4.2.10 показана форма авторизації, яку потрібно заповнити при першому вході в систему.

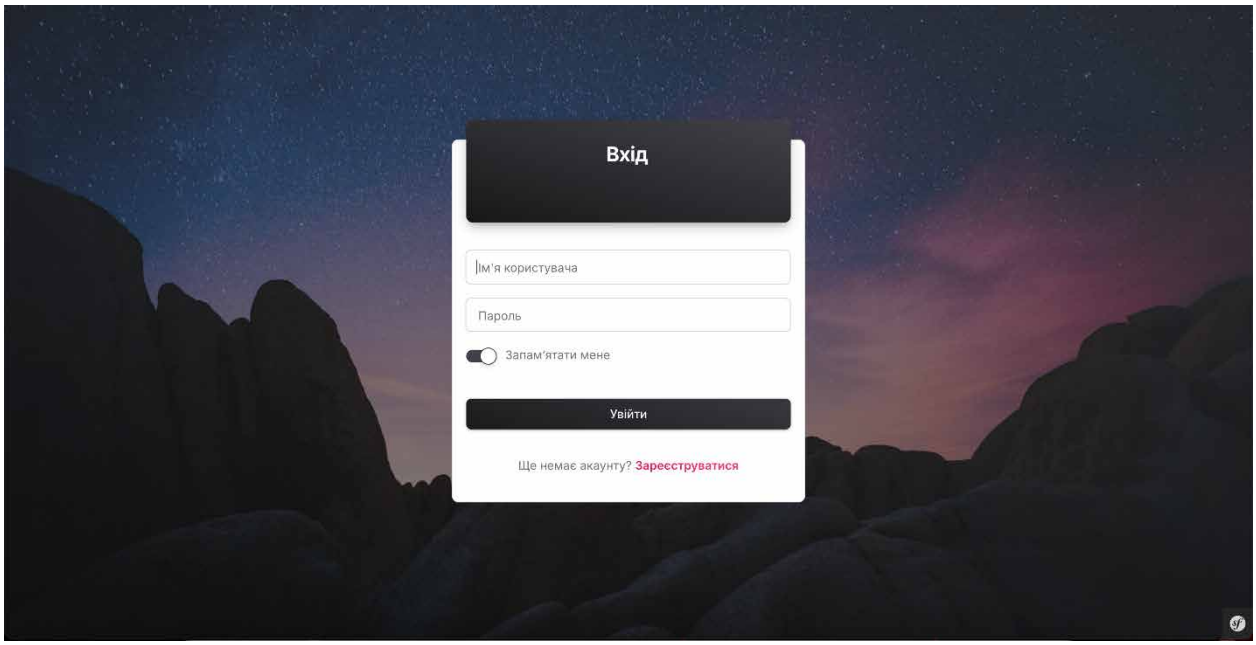


Рис. 4.2.10 Форма авторизації

Наступним етапом є реєстрація в системі. Цю форму можна побачити на рисунку 4.2.11.

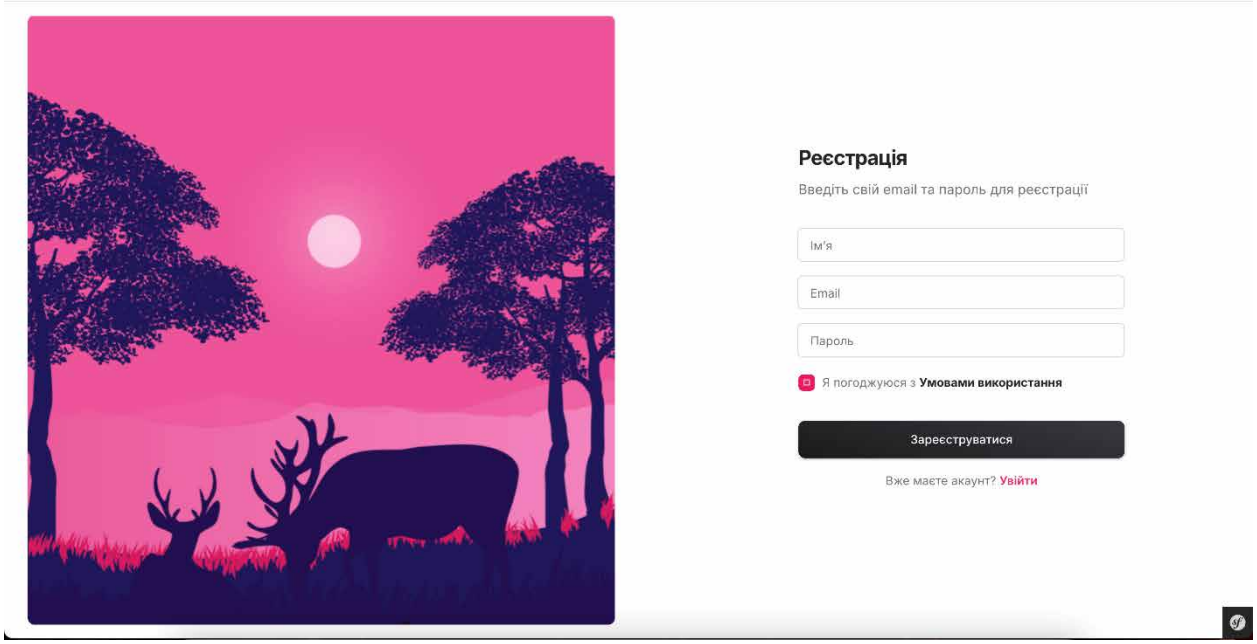


Рис. 4.2.11 Форма реєстрації

# ВИСНОВКИ

Робота над магістерською кваліфікаційною роботою на тему «Програмне забезпечення для оптимізації топології мережевих з’єднань» розпочалася з детального аналізу предметної області та визначення основних проблем, пов’язаних із проєктуванням і вдосконаленням мережевих структур. Було проведено огляд сучасних рішень, таких як Cisco Network Assistant та SolarWinds Network Topology Mapper, що дало змогу виявити їхні сильні сторони, обмеження та потенційні напрями для подальшого вдосконалення систем оптимізації мереж.

У ході дослідження було визначено ключові вимоги до системи, розроблено концептуальну та логічну моделі даних, які відображають структуру інформаційних потоків, зв’язки між вузлами мережі та механізми збереження параметрів. На основі цих моделей створено фізичну модель бази даних, реалізовану в середовищі MySQL із підтримкою аналітичних функцій через MS SQL Server для проведення інтелектуального аналізу даних.

У процесі розробки було застосовано алгоритм Дейкстри як основу для оптимізації топології мережевих з’єднань. Цей алгоритм забезпечив визначення найкоротших шляхів між вузлами, що дозволило мінімізувати затримки в передачі даних і підвищити ефективність функціонування мережі. Архітектура системи, побудована на Symfony (PHP), у поєднанні з HTML, CSS, JavaScript і Docker, забезпечила модульність, масштабованість і кросплатформність програмного продукту.

Розроблене програмне забезпечення пройшло тестування, яке підтвердило його стабільність, точність розрахунків і зручність для адміністратора мережі. Отримані результати доводять, що система може бути ефективно використана для автоматизації процесів проєктування та

оптимізації топології мереж різного масштабу — від локальних до корпоративних.

Попри досягнуті результати, подальші дослідження можуть бути спрямовані на інтеграцію методів машинного навчання для прогнозування навантажень у мережі, розширення підтримки розподілених систем і підвищення рівня безпеки збережених даних. Важливим аспектом також залишається забезпечення конфіденційності користувачів і захисту інформації, що є ключовими критеріями для сучасних ІТ-рішень у сфері мережевих технологій.

Отже, виконана робота зробила вагомий внесок у розвиток засобів автоматизації аналізу та оптимізації топології мережевих з’єднань. Розроблене програмне забезпечення може слугувати основою для подальших досліджень і практичних рішень, спрямованих на вдосконалення управління мережевими інфраструктурами.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Cisco Network Assistant – [Електронний ресурс] – Режим доступу: https://community.cisco.com/
2. SolarWinds NTM – [Електронний ресурс] – Режим доступу: https://www.solarwinds.com/network-topology-mapper
3. Дейкстра, Е. В. Примітка щодо двох проблем, пов'язаних з графами. // Numerische Mathematik. – 1959. – Т. 1, № 1. – С. 269–271.
4. Кормен, Т. Х., Лейсерсон, К. Е., Рівест, Р. Л., Штайн, К. Вступ до алгоритмів. – 3-тє вид. – MIT Press, 2009. – 1312 с.
5. Таненбаум, А. С., Везеролл, Д. Дж. Комп'ютерні мережі. – 5-те вид. – Pearson Education, 2011. – 960 с.
6. Документація MySQL – [Електронний ресурс] – Режим доступу: https://dev.mysql.com/doc/
7. Документація Symfony Framework – [Електронний ресурс] – Режим доступу: https://symfony.com/doc/current/index.html
8. Офіційна документація PHP – [Електронний ресурс] – Режим доступу: https://www.php.net/docs.php
9. Документація Docker – [Електронний ресурс] – Режим доступу: https://docs.docker.com/
10. Документація Visual Studio Code – [Електронний ресурс] – Режим доступу: https://code.visualstudio.com/docs
11. HTML Living Standard – [Електронний ресурс] – Режим доступу: https://html.spec.whatwg.org/
12. CSS: Каскадні таблиці стилів – [Електронний ресурс] – Режим доступу: https://www.w3.org/Style/CSS/
13. Посібник з JavaScript (MDN Web Docs) – [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

14. Алгоритми інтелектуального аналізу даних Microsoft SQL Server – [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/sql/analysis-services/data-mining/data-mining-algorithms
15. Хан, Дж., Камбер, М., Пей, Дж. Інтелектуальний аналіз даних: концепції та методи. – 3-тє вид. – Морган Кауфманн, 2012. – 744 с.
16. Ельмасрі, Р., Наватхе, С. Б. Основи систем баз даних. – 7-ме вид. – Пірсон, 2017. – 1200 с.
17. Соммервіль, І. Програмна інженерія. – 10-те вид. – Пірсон Едукашн, 2015. – 816 с.
18. Специфікація UML 2.5 – Група управління об'єктами – [Електронний ресурс] – Режим доступу: https://www.omg.org/spec/UML/
19. Прессман, Р. С., Максим, Б. Р. Програмна інженерія: підхід практикуючого спеціаліста. – 9-те видання – McGraw-Hill, 2019. – 976 с.
20. Контейнери Docker у розробці на PHP – [Електронний ресурс] – Режим доступу: https://www.cloudbees.com/blog/docker-php-development
21. What is Unified Modeling Language (UML)? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
22. Діаграма варіантів використання (UseCase diagram) – [Електронний ресурс] – Режим доступу: https://flexberry.github.io/ua/fd_use-case-diagram.html
23. What is Sequence Diagram? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/
24. UML - Activity Diagrams – Tutorialspoint – [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm
25. Побудова діаграми класів – [Електронний ресурс] – Режим доступу: https://flexberry.github.io/ua/gpg_class-diagram.html
26. Entity Relationship Diagram - Data Modeling – [Електронний ресурс] – Режим доступу:

https://www.visualparadigm.com/VPGallery/datamodeling/EntityRelationshipDiagram.html

27. Документація Bootstrap (офіційний сайт) – [Електронний ресурс] – Режим доступу: www.getbootstrap.com/documentation.
28. W3Schools – [Електронний ресурс] – Режим доступу: www.w3schools.com
29. Microsoft Docs. (2021). Шаблон багатошарової архітектури – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/layered
30. Що таке Data Mining? Ключові концепції, як це працює? - [Електронний ресурс]. - Режим доступу: https://www.upgrad.com/blog/what-is-datamining key-concepts-how-does-it-work/
31. Інструменти Data Mining для кращого аналізу даних - [Електронний ресурс]. – Режим доступу: https://www.ionos.co.uk/digitalguide/online-marketing/webanalytics/a-comparison-of-data-mining-tools/
32. Системи OLAP. – [Електронний ресурс]. - Режим доступу: https://pidru4niki.com/1670032447784/informatika/olap-sistemi
33. Han, J., Kamber, M., & Pei, J. (2011). Data Mining: Concepts and Techniques (3-тє вид.). Morgan Kaufmann Publishers.
34. Наївні баєсівські класифікатори - [Електронний ресурс]. - Режим доступу: https://www.geeksforgeeks.org/naive-bayes-classifiers/
35. Agrawal, R., Imielinski, T., & Swami, A. (1993). Правила асоціації між наборами елементів у великих базах даних. ACM SIGMOD Record, 22(2), 207-216.
36. Посібник з кластерного аналізу з прикладами - [Електронний ресурс]. – Режим доступу: https://www.resonio.com/market-research/cluster-analysis

# ДОДАТОК А

## Фрагменти програмного коду. Функція оптимізації алгоритмом Дейкстри

```
class OptimizationController extends AbstractController
{
    #[Route('/optimization', name: 'app_optimization')]
    public function index(Request $request, EntityManagerInterface $em): Response
    {
        $networks = $em->getRepository(Network::class)->findAll();
        $result = null;

        if ($request->isMethod('POST')) {
            $networkId = $request->request->get('network');
            $startNodeId = $request->request->get('startNode');
            $endNodeId = $request->request->get('endNode');

            $network = $em->getRepository(Network::class)->find($networkId);
            $startNode = $em->getRepository(Node::class)->find($startNodeId);
            $endNode = $em->getRepository(Node::class)->find($endNodeId);

            if (!$network || !$startNode || !$endNode) {
                flash()->warning('Будь ласка, оберіть усі необхідні параметри');
                return $this->redirectToRoute('app_optimization');
            }

            $connections = $em->getRepository(Connection::class)->findBy(['network' =>
$network]);

            $graph = [];
            foreach ($connections as $conn) {
                $n1 = $conn->getFirstNode()->getId();
                $n2 = $conn->getSecondNode()->getId();
```

```
    $weight = $conn->getCost() ?? 1;

    $graph[$n1][$n2] = $weight;
    $graph[$n2][$n1] = $weight;
}

[$distances, $previous] = $this->dijkstra($graph, $startNode->getId());

$path = [];
for ($at = $endNode->getId(); $at !== null; $at = $previous[$at] ?? null) {
    array_unshift($path, $at);
}

$optimizedLinks = [];
for ($i = 0; $i < count($path) - 1; $i++) {
    $from = $em->getRepository(Node::class)->find($path[$i]);
    $to = $em->getRepository(Node::class)->find($path[$i + 1]);
    $optimizedLinks[] = [
        'from' => $from->getNodeName(),
        'to' => $to->getNodeName(),
        'weight' => $graph[$path[$i]][$path[$i + 1]],
    ];
}

$totalWeight = $distances[$endNode->getId()] ?? null;
$totalCost = $totalWeight / 2;

$resultEntity = new OptimizationResult();
$resultEntity->setNetwork($network);
$resultEntity->setUser($this->getUser());
$resultEntity->setParameters([
    'startNode' => $startNode->getNodeName(),
    'endNode' => $endNode->getNodeName(),
```

```
        ]);
        $resultEntity->setTotalWeight($totalWeight);
        $resultEntity->setTotalCost($totalCost);
        $resultEntity->setOptimizedLinks($optimizedLinks);
        $resultEntity->setCreatedAt(new \DateTimeImmutable());

        $em->persist($resultEntity);
        $em->flush();

        $result = $resultEntity;
    }

    return $this->render('optimization/index.html.twig', [
        'networks' => $networks,
        'result' => $result,
    ]);
}

private function dijkstra(array $graph, int $source): array
{
    $dist = [];
    $prev = [];
    $queue = [];

    foreach ($graph as $vertex => $edges) {
        $dist[$vertex] = INF;
        $prev[$vertex] = null;
        $queue[$vertex] = INF;
    }

    $dist[$source] = 0;
    $queue[$source] = 0;
```

```
    while (!empty($queue)) {
        asort($queue);
        $u = key($queue);
        unset($queue[$u]);

        foreach ($graph[$u] ?? [] as $v => $cost) {
            $alt = $dist[$u] + $cost;
            if ($alt < ($dist[$v] ?? INF)) {
                $dist[$v] = $alt;
                $prev[$v] = $u;
                $queue[$v] = $alt;
            }
        }
    }

    return [$dist, $prev];
}
```

# ДОДАТОК Б

## Фрагменти програмного коду. Функція створення мережі

```
#[IsGranted('ROLE_USER')]

    #[Route('/network/create', name: 'app_network_create', methods: ['GET', 'POST'])]

    public function create(Request $request, EntityManagerInterface $entityManager): 
Response

    {

        if ($request->isMethod('POST')) {

            $data = $request->request->all();


            $network = new Network();

            $network->setNetworkName($data['networkName']);

            $network->setDescription($data['description'] ?? null);

            $network->setOwner($this->getUser());

            $network->setIsActive(true);

            $network->setCreatedAt(new \DateTimeImmutable());


            if (!empty($data['settings'])) {

                try {

                    $json = json_decode($data['settings'], true, 512, 
JSON_THROW_ON_ERROR);

                    $network->setSettings($json);

                } catch (\JsonException $e) {

                    flash()->warning('Невірний формат JSON у полі "Налаштування"');

                    return $this->redirectToRoute('app_network_create');

                }

            }


            $entityManager->persist($network);

            $entityManager->flush();


            flash()->success('Мережу успішно створено');

            return $this->redirectToRoute('app_network');
```

```
        }


        return $this->render('network/create.html.twig');

    }


#[Route('/network/toggle/{id}', name: 'app_network_toggle', methods: ['POST'])]

    public function toggleStatus(Network $network, EntityManagerInterface
$entityManager): JsonResponse

    {

        $network->setIsActive(!$network->isActive());

        $entityManager->flush();


        return new JsonResponse([

            'success' => true,

            'newStatus' => $network->isActive(),

        ]);

    }
```