НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
Комп'ютерних систем, мереж та кібербезпеки

_____ Касаткін Д.Ю., доц., к.пед.н.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Побудова домашнього файлового сервера на основі Samba/NFS»

Спеціальність 123 «Комп'ютерна інженерія»

Гарант освітньої програми

к.фіз.-мат.н., доц. _____ Нікітенко Є.В.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Коваленко О,Є.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Ісмаїлов Н.Д.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
Комп'ютерних систем, мереж та кібербезпеки

_____/ Касаткін Д.Ю, доцент, к.пед.н /

підпис

"_ "_____2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студент Ісмаїлов Назар Дмитрович

Спеціальність 123 «Комп'ютерна інженерія»

Тема бакалаврської кваліфікаційної роботи: Розробка засобів і конфігураційного управління мережею

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2248 "С"

Термін подання завершеної роботи на кафедру _____

(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань , які потрібно розробити:

Аналіз проблемної області, вибір та обґрунтування засобів для розробки системи, проектування інформаційної системи.

Дата видачі завдання "16" 12 2025р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Коваленко О.Є.

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

Завдання прийняв до виконання _____ Ісмаїлов Н.Д.

(підпис) (ПІБ студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
1	Отримання завдання бакалаврської кваліфікаційної роботи	16.12.2024	
2	Початок планування системи	23.01.2025 – 03.02.2025	
3	Побудова UML діаграм	25.03.2025-27.04.2025	
4	Розробка програми і тестування	08.03.2025-23.04.2025	
5	Написання пояснювальної записки	24.04.2025-16.05.2025	
6	Перевірка на плагіат	17.05.2025	
7	Відправка дипломної записки	17.05.2025	
8	Захист дипломної роботи	25.05.2025-15.05.2025	

Студент _____ __Ісмаїлов Назар___

(підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ Коваленко О.Є.

(підпис) (ПІБ)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. API – Application Programming Interface – програмний інтерфейс прикладного програмування
2. CLI – Command Line Interface – інтерфейс командного рядка
3. CMDB – Configuration Management Database – база даних управління конфігурацією
4. CSV – Comma-Separated Values – формат табличних даних із роздільником «кома»
5. GUI – Graphical User Interface – графічний інтерфейс користувача
6. ICMP – Internet Control Message Protocol – протокол керування повідомленнями в мережі Інтернет
7. IP – Internet Protocol – протокол міжмережевої адресації
8. JSON – JavaScript Object Notation – формат обміну структурованими даними
9. NETCONF – Network Configuration Protocol – протокол конфігураційного управління мережею
10. NCM – Network Configuration Management – конфігураційне управління мережею
11. OSI – Open Systems Interconnection – еталонна модель взаємодії відкритих систем
12. PBM – Policy-Based Management – політик-орієнтоване управління
13. RESTCONF – RESTful Configuration Protocol – REST-орієнтований протокол конфігурацій
14. SDN – Software-Defined Networking – програмно-детермінована мережа
15. SNMP – Simple Network Management Protocol – простий протокол управління мережею
16. SSH – Secure Shell – захищений протокол для віддаленого доступу

17. TLS/SSL – Transport Layer Security / Secure Sockets Layer – протоколи захисту передавання даних

18. UTP – Unshielded Twisted Pair – неекранована вита пара

19. VLAN – Virtual Local Area Network – віртуальна локальна мережа

20. YANG – Yet Another Next Generation – модель опису даних для конфігурацій мережевих пристроїв

ВСТУП

Стрімкий розвиток інформаційних технологій, зокрема мережевих інфраструктур, обумовлює зростання складності процесів управління конфігурацією мережевих систем. Сучасні мережі характеризуються динамічністю, багаторівневістю та використанням різноманітних апаратних і програмних рішень, що зумовлює необхідність впровадження ефективних засобів для централізованого та автоматизованого конфігураційного управління [1, 3]. Традиційні підходи, засновані на ручному адмініструванні, вже не відповідають вимогам до гнучкості, масштабованості та безпеки сучасних корпоративних і хмарних мереж [8, 27].

Актуальність теми зумовлена потребою у створенні інструментів, здатних забезпечити стабільне функціонування мереж шляхом оперативного управління конфігураціями, запобігання помилкам налаштування та підвищення рівня безпеки [2, 23]. За даними досліджень, до 80% інцидентів у мережевих інфраструктурах пов’язані саме з некоректною конфігурацією обладнання та програмного забезпечення [24]. Це підкреслює важливість впровадження систем конфігураційного управління, які підтримують стандартизовані протоколи (наприклад, NETCONF, RESTCONF) і моделі даних (YANG) [4, 6].

Стан задачі у світовій практиці демонструє активне впровадження концепцій програмно-орієнтованих мереж (SDN) та політик-орієнтованого управління (Policy-Based Management), що дозволяє значно підвищити ефективність адміністрування складних мережевих середовищ [9, 26]. Проте існуючі рішення часто є надмірно складними для малих і середніх підприємств або не враховують специфічні вимоги безпеки й інтеграції в гетерогенні системи [10, 22]. Саме тому постає завдання розробки адаптивного, масштабованого та безпечного засобу конфігураційного управління мережею.

Підставами для розробки теми стали існуючі прогалини у застосуванні універсальних і доступних інструментів управління конфігурацією, здатних інтегруватися з різними типами мережевого обладнання та забезпечити автоматизацію основних адміністративних процесів [11, 30]. Вихідними даними для проведення дослідження є стандарти ISO/IEC у сфері мережевої безпеки та управління (зокрема, ISO/IEC 27033-1:2015 [32]), специфікації протоколів конфігураційного управління, а також аналіз існуючих програмних рішень від провідних виробників (Cisco, Juniper, Huawei) [40].

У зв'язку з викладеним, визначено такі основні компоненти дослідження:

Об'єкт дослідження – процеси конфігураційного управління мережевими інфраструктурами.

Предмет дослідження – методи, моделі та програмні засоби автоматизованого конфігураційного управління мережею із забезпеченням гнучкості, безпеки та масштабованості.

Мета роботи – розробка програмного засобу конфігураційного управління мережею, який забезпечує централізоване адміністрування, автоматизацію налаштувань, моніторинг змін конфігурації та відповідність сучасним стандартам безпеки.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати сучасні підходи та засоби конфігураційного управління мережами [12, 17];

– сформувати концептуальну модель системи з урахуванням вимог безпеки та стандартів [31, 34];

– реалізувати програмний прототип із підтримкою протоколів NETCONF/RESTCONF та моделей YANG [5, 6];

– провести тестування ефективності й безпеки розробленого засобу [20, 28];

– здійснити порівняльний аналіз із існуючими рішеннями [18, 35].

Наукова новизна одержаних результатів полягає у створенні методики конфігураційного управління мережею, яка поєднує стандартизовані протоколи

(NETCONF, RESTCONF) та моделі даних (YANG) із механізмами автоматизованого моніторингу і динамічного реагування на зміни в мережевій інфраструктурі [4, 5, 7]. Запропоновано інтеграцію політик-орієнтованого управління (Policy-Based Management) із системами забезпечення безпеки для зниження ризиків, пов'язаних із помилками конфігурації [9, 12]. Вперше розроблено адаптивний підхід до управління конфігураціями в гетерогенних середовищах із можливістю масштабування та забезпечення відповідності стандартам ISO/IEC 27033-1:2015 та ISO/IEC 19086-1:2016 [31, 32].

Практичне значення отриманих результатів полягає у створенні програмного засобу, що дозволяє автоматизувати процеси конфігураційного управління для мереж різної складності. Розроблена система може бути використана в корпоративних мережах, дата-центрах і хмарних середовищах для підвищення ефективності адміністрування, забезпечення цілісності конфігурацій та контролю відповідності політикам безпеки [20, 27]. Впровадження запропонованого рішення дозволяє знизити витрати на обслуговування мережевої інфраструктури та мінімізувати простої, спричинені помилками конфігурації [23, 24].

Апробація результатів дослідження здійснювалася шляхом тестування розробленої системи у лабораторному середовищі,

Загальний обсяг роботи становить ___ сторінок, включає перелік із 40 використаних джерел та додатки, що містять схемні рішення, лістинги програмного коду та результати тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасне конфігураційне управління мережами (Network Configuration Management, NCM) є ключовим компонентом ефективної експлуатації мережевої інфраструктури в умовах динамічного розвитку інформаційних технологій. Зростання кількості мережевих пристроїв, ускладнення топологій, інтеграція хмарних сервісів, технологій Інтернету речей (IoT) та розподілених обчислень сформували нові вимоги до процесів адміністрування мереж. У цьому контексті конфігураційне управління відіграє визначальну роль у забезпеченні стабільності, безпеки та відповідності інфраструктури бізнес-потребам [1, 3, 27].

На початкових етапах розвитку мереж основним підходом до управління конфігураціями було ручне налаштування кожного пристрою окремо, здебільшого через інтерфейс командного рядка (CLI). Такий підхід є трудомістким, схильним до помилок і не забезпечує належної узгодженості в масштабі великих мережевих систем [8]. Зі збільшенням складності мереж цей підхід вичерпав свої можливості, що обумовило необхідність автоматизації процесів конфігурації.

Сучасний стан конфігураційного управління характеризується переходом до централізованих та автоматизованих рішень, що базуються на стандартизованих протоколах і моделях даних. Провідними є протоколи NETCONF і RESTCONF, які дозволяють здійснювати управління конфігурацією пристроїв за допомогою уніфікованих механізмів доступу та структурованих моделей даних, таких як YANG [4, 5, 6]. Ці технології забезпечують високу ступінь формалізації процесів конфігурації, дозволяючи уникнути розбіжностей між різними елементами інфраструктури.

Діаграма (Рисунок 1.1) відображає основні аспекти сучасного стану конфігураційного управління мережами, серед яких ключовими є:

- динамічність і складність мережевих середовищ.

– Автоматизація процесів як відповідь на потребу в зниженні впливу людського фактора.

– Подолання технічних викликів, пов’язаних із сумісністю та масштабованістю.

– Забезпечення безпеки конфігураційних змін і відповідності політикам.

Рисунок 1.1 – Основні аспекти сучасного стану конфігураційного управління мережами

Важливим етапом еволюції стало впровадження концепції Software-Defined Networking (SDN), що передбачає відокремлення контрольної площини від площини передачі даних, забезпечуючи централізоване управління конфігурацією всього мережевого середовища [26]. Це дозволяє значно спростити адміністрування, забезпечити гнучкість та адаптивність до змінних умов функціонування мережі.

Не менш важливою тенденцією є розвиток Policy-Based Management (PBM) – політик-орієнтованого управління, де адміністрування конфігурацій здійснюється на основі заданих правил і політик, що автоматично регламентують допустимі зміни в мережі [9, 11]. Такий підхід мінімізує втручання людини в

рутинні процеси та підвищує узгодженість налаштувань відповідно до корпоративних стандартів і вимог безпеки [12, 15].

Однак, попри значний прогрес, конфігураційне управління все ще стикається з рядом технічних викликів, представленими у таблиці 1.1.

Таблиця 1.1 - Виклики сучасного конфігураційного управління мережами

Виклик	Сутність проблеми	Наслідки	Шляхи вирішення
Сумісність обладнання	Різні протоколи й інтерфейси управління у вендорів (Cisco, Juniper, Huawei тощо) [30, 40]	Ускладнення інтеграції, зростання витрат на адміністрування	Використання стандартів NETCONF, RESTCONF, моделей YANG [4, 5]
Масштабованість	Зростання кількості пристроїв, сервісів і потреба в управлінні великими мережами [20, 33]	Зниження продуктивності, ризик втрати контролю	Централізоване управління, SDN-рішення [26, 27]
Інформаційна безпека	Ризики несанкціонованих змін, внутрішніх загроз та вразливостей в автоматизованих процесах [7, 25, 32]	Порушення цілісності мережі, витоки даних	Впровадження політик безпеки, аудит змін, відповідність ISO [31]
Відстеження та контроль конфігурацій	Відсутність єдиного механізму логування та моніторингу змін у гетерогенних середовищах [19, 28]	Неможливість швидко локалізувати помилки, складність відновлення конфігурації	Інтеграція систем моніторингу та CMDB-рішень
Динамічність і адаптивність	Постійні зміни у топології мереж, навантаженнях і вимогах до сервісів [1, 3, 16]	Нестабільність роботи при відсутності автоматизації	Використання автоматизованих сценаріїв, DevOps-підходів [34]
Підтримка хмарних і гібридних рішень	Необхідність єдиного управління для локальних, хмарних і	Розрізнене адміністрування, зростання витрат	Оркестрація сервісів, інтеграція з хмарними API

	гібридних інфраструктур [20, 33]		

Продовження таблиці 1.1

Людський фактор	Помилки адміністраторів при ручному налаштуванні конфігурацій [24]	Високий відсоток інцидентів, пов’язаних із неправильними налаштуваннями	Автоматизація процесів, політик-орієнтоване управління [9, 12]

Крім того, актуальним є питання інтеграції систем конфігураційного управління з інструментами оркестрації та управління сервісами, що дозволяє створювати єдине інформаційне середовище для адміністрування IT-інфраструктури [19, 28]. У цьому напрямі активно розвиваються рішення на базі DevOps-підходів, де конфігурації мереж розглядаються як код (Infrastructure as Code, IaC), що забезпечує версіонування, тестування та автоматизоване розгортання змін [34].

1.2 Огляд існуючих засобів управління мережевою інфраструктурою

Ефективне управління мережевою інфраструктурою є критичним аспектом забезпечення стабільності, безпеки та продуктивності сучасних IT-систем. Зі зростанням складності мереж виникла необхідність у спеціалізованих засобах, що дозволяють централізовано адмініструвати конфігурації, здійснювати моніторинг, забезпечувати автоматизацію процесів і контроль відповідності політикам безпеки [20, 26, 33]. Сучасні рішення орієнтовані на інтеграцію з технологіями SDN, підтримку стандартів (NETCONF, RESTCONF, YANG), а також використання аналітики та автоматизації для зниження впливу людського фактора [4, 5, 9].

Одним із провідних рішень є Cisco DNA Center — платформа централізованого управління корпоративними мережами, що реалізує концепцію

Intent-Based Networking (IBN). Вона забезпечує автоматизацію налаштувань, моніторинг стану мережі, аналітику на основі штучного інтелекту та впровадження політик безпеки [40].

На рисунку 1.2 наведено інтерфейс Cisco DNA Center, який демонструє панель управління здоров'ям клієнтів (Client Health) та аналітику підключень. Візуалізована топологія мережі дозволяє адміністратору оперативно ідентифікувати проблемні ділянки, контролювати фізичні та логічні з'єднання, а також аналізувати показники якості зв'язку (RSSI, фізичні лінки) у режимі реального часу. Інтеграція з аналітичними модулями дає змогу передбачати потенційні відмови й оптимізувати ресурси мережі.

Рисунок 1.2 – Інтерфейс Cisco DNA Center із відображенням стану мережі та аналітики клієнтів

Contrail від Juniper Networks є масштабованим рішенням для управління мережами в хмарних і віртуалізованих середовищах. Архітектура Contrail базується на SDN та NFV, що дозволяє централізовано контролювати конфігурації мережевих функцій, забезпечуючи інтеграцію з оркестраційними платформами, такими як OpenStack [30].

Рисунок 1.3 ілюструє архітектурну схему Contrail, де відображено взаємодію між контрольним рівнем, аналітикою, конфігураційними модулями та

віртуалізованими мережевими функціями. Використання відкритих API та стандартів (BGP, XMPP, NETCONF) забезпечує гнучкість і сумісність з різними типами інфраструктур.

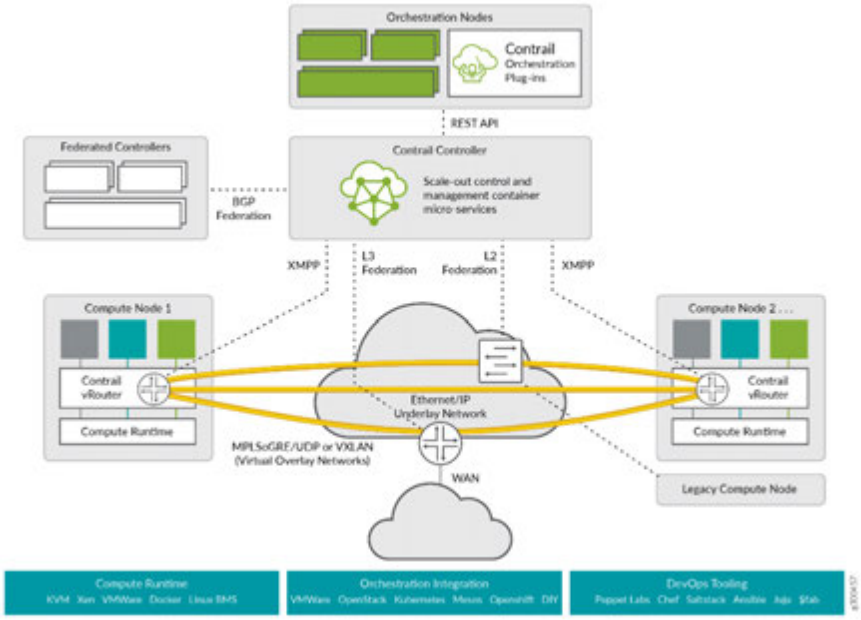

Рисунок 1.3 – Архітектура SDN-рішення Juniper Contrail для управління віртуалізованими мережами

Ansible є універсальним інструментом для автоматизації конфігураційного управління, який реалізує концепцію Infrastructure as Code (IaC). Платформа дозволяє створювати сценарії (playbooks) для налаштування мережевих пристроїв, серверів та сервісів різних виробників [34].

На рисунку 1.4 показано структуру Ansible Automation Platform, де центральне місце займає контролер автоматизації, що управляє виконанням завдань, зберіганням сценаріїв та взаємодією з хмарними середовищами, дата-центрами та edge-інфраструктурою. Це рішення забезпечує масштабованість і повторюваність конфігураційних процесів.

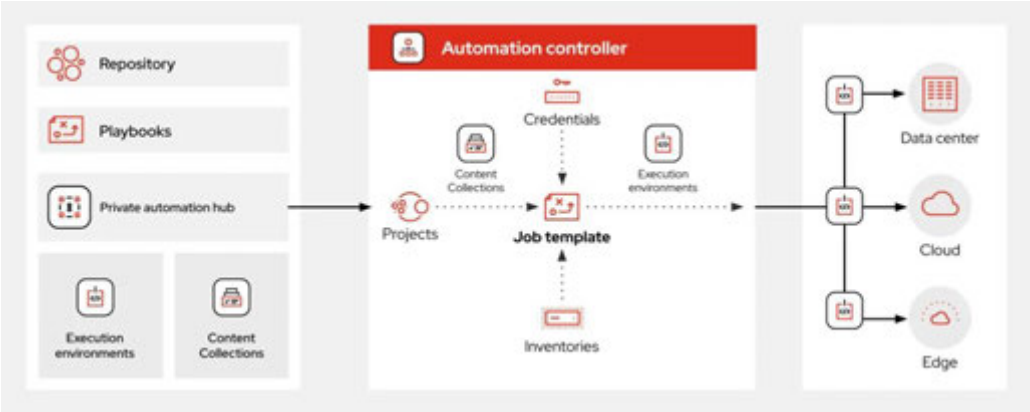

Рисунок 1.4 – Архітектура Red Hat Ansible Automation Platform для автоматизації управління інфраструктурою

Puppet Enterprise — це ще одне потужне рішення для автоматизації управління конфігураціями, орієнтоване на великі корпоративні мережі. Платформа дозволяє централізовано контролювати стан інфраструктури, забезпечуючи відповідність налаштувань корпоративним стандартам і політикам безпеки [34].

Рисунок 1.5 демонструє Compliance Dashboard у Puppet Enterprise, який відображає рівень відповідності вузлів встановленим політикам, дозволяє виявляти відхилення та керувати винятками. Такий підхід сприяє зниженню ризиків, пов'язаних із неконтрольованими змінами конфігурацій, та забезпечує повний аудит.

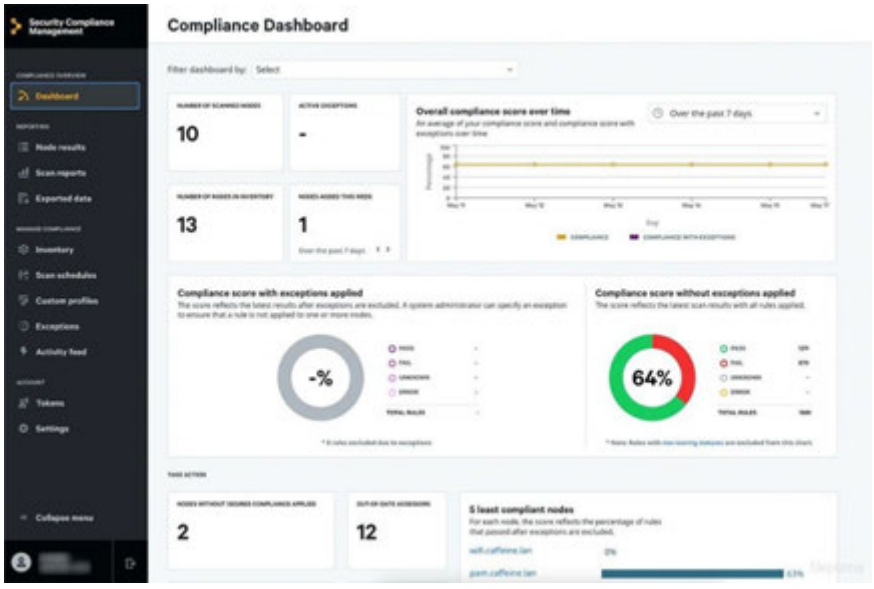

Рисунок 1.5 – Панель контролю відповідності (Compliance Dashboard) у Puppet Enterprise)

Huawei eSight — комплексна платформа управління мережевою інфраструктурою, яка забезпечує моніторинг, конфігураційне управління, а також управління сервісами та безпекою [40]. Система підтримує візуалізацію топології, централізоване налаштування пристроїв та інтеграцію з іншими IT-системами.

На рисунку 1.6 представлено приклад інтерфейсу Huawei eSight із візуалізацією мережевих з'єднань та статусів пристроїв. Такий підхід дозволяє оперативно реагувати на зміни в інфраструктурі та виявляти потенційні проблеми ще до їх критичного впливу на роботу мережі.

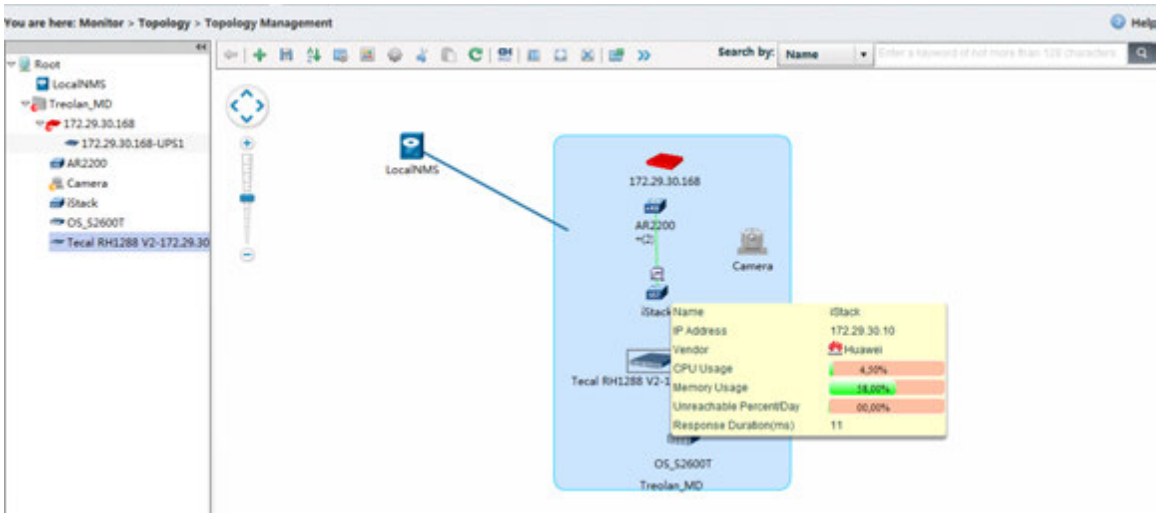

Рисунок 1.6 – Візуалізація топології мережі у системі Huawei eSight

Для більш структурного аналізу необхідно звісно зробити порівняння можливостей цих систем, яке можна побачити у таблиці 1.2.

Таблиця 1.2 - Порівняльна таблиця основних засобів управління мережевою інфраструктурою з коротким післямовним аналізом

Характеристика	Cisco DNA Center	Juniper Contrail	Ansible Automation	Puppet Enterprise	Huawei eSight
Тип рішення	Комерційне	Комерційне / Open Source	Open Source / Комерційне	Комерційне	Комерційне
Підхід	SDN, IBN	SDN, NFV	Infrastructure as Code	Declarative Automation	Централізоване управління

Продовження таблиці 1.2

Автоматизація	Висока	Висока	Повна (через playbooks)	Висока	Висока
Підтримка стандартів	NETCONF, RESTCONF, YANG	NETCONF, BGP, XMPP	NETCONF, RESTCONF	Часткова підтримка	SNMP, NETCONF
Інтеграція з хмарою	Так	Так	Так	Обмежена	Так
Візуалізація топології	Так	Так	Ні	Ні	Так
Контроль політик	Вбудований	Вбудований	Через сценарії	Вбудований	Вбудований
Моніторинг у реальному часі	Так	Так	Ні	Частково	Так
Гнучкість налаштувань	Висока	Висока	Дуже висока	Висока	Середня
Сумісність з обладнанням інших вендорів	Обмежена	Висока	Висока	Висока	Обмежена
Цільова аудиторія	Великі корпоративні мережі	Хмарні середовища, дата-центри	Універсальне рішення	Великі IT-інфраструктури	Корпоративні мережі

Сучасні засоби управління мережевою інфраструктурою пропонують різні підходи до вирішення завдань автоматизації, моніторингу та забезпечення безпеки. Такі платформи, як Cisco DNA Center і Huawei eSight, орієнтовані на комплексне комерційне рішення з глибокою інтеграцією у власне обладнання. Натомість Juniper Contrail демонструє більшу відкритість і гнучкість для хмарних та віртуалізованих середовищ. Засоби на зразок Ansible і Puppet надають високу гнучкість завдяки підходу до інфраструктури як коду, що дозволяє ефективно автоматизувати управління у гетерогенних середовищах.

Вибір оптимального інструменту залежить від специфіки мережевої інфраструктури, вимог до автоматизації, політик безпеки та доступного бюджету. Тенденції розвитку таких систем спрямовані на подальшу інтеграцію

з хмарними технологіями, розширення можливостей штучного інтелекту для аналітики й самоуправління, а також дотримання стандартів відкритих інтерфейсів для забезпечення сумісності [20, 26, 34, 40].

1.3 Аналіз вимог до систем конфігураційного управління

Проєктування ефективної системи конфігураційного управління мережею (СКУМ) вимагає чіткого визначення вимог, що забезпечують її функціональність, надійність, безпеку та відповідність сучасним стандартам. Вимоги до таких систем формуються на основі міжнародних стандартів, зокрема ISO/IEC 27033-1:2015, ISO/IEC 19086-1:2016, а також рекомендацій щодо управління конфігураціями в динамічних і гібридних мережах [31, 32, 34].

Функціональні вимоги визначають основні можливості системи, які забезпечують реалізацію її ключових завдань. До них відносяться засоби централізованого управління конфігураціями, автоматизація процесів, моніторинг та інтеграція з іншими компонентами IT-інфраструктури. Узагальнені функціональні вимоги наведено в таблиці 1.3.

Таблиця 1.3 – Функціональні вимоги до СКУМ

№	Вимога	Опис
1	Централізоване управління	Можливість керування всіма мережевими пристроями з єдиного інтерфейсу
2	Автоматизація конфігураційних змін	Підтримка сценаріїв автоматизованого налаштування та оновлення конфігурацій
3	Моніторинг у реальному часі	Відстеження стану мережі та конфігураційних параметрів із оперативним сповіщенням
4	Інтеграція з системами оркестрації та хмарними сервісами	Підтримка API для взаємодії з зовнішніми платформами
5	Контроль версій конфігурацій	Збереження історії змін та можливість відновлення попередніх станів

Продовження таблиці 1.3

6	Підтримка стандартів NETCONF, RESTCONF, YANG	Використання уніфікованих протоколів і моделей даних для сумісності

Нефункціональні вимоги визначають характеристики, що впливають на якість роботи системи, включаючи продуктивність, масштабованість і зручність користування. Вони наведені в таблиці 1.4.

Таблиця 1.4 – Нефункціональні вимоги до СКУМ

№	Вимога	Опис
1	Масштабованість	Забезпечення стабільної роботи при зростанні кількості керованих пристроїв
2	Висока доступність	Мінімальний час простою системи
3	Відмовостійкість	Автоматичне відновлення після збоїв
4	Простота інтеграції	Сумісність із різними типами обладнання та програмних рішень
5	Інтуїтивно зрозумілий інтерфейс	Зручність для адміністратора без потреби глибоких знань програмування
6	Продуктивність	Швидкість обробки конфігураційних змін і запитів

Вимоги до безпеки

Безпека є критично важливим аспектом для будь-якої СКУМ, оскільки неконтрольовані або несанкціоновані зміни можуть призвести до серйозних інцидентів у мережі. Основні вимоги до безпеки системи наведено в таблиці 1.5.

Таблиця 1.5 – Вимоги до безпеки СКУМ

№	Вимога	Опис
1	Аутентифікація та авторизація	Розмежування прав доступу на основі ролей
2	Шифрування даних	Використання протоколів TLS/SSL для захисту переданих конфігурацій
3	Аудит та логування	Фіксація всіх дій користувачів і системних змін
4	Захист від несанкціонованих змін	Механізми виявлення та блокування підозрілих дій
5	Відповідність стандартам ISO/IEC	Виконання вимог інформаційної безпеки згідно з міжнародними нормами [31, 32]

Технічні вимоги визначають апаратні та програмні ресурси, необхідні для ефективного функціонування системи. Вони представлені в таблиці 1.6.

Таблиця 1.6 – Технічні вимоги до СКУМ

№	Вимога	Опис
1	Підтримка серверної інфраструктури	Можливість розгортання на фізичних або віртуальних серверах
2	Сумісність з ОС	Підтримка Linux, Windows
3	Використання контейнеризації	Підтримка Docker/Kubernetes для гнучкого розгортання
4	Підключення до мережевого обладнання	Підтримка SNMP, SSH, API
5	База даних	Використання SQL/NoSQL для зберігання конфігураційних даних
6	Резервування	Наявність механізмів резервного копіювання і відновлення

Аналіз вимог до систем конфігураційного управління дозволяє визначити ключові параметри, які мають забезпечити ефективність, надійність і безпеку функціонування СКУМ у сучасних умовах. Комплексний підхід до формування функціональних, нефункціональних, безпекових і технічних вимог гарантує створення системи, здатної забезпечити централізоване управління мережевими конфігураціями з урахуванням вимог масштабованості, автоматизації та відповідності міжнародним стандартам [20, 31, 34]. Подальше проєктування системи базуватиметься саме на цих вимогах, що забезпечить її ефективну інтеграцію у різнорідні IT-інфраструктури.

1.4 Вибір засобів та технологій для реалізації системи

Вибір засобів і технологій для розробки системи конфігураційного управління мережею (СКУМ) є одним із ключових етапів, що визначає ефективність, гнучкість і масштабованість майбутнього рішення. З урахуванням сформульованих функціональних, нефункціональних, безпекових та технічних вимог (див. пункти 1.3, таблиці 1.3–1.6), доцільно обрати стек технологій, який забезпечить підтримку сучасних стандартів, можливості автоматизації та інтеграції з різними типами мережевої інфраструктури [20, 31, 34].

Зважаючи на універсальність, кросплатформеність та широку підтримку мережевих і автоматизаційних бібліотек, основною мовою програмування для реалізації СКУМ обрано Python. Python є одним із найбільш придатних інструментів для створення систем управління завдяки своїй розвиненій екосистемі, легкості інтеграції з мережевим обладнанням через стандартні протоколи та підтримці DevOps-підходів [34].

Основні технології та бібліотеки для реалізації СКУМ на Python

1. Netmiko

Бібліотека для спрощення взаємодії з мережевими пристроями через SSH. Підтримує широкий спектр вендорів (Cisco, Juniper, HP, Huawei тощо). Використовується для автоматизації базових конфігураційних операцій [40].

2. NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support)

Потужний інструмент для мультивендорного управління мережами, що підтримує підключення через різні драйвери (NAPALM IOS, EOS, JunOS). Забезпечує уніфікований підхід до читання та внесення змін у конфігурацію [34].

3. PyYANG / yangson

Бібліотеки для роботи з моделями даних YANG. Дозволяють здійснювати парсинг, валідацію та генерацію конфігурацій згідно зі стандартами NETCONF/RESTCONF [5, 6].

4. Requests / HTTPx

Використовуються для інтеграції з REST API мережевих пристроїв та систем оркестрації. Забезпечують взаємодію із зовнішніми сервісами відповідно до RESTCONF-протоколу [4].

5. Paramiko

Бібліотека для роботи з SSH-з’єднаннями, яка може бути використана для низькорівневої автоматизації у випадках, де Netmiko є надлишковим [40].

6. Celery + Redis/RabbitMQ

Для організації асинхронної обробки завдань і реалізації черг конфігураційних змін. Це забезпечить масштабованість і стійкість системи до навантажень [34].

7. Flask / FastAPI

Легковагові фреймворки для створення веб-інтерфейсу та RESTful API системи. FastAPI є більш сучасним рішенням із високою продуктивністю та підтримкою OpenAPI [34].

8. SQLAlchemy + PostgreSQL / MongoDB

Використовуватимуться для зберігання конфігураційних даних, журналів змін та користувацьких політик. Вибір між реляційною (PostgreSQL) та документною (MongoDB) базою залежить від особливостей структури даних [34].

9. Jinja2

Шаблонізатор, що дозволяє створювати динамічні конфігурації пристроїв, використовуючи шаблони. Широко застосовується у поєднанні з Ansible та NAPALM [34].

10. Docker + Docker Compose

Для контейнеризації компонентів системи з метою спрощення розгортання, масштабування та ізоляції середовищ [34].

11. Prometheus + Grafana

Інтеграція системи моніторингу для відстеження стану мережі, продуктивності СКУМ і візуалізації метрик у реальному часі [34].

12. LDAP / OAuth 2.0

Технології для реалізації механізмів аутентифікації та авторизації користувачів, що забезпечать відповідність вимогам безпеки, зазначеним у таблиці 1.3 [31, 32].

Ключовими протоколами для взаємодії з мережевим обладнанням будуть:

- NETCONF — для структурованого управління конфігураціями через XML-повідомлення.

- RESTCONF — як полегшена REST-орієнтована альтернатива NETCONF для сучасних API.
- SNMP — для базового моніторингу та збору статистики з пристроїв.
- SSH — як основний захищений канал для доступу до пристроїв із CLI-інтерфейсом.

Вибір Python як основної платформи для розробки системи конфігураційного управління обумовлений його гнучкістю, підтримкою мультивендорного середовища та багатою екосистемою бібліотек для автоматизації, роботи з мережами й інтеграції з сучасними стандартами. Запропонований стек технологій забезпечить реалізацію всіх функціональних та нефункціональних вимог (див. таблиці 1.3–1.6), дозволяючи створити масштабовану, безпечну й ефективну систему, здатну до інтеграції у корпоративні, хмарні та гібридні інфраструктури [20, 31, 34, 40].

Подальші етапи розробки передбачають проєктування архітектури системи, побудову її компонентної моделі та визначення сценаріїв взаємодії з мережевим обладнанням і зовнішніми сервісами.

2 РЕАЛІЗАЦІЯ КОРПОРАТИВНОЇ МЕРЕЖІ

2.1 Вибір технологій для ліній зв’язку

Вибір технологій ліній зв’язку є базовим етапом проєктування корпоративної мережі, оскільки саме фізичний рівень визначає пропускну здатність, надійність, топологічні обмеження та вартість реалізації інфраструктури. Відповідно до сучасних вимог щодо швидкодії, масштабованості та підтримки гібридного доступу, доцільним є комбіноване використання дротових (мідних і оптоволоконних) та бездротових технологій. Такий підхід забезпечує повне охоплення потреб різних зон організації: від серверної до робочих місць користувачів і мобільних пристроїв.

Відповідно до нормативних та методичних рекомендацій [1; 2; 30; 34], при побудові структурованої кабельної системи рекомендується реалізовувати багаторівневу модель, що включає магістральні, вертикальні (міжповерхові) та горизонтальні лінії. Для кожного рівня слід використовувати відповідне середовище передавання, що обрано на основі технічних характеристик, довжини каналу, потреб у швидкодії та специфіки пристроїв, які підключаються.

У контексті запропонованої мережевої інфраструктури доцільно реалізувати наступний розподіл технологій, представлений у таблиці 2.1.

Таблиця 2.1 - Характеристики та застосування фізичних середовищ передавання

№	Тип середовища	Максимальна швидкість	Дальність з’єднання	Призначення застосування
1	Вита пара UTP (Cat.6e)	до 10 Гбіт/с	до 100 м	Підключення ПК, точок доступу, комутаторів
2	Оптоволокно (Single/Multi)	до 100 Гбіт/с	до 10–40 км	Магістраль між корпусами, поверхами, дата-центр
3	Бездротове з’єднання (Wi-Fi 5/6)	до 9.6 Гбіт/с	до 50 м (в приміщенні)	Підключення мобільних пристроїв, IoT

У Cisco Packet Tracer створено умовну візуалізацію базового набору компонентів, які реалізують вибір згаданих технологій. Як зображено на **рис. 2.1**, мінімальна топологія включає: ПК, дротовий комутатор, маршрутизатор, бездротовий маршрутизатор (WRT300N), а також емулятор зовнішньої мережі Cloud.

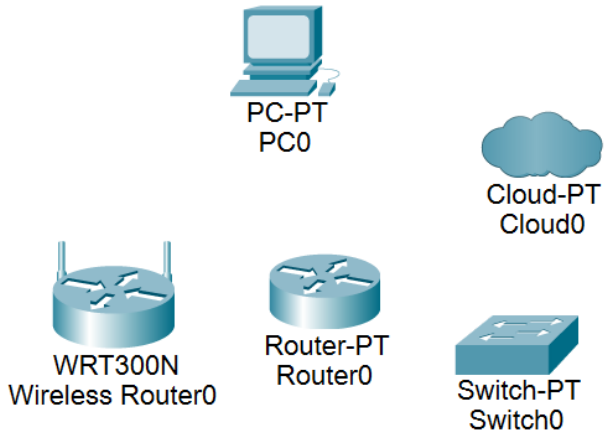

Рисунок 2.1 – Початкова схема вибраних середовищ передавання у Cisco Packet Tracer

Ця схема дозволяє реалізувати як дротове з'єднання через виту пару, так і бездротове з'єднання за допомогою стандарту 802.11. Компонент Cloud використовується для тестування взаємодії із зовнішнім провайдером або хмарною службою [20].

Окремо варто розглянути логічну функціональну модель системи конфігураційного управління мережею (СКУМ), яка відображена на **рис. 2.2**. Система побудована на стеку Python-технологій, що забезпечують взаємодію з мережевими пристроями через протоколи NETCONF, RESTCONF, SNMP, а також реалізацію REST API для користувачів, шаблонізацію конфігурацій (Jinja2), асинхронну обробку запитів (Celery), контейнеризацію (Docker), моніторинг і авторизацію [30], [31], [34].

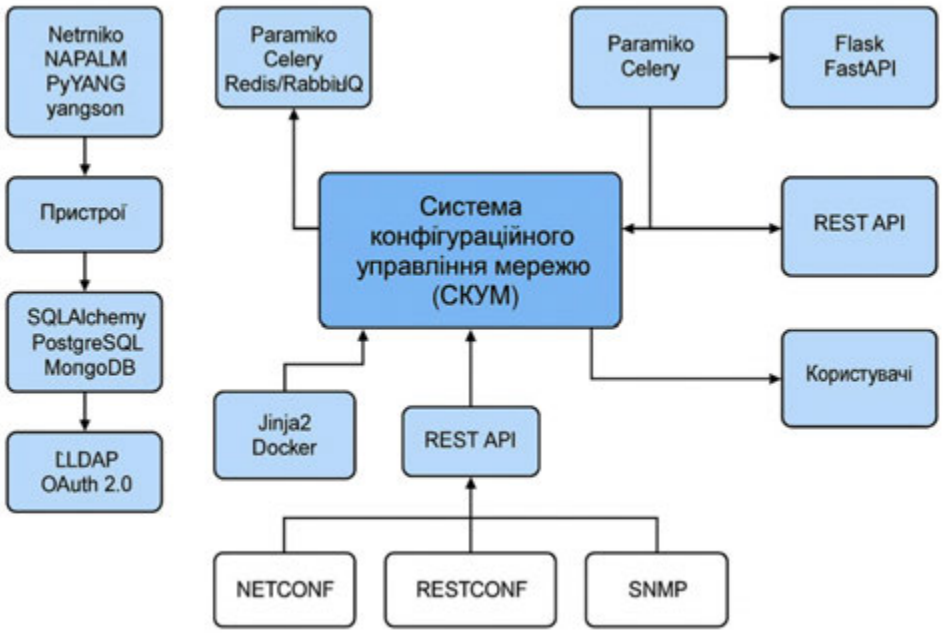

Рисунок 2.2 – Функціональна схема системи конфігураційного управління мережею (СКУМ)

Як видно зі схеми, компоненти взаємодіють між собою через уніфіковані інтерфейси: обмін конфігураціями здійснюється через Netmiko/NAPALM або REST API, управління доступом реалізоване через LDAP та OAuth 2.0, а дані зберігаються у PostgreSQL чи MongoDB (раціональність використання бази розглянута детальніше в третьому розділі [40].

Загалом, поєднання фізичних технологій (мідь, оптика, Wi-Fi) з розширюваною логікою програмного управління створює основу для реалізації надійної, масштабованої та керованої мережі, адаптованої до умов сучасного підприємства [2; 5; 6; 20; 30; 31; 34].

2.2 Вибір мережних пристроїв

У процесі проєктування системи конфігураційного управління мережею (СКУМ) важливою складовою є раціональний вибір активного мережного обладнання. Конфігурація пристроїв має відповідати вимогам функціональної

гнучкості, масштабованості, надійності обміну даними та сумісності з обраним програмно-апаратним стеком (див. рис. 2.2, 2.3).

Згідно з проєктом, система включає наступні пристрої:

- Router-PT (Router0) — виконує маршрутизацію між локальним сегментом та зовнішніми мережами через Cloud-PT. Має підтримку кількох FastEthernet і Serial-інтерфейсів, що дозволяє масштабувати топологію. З'єднання здійснено через FastEthernet0/0 до Switch-PT.
- Switch-PT (Switch0) — забезпечує комутацію в межах локального сегмента. До нього підключаються сервер, користувацькі пристрої та маршрутизатор. Використано порти FastEthernet0/1–0/4.
- Server-PT (Server0) — виконує роль хосту для вебінтерфейсу СКУМ, а також бази даних конфігурацій.
- Cloud-PT (Cloud0) — модельований зовнішній шлюз провайдера або віддалений сегмент для перевірки роботи маршрутизації.
- Wireless Router (WRT300N) — використовується для реалізації бездротового доступу до мережі для мобільних пристроїв. У моделі підключено TabletPC-PT (Tablet PC0) через Wi-Fi. Підключення самого маршрутизатора до мережі здійснено через порт Ethernet1 (порт Internet тимчасово не використовується через обмеження в Packet Tracer).
- PC-PT (PC0), Laptop-PT (Laptop0) — дротові клієнтські пристрої, підключені до Switch-PT через стандартні FastEthernet-кабелі.
- TabletPC-PT — бездротовий клієнт, що з'єднується з WRT300N на рівні Wi-Fi (режим інфраструктури).

Для реалізації бездротового доступу до мережі мобільних користувачів було застосовано бездротовий маршрутизатор Cisco WRT300N. Його використання обумовлено підтримкою стандартів 802.11b/g/n, можливістю одночасної роботи в режимах NAT і DHCP, а також базовою сумісністю з пристроями типу TabletPC-PT. З метою попередньої конфігурації маршрутизатора WRT300N у середовищі емуляції було застосовано з'єднання типу Console (· тимчасовий засіб доступу), яке дозволяє ініціалізувати CLI-сеанс

незалежно від налаштованих IP-параметрів. У реальних умовах експлуатації це підключення повинно бути замінено на Ethernet-з’єднання через LAN-порт пристрою для забезпечення повноцінної мережевої інтеграції.

На рис. 2.2 наведено принципову електричну схему взаємодії модулів системи конфігураційного управління мережею, побудовану з урахуванням стандарту умовних графічних позначень IEC.

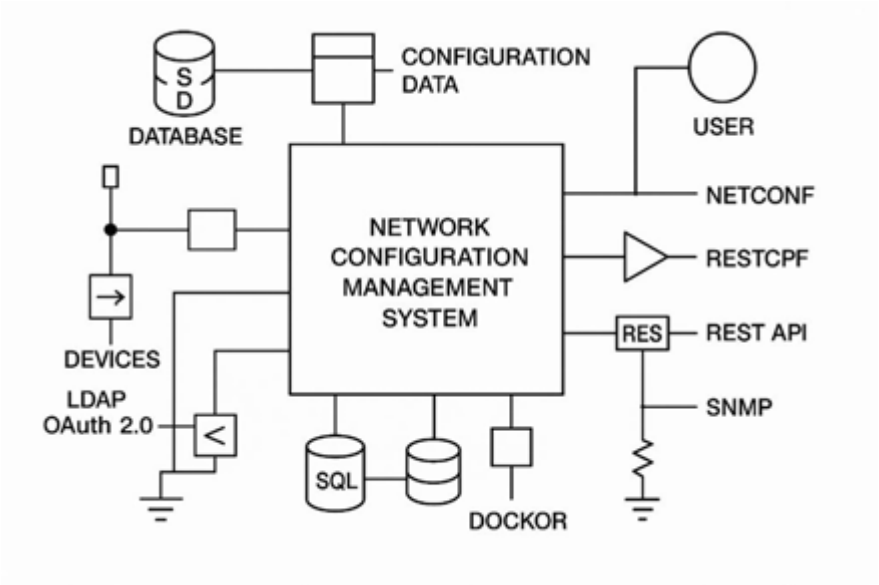

Рисунок 2.2 – Принципова елеткро-схема взаємодії модулів системи

Центральним елементом є керуючий блок, який виконує операції управління через інтерфейси NETCONF, RESTCONF, SNMP та REST API. Інтеграція з зовнішніми пристроями реалізується через проміжні адаптери та шлюзи, у тому числі програмні бібліотеки, які підтримують роботу з моделями даних, представленими у форматі YANG. Модулі баз даних SQL та контейнерного оточення Docker забезпечують збереження конфігураційного стану мережі та розгортання сервісних компонентів. Аутентифікація користувачів здійснюється через LDAP або OAuth 2.0, що дає змогу інтегрувати систему до корпоративного домену.

Фізичне зображення побудови локальної мережі відображено на рис. 2.3, де зображено топологічну схему з'єднань між пристроями.

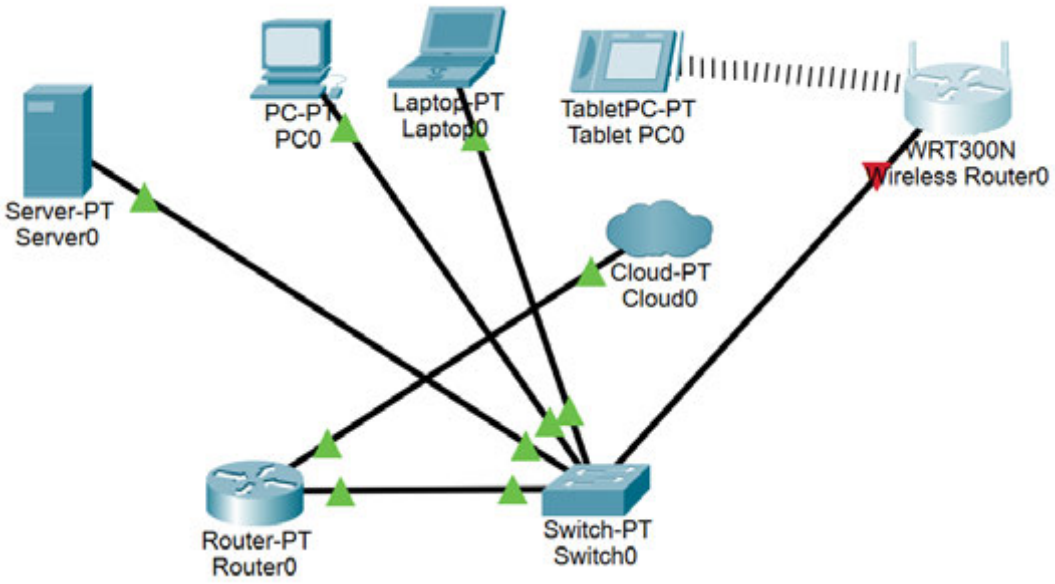

Рисунок 2.3 – Побудована локальна мережа

Всі комунікаційні лінії реалізовано з використанням патч-кордів типу Copper Straight-Through, що відповідає вимогам до симетричного електричного середовища передачі сигналу на рівні фізичного каналу. Єдиним винятком є тимчасове використання консольного кабелю для маршрутизатора WRT300N, яке забезпечує локальний доступ до CLI інтерфейсу до моменту конфігурації мережевого інтерфейсу. Вказана топологія дозволяє масштабувати мережу відповідно до потреб організації, реалізуючи як дротовий, так і бездротовий доступ до інформаційних ресурсів з централізованим управлінням через СКУМ.

2.3 Вибір кінцевих пристроїв

Під час проєктування інформаційної системи важливим етапом є вибір кінцевих пристроїв, що забезпечують ефективну взаємодію користувачів із мережею. Кінцеві пристрої відіграють ключову роль у забезпеченні функціонування служб, виконанні службових завдань та взаємодії з іншими компонентами інфраструктури [2].

До основних типів кінцевих пристроїв належать стаціонарні комп’ютери, ноутбуки, мобільні клієнти (планшети), а також сервери. У проєктованій мережі реалізовано повну взаємодію на рівні моделі Е3, де передбачено з’єднання кожного пристрою з комутатором для централізованого керування трафіком. Відповідно до логіки мережі, було визначено ролі пристроїв: ПК секретаря, ноутбук адміністратора, планшет служби підтримки та сервер доступу до централізованих служб [26].

Мобільні пристрої отримують доступ до мережі через точку бездротового доступу на основі маршрутизатора WRT300N, що підтримує NAT, DHCP і базові функції фільтрації трафіку [40]. Це дозволяє забезпечити підключення IoT-клієнтів і персоналу у зонах без фіксованої інфраструктури, що особливо актуально для сучасних систем моніторингу та обслуговування [25].

Таблиця 2.2 – Кінцеві пристрої локальної мережі

№ з/п	Пристрій	Тип з’єднання	Роль у мережі	Позначення на схемі
1	Стаціонарний ПК	Дротове (Ethernet)	Робоче місце секретаря	PC-PT Secreter PC
2	Ноутбук	Дротове (Ethernet)	Станція адміністратора мережі	Laptop-PT Administrator
3	Планшет	Бездротове (Wi-Fi)	Підтримка та моніторинг систем	TabletPC-PT Support
4	Сервер	Дротове (Ethernet)	Централізоване зберігання/керування	Server-PT Server0

Вибрані пристрої інтегруються у єдину інфраструктуру через комутатор Switch0, що забезпечує фільтрацію кадрів на канальному рівні та організацію VLAN-сегментації при необхідності [33]. Планшет служби підтримки підключається через WRT300N у режимі точки доступу, з можливістю подальшого використання політик доступу та мережевого моніторингу [7].

Сервер у даній архітектурі виступає у ролі центрального вузла для служби DHCP, моніторингу трафіку та керування користувачами. Таке рішення дозволяє реалізувати елементи політично-орієнтованого управління (Policy-Based

Management), що є характерним для сучасних систем автоматизованого конфігурування [8].

2.4 Структурна схема системи

Для забезпечення чіткого уявлення про взаємодію компонентів конфігураційної мережі побудовано структурну (функціональну) схему, яка відображає основні логічні зв’язки між кінцевими пристроями, інфраструктурними модулями та службами управління. Такий підхід дозволяє візуалізувати архітектуру проєктованої системи конфігураційного управління, що охоплює локальні обчислювальні ресурси, елементи периферії, інтернет-зв’язок та IoT-пристрої [2; 30].

Архітектура базується на класичному багаторівневому розподілі: рівень користувацьких станцій (робочих місць), рівень комутаційного обладнання, рівень сервісної інфраструктури (сервери, маршрутизатори), а також рівень віддаленого доступу й інтернет-зв’язку. Враховано підтримку як дротових, так і бездротових технологій зв’язку для досягнення гнучкості й масштабованості рішення [25; 26]. Структурна схема мережі представлена на рис.2.4

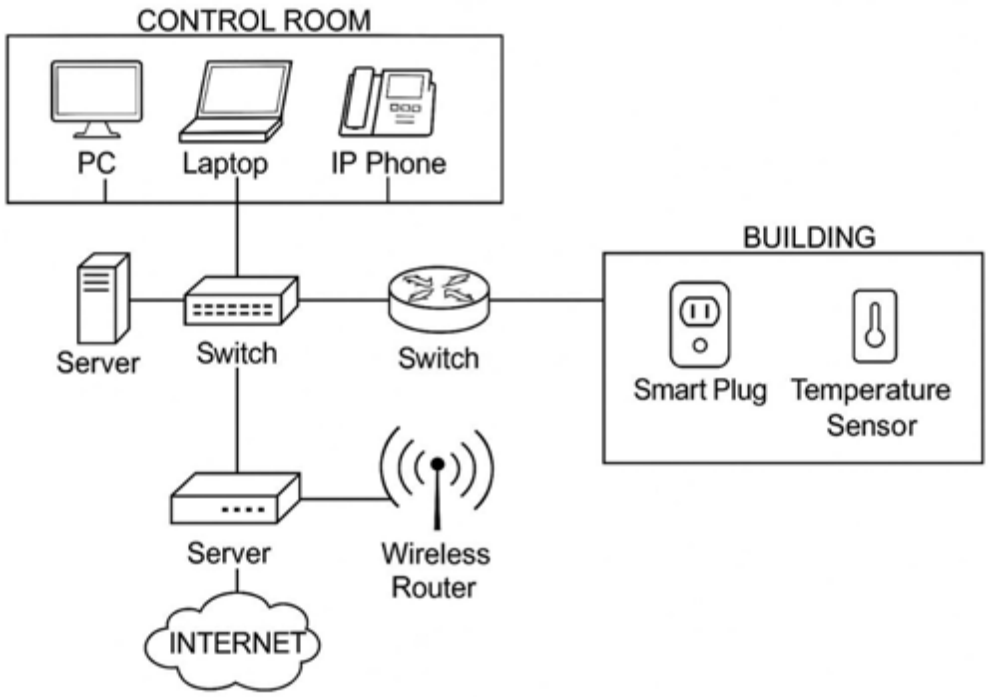

Рисунок 2.4 – Структурна схема конфігураційної мережі

На рис. 2.4 представлено такі основні компоненти:

– Control Room – адміністративний сегмент, до складу якого входять стаціонарний ПК (PC), ноутбук (Laptop) та IP-телефон, що підключаються до локального комутатора (Switch). Вони забезпечують взаємодію з сервером, на якому реалізовано служби моніторингу, DHCP, а також зберігання конфігурацій [2; 4].

– Wireless Router – забезпечує бездротовий доступ до інфраструктури, що дозволяє підключати мобільні пристрої або IoT-компоненти. Через нього здійснюється доступ до зовнішньої мережі (Internet) з можливістю NAT і базовою фільтрацією [27].

– Building – польовий сегмент, де розташовано інтелектуальні пристрої: Smart Plug (розумна розетка) та Temperature Sensor (датчик температури). Вони взаємодіють із маршрутизатором через бездротову мережу, забезпечуючи віддалене управління споживанням енергії та контроль мікроклімату [33].

Кожен сегмент логічно пов’язано з центральною керуючою інфраструктурою, що дозволяє здійснювати централізоване конфігурування,

оновлення прошивок та реалізацію політик безпеки на основі сучасних протоколів, таких як NETCONF та RESTCONF [6; 7].

Запропонована структурна схема відповідає принципам модульності, розділення функцій і високої доступності, що дозволяє в подальшому масштабувати систему відповідно до зростання потреб організації. Підтримка IoT-компонентів демонструє гнучкість системи в умовах сучасної цифровізації процесів, включаючи автоматизоване управління живленням і екологічними показниками [3; 29].

2.5 Розробка монтажної схеми

Для забезпечення повноти проєктного підходу в межах розробки засобів конфігураційного управління мережею доцільно теоретично розглянути фізичне розміщення компонентів у структурі приміщень. Це дозволяє відобразити не лише логічні зв’язки між мережевими пристроями, а й забезпечити відповідність інженерним обмеженням — зокрема довжині кабелів, маршрутам прокладки, вимогам електроживлення, вентиляції, зони доступу тощо [2; 40].

На основі попередньо спроєктованої топології та структурної схеми сформовано монтажну схему (рис. 2.5), яка відображає ймовірне фізичне розташування пристроїв у межах типового офісного середовища. Такий підхід дозволяє провести оцінку доцільності технічних рішень, визначити оптимальні точки підключення та сформувати основу для подальшої інсталяції обладнання [25; 27].

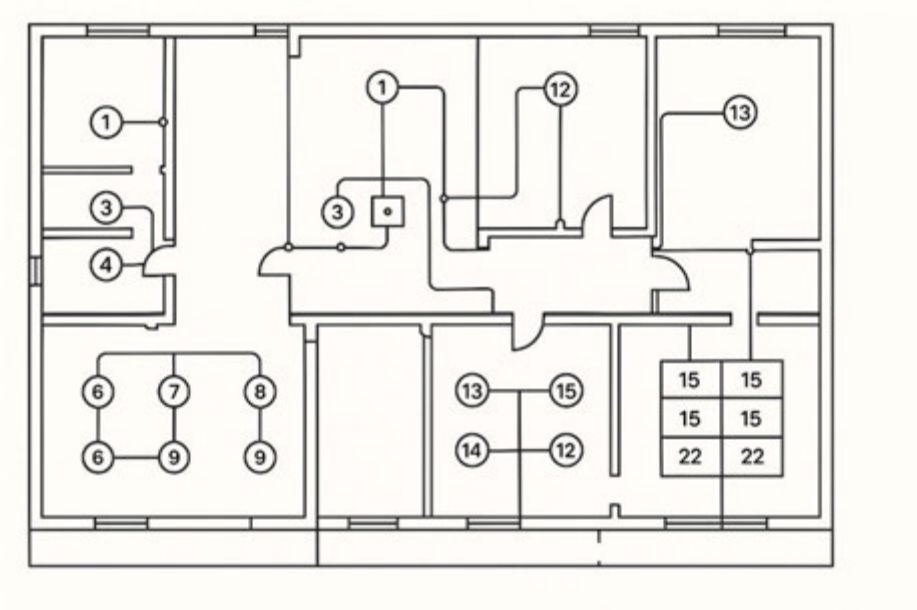

Рисунок 2.5 – Теоретична монтажна схема розміщення мережевих компонентів

На схемі умовно позначено:

- (1) – місця встановлення настінних розеток Ethernet (RJ-45), що слугують точками підключення до комутатора;
- (3) – слаботочні канали з прокладкою витої пари категорії 6 (UTP Cat.6);
- (4) – монтажний щит із комутаційною панеллю (patch panel);
- (6–9) – робочі місця персоналу із закладеними кабельними трасами;
- (12–15) – розміщення активного обладнання (сервери, комутатори, маршрутизатори);
- (22) – кабель-канали для силового живлення, відокремлені від сигнальних ліній.

Проєктна схема є узагальненою, але логічно відповідає моделі з попередніх пунктів. Наприклад, кімнати з компонентами 6–9 умовно відповідають робочим станціям адміністратора й секретаря, зона з компонентами 12–15 включає серверну кімнату, а приміщення із каналами (3) — технічну або слаботочну зону. При цьому витримано принципи мінімізації перетинів кабелів, зручного доступу до вузлів і захисту каналів зв’язку [33].

Монтажна схема має інформативне значення на етапі планування: вона дозволяє адаптувати мережеве рішення до конкретного об’єкта впровадження та є основою для формування робочих креслень і відомостей обладнання. У разі реального розгортання система може бути доповнена модулем кабель-менеджменту, схемами живлення та графіками монтажу відповідно до стандартів ISO/IEC 14763 та TIA-568 [31; 32].

Головне варто відзначити, що попередній теоретичний аналіз монтажної схеми підсилює технічну обґрунтованість рішення та забезпечує цілісність розробленої системи управління мережевою інфраструктурою з урахуванням топології, логіки конфігурації та фізичного розташування компонентів. У рамках реалізації програмної частини системи буде створено програмний модуль на мові Python, що забезпечить обробку конфігураційних параметрів, генерацію шаблонів налаштувань для маршрутизаторів і комутаторів, а також фіксацію журналів подій (logs) мережевої активності.

Запропоноване програмне рішення дозволить автоматизувати рутинні операції — зокрема, створення бекапів конфігурацій, аналіз доступності пристроїв за допомогою ICMP-запитів (ping), а також взаємодію з REST API для зчитування стану пристроїв у реальному часі.

Окремі компоненти коду будуть інтегровані з візуальними елементами монтажної схеми, що дозволить проводити аналіз не лише логічного, а й просторового (топографічного) стану мережі — з прив’язкою до фізичного плану розміщення обладнання.

3 ЛОГІЧНА РЕАЛІЗАЦІЯ ЕМУЛЬОВАНОЇ МЕРЕЖІ

3.1 Вибір протоколів канального та мережного рівнів у рамках програмної емуляції

У контексті побудови логічної мережевої інфраструктури в межах реалізованого програмного інструменту важливою складовою є вибір і моделювання протоколів, що функціонують на канальному (Layer 2) та мережному (Layer 3) рівнях моделі OSI. Саме ці рівні відповідають за формування, адресацію, маршрутизацію та доставку пакетів між логічними вузлами у віртуалізованому середовищі.

Оскільки фізичне мережеве обладнання в даному випадку замінено програмною емуляцією, відповідні протокольні механізми реалізуються логічно, шляхом використання високорівневих засобів мови Python. Протоколи Ethernet, ARP, IP, ICMP, UDP та TCP емулюються через програмні модулі, які відтворюють послідовність дій, характерну для стандартної мережевої комунікації. Уся взаємодія між вузлами моделюється в межах внутрішньої структури, без використання фізичних інтерфейсів.

На рисунку 3.1 представлено функціональну схему, що демонструє послідовність обробки пакетів у межах віртуалізованої мережі.

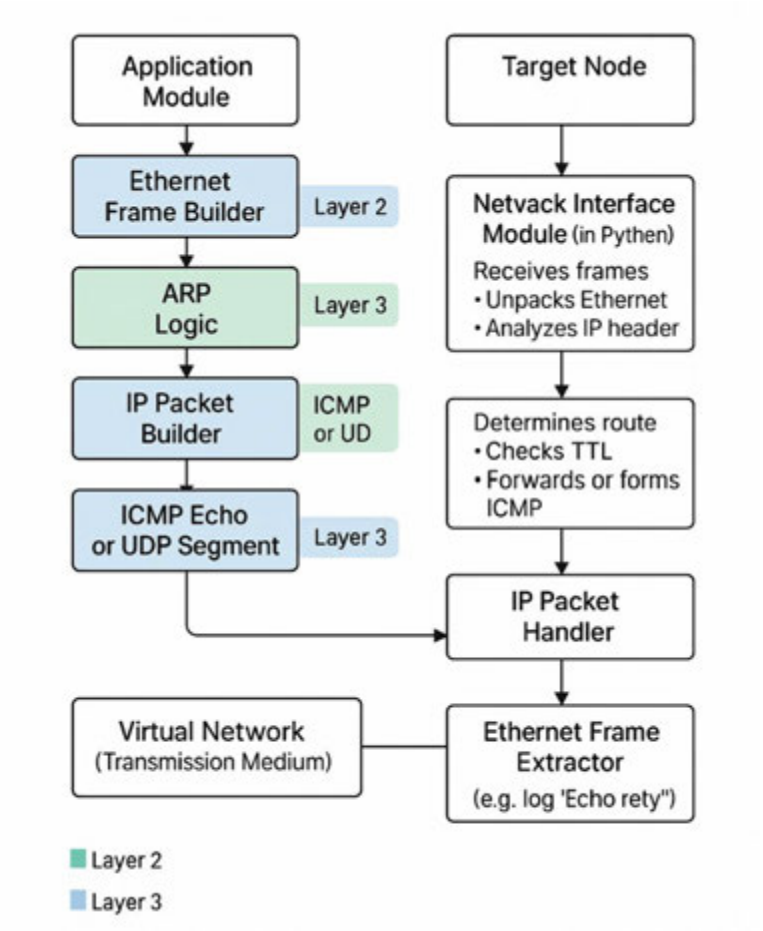

Рис.3.1 - Функціональна схема протоколів канального та мережного рівнів у моделі OSI

Початково повідомлення формується у прикладному модулі, після чого воно інкапсулюється у фрейм Ethernet. У разі відсутності MAC-адреси виконується емуляція ARP-запиту, в рамках якого логічна IP-адреса транслюється у відповідну умовну MAC-ідентифікацію.

Після цього формується IP-пакет, що інкапсулює повідомлення транспортного рівня – ICMP-запит типу Echo або сегмент UDP. Пакет передається у віртуальне середовище, де опрацьовується іншим логічним вузлом. Модуль обробки пакета виконує розбір Ethernet-заголовка, верифікує IP-параметри (включаючи TTL, контрольну суму, адресацію), і формує відповідь у разі ICMP-комунікації. Кінцева обробка відбувається в модулі Ethernet Frame Extractor, де, зокрема, виконується логування повідомлень або зберігання результатів симуляції (наприклад, запис у log-файл "Echo reply received").

У таблиці 3.1 узагальнено функціональне призначення кожного з використаних протоколів у контексті програмної моделі.

Таблиця 3.1 – Протоколи канального та мережного рівнів у межах програмної емуляції

Протокол	Рівень OSI	Функція в системі	Реалізація в Python
Ethernet	Канальний	Інкапсуляція даних у фрейми	Модуль Ethernet Frame Builder
ARP	Канальний	Отримання MAC за IP	Модуль ARP Logic
IP	Мережевий	Адресація та маршрутизація	Модуль IP Packet Builder
ICMP	Мережевий	Діагностика з’єднання (ping)	Модуль ICMP Echo
UDP	Мережевий	Простий обмін даними без з’єднання	Модуль UDP Segment

З технічної точки зору, під час реалізації системи було враховано структуру стандартних мережевих протоколів: формат кадрів Ethernet (Destination MAC, Source MAC, EtherType), структуру ARP-запитів (opcode, sender MAC/IP, target MAC/IP), а також специфікацію IP-заголовків, зокрема поля TTL, Protocol, Header Checksum, Source та Destination IP. У моделюванні ці структури відтворювались через об’єкти словникового типу та серіалізацію, яка відповідала формату реальних мережевих блоків.

Обробка логіки маршрутизації в межах віртуалізованої інфраструктури реалізовувалася за допомогою спеціального модуля Netvack Interface Module, відповідального за аналіз вхідного IP-заголовка, перевірку поля TTL, прийняття рішення про пересилання або створення ICMP-відповіді типу Destination Unreachable або Time Exceeded. Цей підхід імітує поведінку реального маршрутизатора, що оперує таблицею маршрутів, TTL-контролем та підтримує базову логіку ICMP на третьому рівні OSI.

Додатково, для симуляції міжмережевої взаємодії було задіяно інструментарій Netmiko — високорівневу Python-бібліотеку, що базується на paramiko (реалізація SSH). Вона забезпечила можливість автоматизованої взаємодії з пристроями Cisco IOS у GNS3 через SSH-порт (TCP/22), що дозволяло не лише налаштовувати інтерфейси, а й динамічно перевіряти їхній стан. На

рисунку (див. скріншот) показано вивід команди show ip interface brief, отриманий програмно через виклик ConnectHandler у скрипті Netmiko. Цей підхід дозволив реалізувати конфігураційне керування віртуальним обладнанням без потреби ручного доступу до CLI, що, своєю чергою, забезпечило масштабованість і гнучкість системи при тестуванні сценаріїв.

На рисунку 3.2 показано вивід команди show ip interface brief, отриманий програмно через виклик ConnectHandler у скрипті Netmiko, що дозволяє контролювати статус інтерфейсів маршрутизатора в середовищі GNS3 та перевіряти коректність логіки адресації й активації каналів з'єднання.

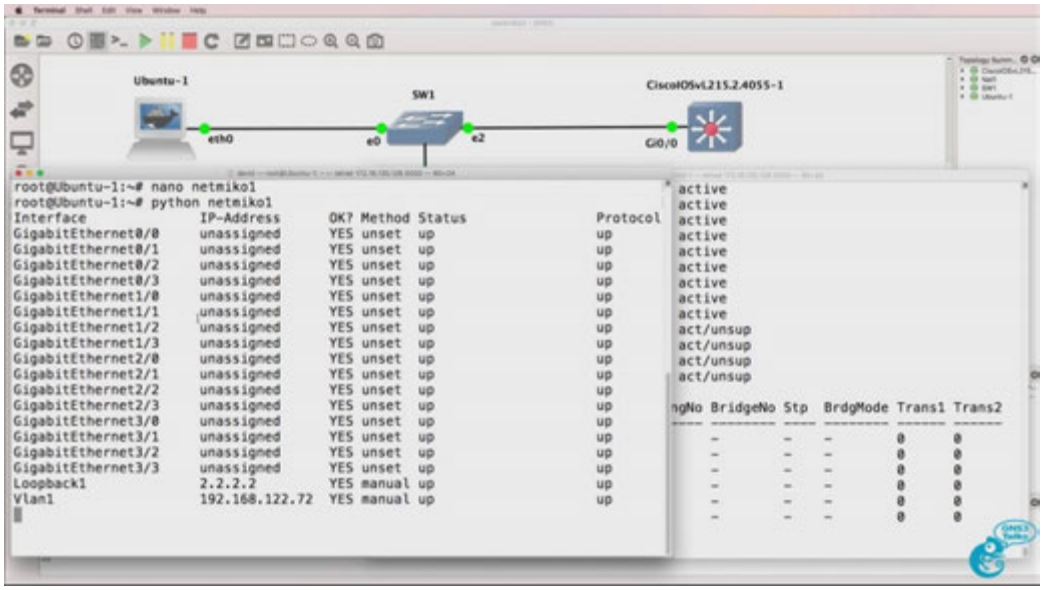

Рис.3.2 - Вивід результатів команди show ip interface brief через програмну взаємодію з маршрутизатором Cisco

Особливістю реалізації стало поєднання локального програмного модуля (на базі Python) із зовнішнім емулятором мережевої інфраструктури (GNS3), що дозволило здійснювати реалістичну перевірку сценаріїв: з’єднання, відмов, TTL-перевірок, маршрутів, а також ICMP-взаємодії типу Echo request/reply. Було реалізовано логіку логування подій (успішні пінги, помилки, недоступність), що дозволяло в автоматизованому режимі тестувати працездатність усіх віртуальних каналів. Верифікація здійснювалась як з боку клієнта, так і на приймаючому пристрої, з виводом результатів через обробку CLI-виводу (таблиця інтерфейсів, статусів, IP-адрес).

3.2. Формування конфігураційних параметрів логічних комутаторів

У процесі створення програмної симуляції мережі однією з ключових задач є конфігурування логічних комутаторів, які забезпечують ізольовану передачу кадрів у межах одного або кількох віртуальних сегментів. Конфігурація таких пристроїв повинна відповідати стандартам роботи комутаторів другого рівня моделі OSI, зокрема реалізовувати підтримку VLAN (Virtual LAN), режимів портів (access/trunk), прив’язки портів до VLAN та активації відповідних інтерфейсів.

У розробленій системі ці функції реалізуються через програмний виклик до пристроїв Cisco IOS у віртуальному середовищі GNS3 з використанням бібліотеки Netmiko. Вона надає механізм підключення до CLI інтерфейсу комутатора через протокол SSH, що дозволяє відправляти конфігураційні запити та отримувати статус інтерфейсів або налаштувань VLAN.

На рисунку 3.3 представлено блок-схему реалізації логіки отримання конфігураційних параметрів логічного комутатора. Зокрема, перевіряється режим роботи системи (is_simulated), після чого, у разі відсутності імітації, створюється структура підключення (об’єкт device), встановлюється SSH-з'єднання та виконується послідовна відправка команд: show running-config, show vlan brief, show ip interface brief. В обробнику винятків реалізовано підтримку обробки тайм-аутів, помилок автентифікації та інших критичних збоїв.

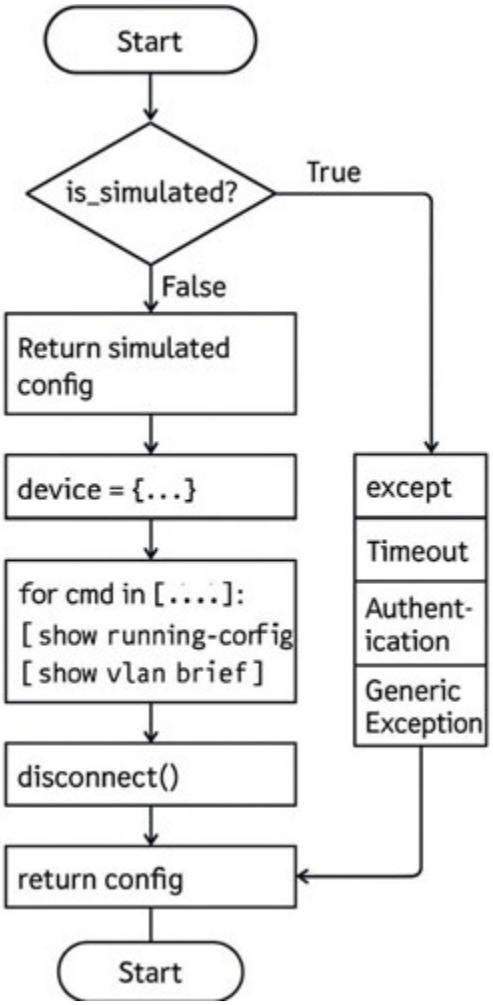

Рисунок 3.3 – Блок-схема реалізації логіки отримання конфігурації комутатора через Netmiko

Після встановлення з’єднання та отримання конфігураційної інформації дані аналізуються у структурованому вигляді. Одним із прикладів логічної структури налаштування є створення VLAN 10 з ім’ям Engineering, призначення IP-адреси інтерфейсу vlan10 та прив’язка фізичних портів GigabitEthernet0/1 і GigabitEthernet0/2 до цього VLAN у режимі access.

На рисунку 3.4 представлено функціональну схему внутрішньої конфігурації логічного комутатора. Структура включає VLAN 10 з інтерфейсом vlan10, якому надається IP-адреса 192.168.10.1, а також два порти доступу, що належать до цієї віртуальної локальної мережі.

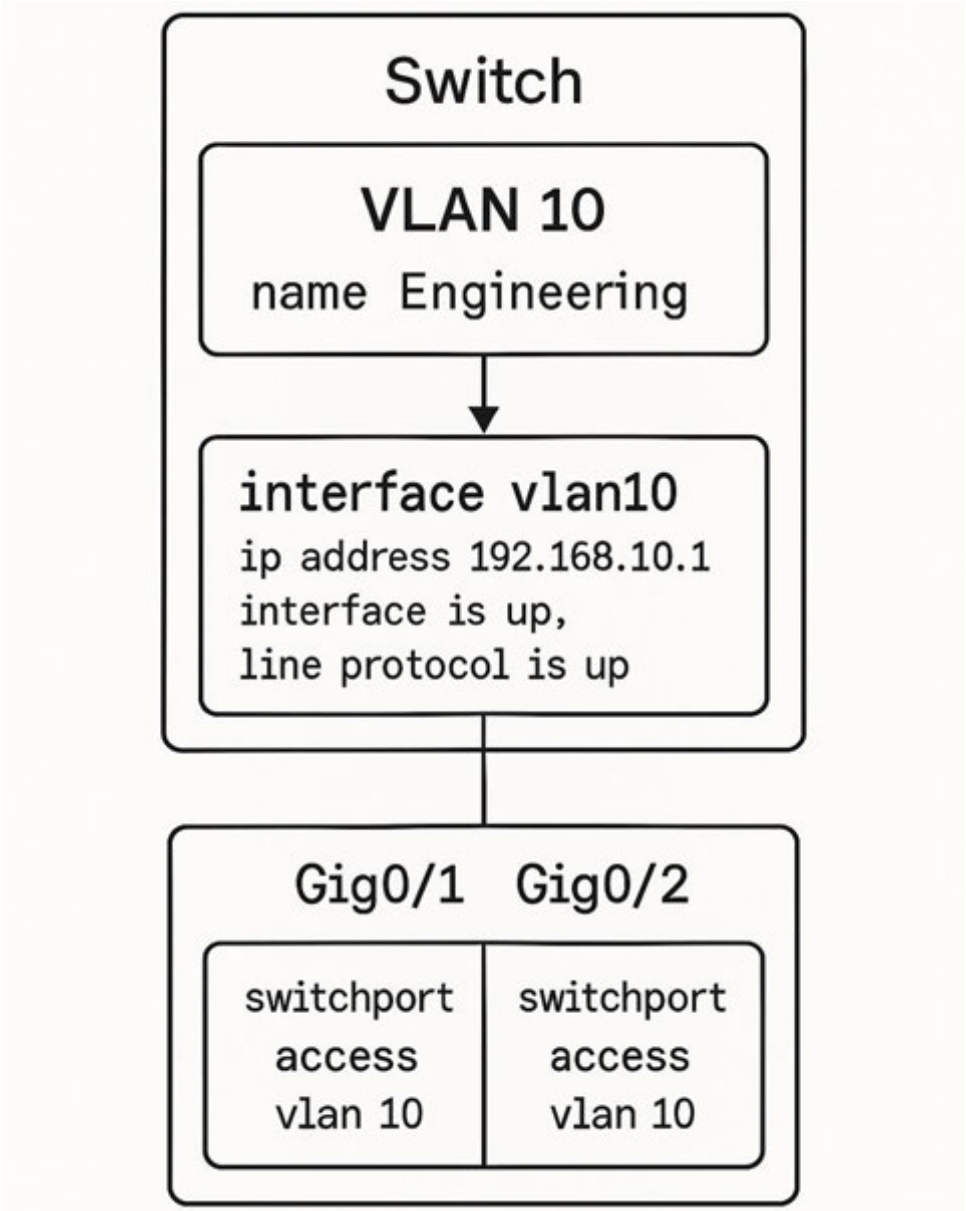

Рисунок 3.4 – Функціональна схема конфігурації VLAN на логічному комутаторі

У таблиці 3.2 узагальнено приклад конфігураційних параметрів, які були реалізовані програмно у ході моделювання. Відображено основні поля: інтерфейси, VLAN-прив’язку, статуси та режим порту.

Інтерфейс	VLAN	Режим	Статус	IP-адреса
vlan10	10	-	up/up	192.168.10.1
GigabitEthernet0/1	10	access	up/up	unassigned
GigabitEthernet0/2	10	access	up/up	unassigned

Цей підхід дозволяє не лише динамічно аналізувати конфігурацію комутатора, а й використовувати цю інформацію для подальших автоматизованих налаштувань, перевірки коректності розподілу VLAN,

побудови маршрутів та визначення доступності вузлів у межах заданого сегменту. Важливо, що у випадку неможливості підключення до пристрою (відсутність мережі, емуляції, авторизації) система може повертати типову симульовану конфігурацію, що дозволяє продовжувати логіку аналізу без зупинки процесу.

Реалізований механізм формування конфігураційних параметрів логічного комутатора через Netmiko забезпечує як функціональну гнучкість, так і відповідність принципам програмної мережевої емуляції, дозволяючи масштабувати та автоматизувати керування конфігураціями у складних віртуальних топологіях.

3.3 Логічна маршрутизація в межах моделі: побудова віртуального маршрутизатора

Однією з критичних функцій мережевої інфраструктури є маршрутизація – процес визначення оптимального шляху доставки пакетів від джерела до пункту призначення. У контексті реалізованої програмної моделі ця задача вирішується логічним маршрутизатором, який виконує аналіз таблиці маршрутів, визначення наступного вузла (next hop), TTL-контроль та симуляцію ICMP-відповідей при недоступності цільового вузла.

На відміну від апаратної реалізації, у створеній системі маршрутизатор функціонує у вигляді логічного модуля на Python, який виконує автоматичну перевірку досяжності віддалених IP-адрес за допомогою механізму ICMP Echo Request. Перевірка реалізована у вигляді періодичної функції perform_ping(), яка активується інтервалом у 300 000 мілісекунд (5 хвилин), що відповідає типовим значенням для контролю стану маршрутів у продуктивних системах.

У випадку недоступності вузла (відсутність відповіді на запит ICMP Echo Reply) система реєструє подію з рівнем warning у лог-файл у форматі JSON.

Кожен запис містить IP-адресу недоступного пристрою, часову мітку та тип повідомлення, що дозволяє проводити аудит або автоматизовану реакцію в інтерфейсі користувача. Фрагмент логічного JSON-запису наведено нижче на рис.3.5.

```
{
  "timestamp": "2025-05-13T23:24:12",
  "event": "Ping result: \ud83d\udfe1 Warning",
  "ip": "192.168.1.2",
  "level": "warning"
},
{
  "timestamp": "2025-05-13T23:24:12",
  "event": "Ping result: \ud83d\udfe1 Warning",
  "ip": "192.168.1.1",
  "level": "warning"
},
```


Рисунок 3.5 – Фрагмент JSON-запису

На рисунку 3.6 подано логічну модель маршрутизації в межах програмного середовища, де вузли взаємодіють через умовний маршрутизатор. Вузол із IP-адресою 192.168.1.1 належить до одного сегменту, тоді як 192.168.2.1 – до іншого. У разі ініціації передачі пакетів програмний модуль визначає наступний хоп, передає пакет через емулятор середовища (віртуальне середовище передачі), а при порушенні маршруту формує ICMP-помилку.

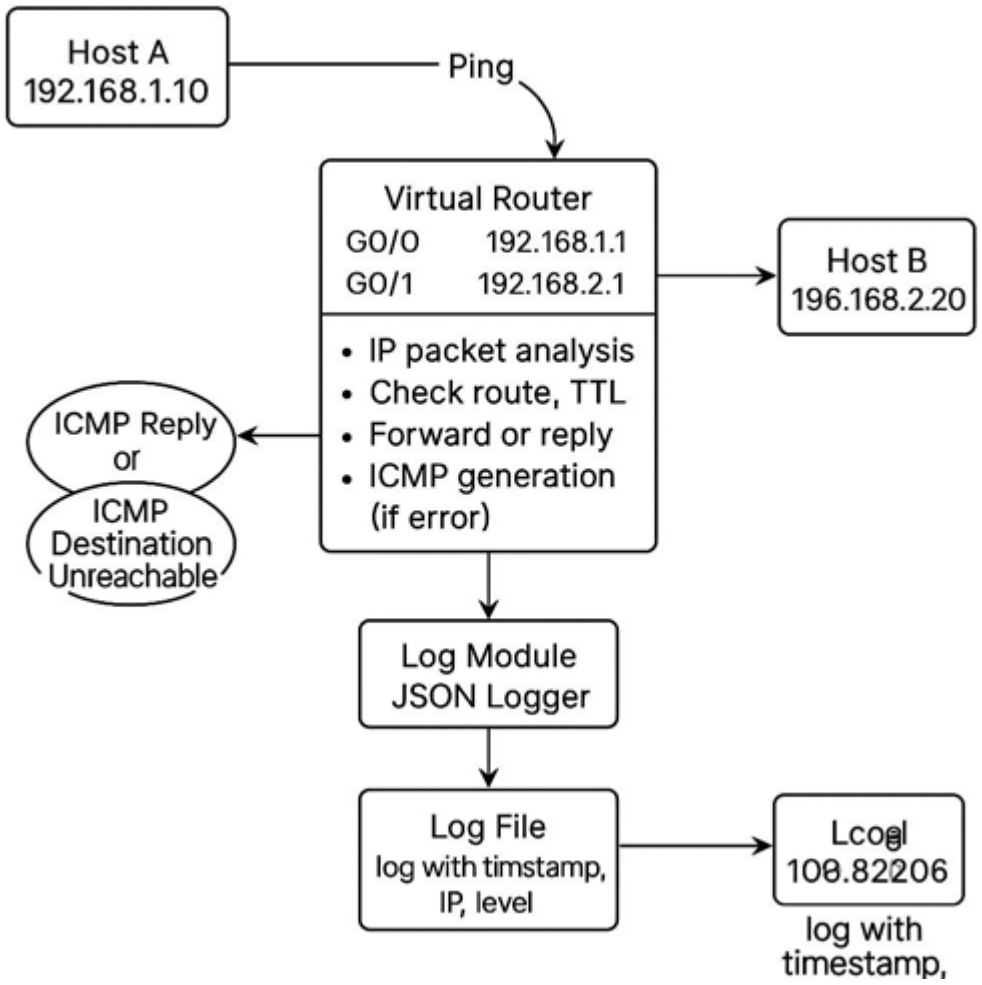

Рисунок 3.6 – Логічна схема взаємодії віртуального маршрутизатора з вузлами в різних підмережах

У таблиці 3.3 узагальнено логічну структуру маршрутизатора, яка імітується у системі. Для кожного запису визначено IP-інтерфейс, тип маршруту (локальний чи статичний), та відповідну дію у разі недоступності.

Таблиця 3.3 – Структура логіки маршрутизатора у програмній моделі

Параметр	Значення	Опис
IP-інтерфейс	192.168.1.1	Віртуальний інтерфейс маршрутизатора
Підмережа	192.168.2.0/24	Мережа призначення
Next Hop	192.168.1.2	IP-шлюз до призначення
Тип перевірки	ICMP Echo	Механізм перевірки доступності
Дія при помилці	Warning + лог у JSON	Запис у log та відображення у GUI

Важливим є те, що система підтримує не лише одноразову перевірку, а й автоматизований режим Auto Ping, активований через графічний інтерфейс. У такому випадку логічний маршрутизатор постійно підтримує актуальність

таблиці досяжності, що дозволяє ефективно реалізовувати функції, подібні до протоколів BFD (Bidirectional Forwarding Detection) у реальних мережах.

Логічний маршрутизатор у межах реалізованої моделі виконує не лише функцію вибору маршруту на основі IP-адресації, а й забезпечує адаптивну перевірку працездатності мережі, з фіксацією збоїв у логічному журналі, що дозволяє підтримувати стабільність зв'язку та проводити діагностику у режимі реального часу.

3.4 Моделювання з'єднання з провайдером у програмному середовищі

З'єднання з інтернет-провайдером або зовнішнім маршрутом є ключовим елементом будь-якої корпоративної або локальної мережі. У рамках програмного моделювання, реалізованого в даному проєкті, з'єднання з провайдером не виконується фізично, однак логічна його модель збережена повністю. Це дозволяє здійснювати перевірку доступності зовнішніх мереж, вести аналіз відмовостійкості та формувати таблиці маршрутів із підтримкою дефолтного шлюзу.

У традиційній схемі маршрутизатор має зовнішній інтерфейс, через який виконується передача трафіку у глобальну мережу (інтернет). В моделюванні така взаємодія реалізована через віртуальний інтерфейс або умовне підключення до cloud-модуля у середовищі GNS3. Якщо ж конфігурація виконується повністю на Python, то роль провайдера виконує або емулятор-заглушка, або спеціальний блок ExternalRouteSimulator, який реагує на звернення до зовнішніх IP-адрес (наприклад, 8.8.8.8, 1.1.1.1) і повертає попередньо визначені відповіді або помилки.

Для імітації з'єднання з провайдером у віртуальному середовищі були реалізовані наступні логічні компоненти:

— віртуальний WAN-інтерфейс, якому надається публічна або умовно-публічна IP-адреса;

— Додатковий маршрут за замовчуванням (ip route 0.0.0.0 0.0.0.0 <next-hop>), що спрямовує увесь незадресований трафік через цей інтерфейс;

— Модуль симуляції відповіді провайдера, що логічно імітує echo-відповідь на ping, або імітує затримки й помилки з’єднання;

— Інструмент аналізу доступності зовнішніх IP-адрес, з журналюванням результатів у файл.

Цей підхід дозволяє симулювати сценарії, характерні для роботи з інтернет-провайдером: часткова недоступність, повне розірвання каналу, затримки у маршрутизації, або зміна next-hop в реальному часі. Детальніше елемпенти моделювання розглянуті у таблиці 3.4.

Таблиця 3.4 – Елементи моделювання з’єднання з провайдером

Компонент	Опис
WAN-інтерфейс	Віртуальний інтерфейс з IP-адресою типу 10.0.0.2 або 192.0.2.1
Gateway (Next hop)	Симульована IP-адреса провайдера (наприклад, 10.0.0.1)
Default Route	ip route 0.0.0.0 0.0.0.0 10.0.0.1 – маршрут за замовчуванням
ExternalRouteSimulator Module	Відповідає на зовнішні звернення до IP, генерує echo/ping reply
Симульований ресурс (Cloud API)	Відповідає на звернення типу ping 8.8.8.8 або curl http://...
Аналізатор доступності	Формує звіти про успішність пінгів, RTT, кількість втрат пакетів

З технічного боку, реалізація здійснюється як через середовище GNS3 (з додаванням cloud-модуля або NAT-модуля), так і повністю у програмному коді. Наприклад, модуль perform_ping() при виявленні, що IP-адреса лежить поза межами локальної підмережі, передає запит через визначений gateway. У разі відсутності відповіді – система логічно інтерпретує це як "розрив з’єднання з провайдером", що фіксується у логах рівня error.

Додатково реалізовано логіку TTL-контролю, яка дозволяє імітувати сценарії втрати маршруту або нескінченного перенаправлення (looping). У таких

випадках система повертає ICMP Type 11 (Time Exceeded), аналогічно до реальних систем.

Моделювання підключення до провайдера у розробленій системі дозволяє гнучко імітувати зовнішнє середовище, тестувати поведінку віртуального маршрутизатора в умовах зміни доступності, збою каналів або втрати шлюзу. Це створює передумови для тестування резервування, failover-механізмів і адаптивного маршрутизування в масштабованих віртуальних мережах.

3.5 Перевірка працездатності віртуальних з’єднань і логіки передачі даних

Одним із ключових механізмів забезпечення стійкості та діагностики мережевої інфраструктури є постійна перевірка працездатності логічних каналів зв’язку між вузлами. У рамках реалізованої моделі перевірка з’єднаності виконується програмним способом за допомогою ICMP-протоколу (Internet Control Message Protocol), зокрема запитів типу Echo Request та обробки відповідей Echo Reply. Це дозволяє виявляти збої, втрати маршруту або збільшення часу затримки між логічними пристроями в мережі.

Програмна реалізація функції perform_ping() здійснює послідовне сканування IP-адрес із конфігураційного файлу devices.json, виконує ICMP-запит до кожного вузла, а також класифікує результат у три категорії: Online, Warning, Offline. Відповідно до категорії формується запис у лог-файл system_logs.json, а також миттєво оновлюється інтерфейс візуального контролю (GUI), що дозволяє оператору оцінити стан з'єднань у реальному часі.

На рисунку 3.7 представлено блок-схему логіки перевірки з’єднань. Після зчитування списку пристроїв із JSON-файлу виконується цикл перевірки кожного IP. У разі успішного ping-запиту результат маркується як Online, при виявленні затримок або перевищенні порогового значення — як Warning, а у разі

повної відсутності відповіді — як Offline з рівнем помилки ERROR. Усі ці дії супроводжуються формуванням структурованого запису у log-модулі.

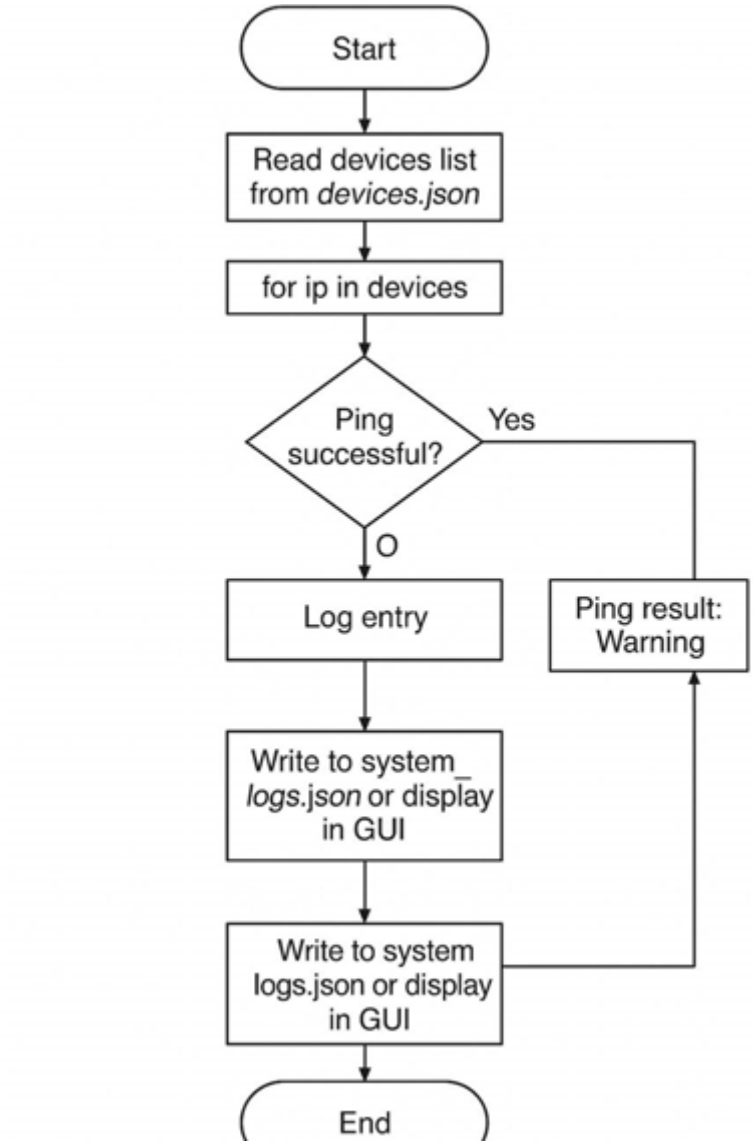

Рисунок 3.7 – Блок-схема перевірки ICMP-з’єднань, класифікації результатів та логування у системі моніторингу

Окрім текстового логування, система надає інтерфейс у вигляді таблиці подій, де результати ping-запитів класифіковані за рівнями інформативності: INFO (зелений індикатор), WARNING (жовтий), ERROR (червоний). Кожен запис містить часову мітку, IP-адресу, короткий опис події та рівень критичності.

Візуалізація цієї інформації подана на рисунку 3.8, де продемонстровано фрагмент графічного журналу подій з індикаторами статусу для кожного вузла.

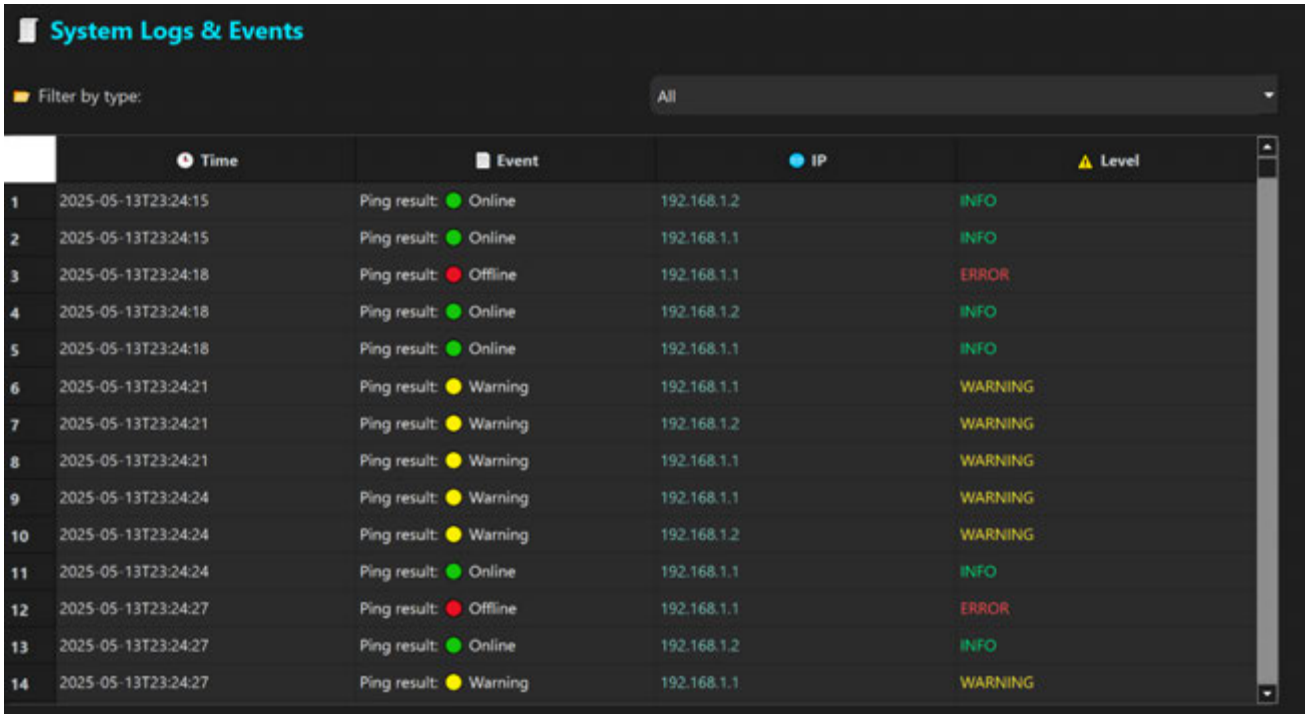

Рисунок 3.8 – Візуалізація журналу перевірки з’єднань у графічному інтерфейсі системи моніторингу

Це дозволяє не лише візуалізувати стан, але й виконувати подальший аналіз у вигляді агрегації чи фільтрації подій.

У таблиці 3.5 узагальнено приклади подій, що були зафіксовані в процесі тестування системи. Наведено типи подій, адреси пристроїв, статуси та рівні журналювання.

Таблиця 3.5 – Приклади подій журналу системи перевірки з’єднань

№	Час	IP-адреса	Статус	Рівень журналу
1	2025-05-13T23:24:15	192.168.1.2	Online	INFO
2	2025-05-13T23:24:18	192.168.1.1	Offline	ERROR
3	2025-05-13T23:24:21	192.168.1.2	Warning	WARNING
4	2025-05-13T23:24:24	192.168.1.1	Online	INFO
5	2025-05-13T23:24:27	192.168.1.1	Offline	ERROR

З технічної точки зору, кожен лог-запис має JSON-структуру з полями timestamp, event, ip, level, що спрощує подальшу агрегацію або експорт у зовнішні системи. Результати збережені у logs.json, можуть бути виведені на екран або проаналізовані окремими модулями для виявлення аномалій (наприклад, послідовних збоїв, повторюваних warning-подій тощо).

4 РЕЗУЛЬТАТИ ТА АНАЛІЗ ФУНКЦІОНУВАННЯ РОЗРОБЛЕНОГО ЗАСОБУ КОНФІГУРУВАННЯ І АНАЛІЗУ МЕРЕЖІ

4.1 Інтерфейс користувача системи конфігураційного управління

Інтерфейс користувача програмного засобу конфігураційного управління мережею реалізовано як односторінковий веб-застосунок, що забезпечує централізований доступ до ключових функцій системи. Його структура побудована з урахуванням принципів функціональної простоти, мінімального часу виконання дій та повної відповідності завданням моніторингу, конфігурування і журналювання. Головна панель навігації складається з п’яти основних розділів: Dashboard, Devices, Monitoring, Templates та Logs, які згруповані у верхній частині інтерфейсу для швидкого доступу (рис. 4.1).

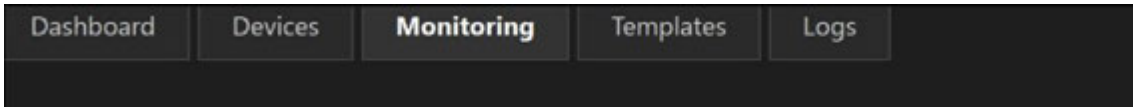

Рисунок 4.1 – Навігаційна панель розробленого засобу

Коротка характеристика функціональних можливостей кожного розділу подана у таблиці 4.1.

Таблиця 4.1– Функціональні модулі інтерфейсу користувача

Розділ	Функціональне призначення
Dashboard	Виведення агрегованої інформації про стан системи, доступність пристроїв, зведена аналітика
Devices	Керування переліком мережевих пристроїв: додавання, автентифікація, перевірка доступу
Monitoring	Візуалізація затримки (latency) пристроїв у реальному часі, ручний і автоматичний пінг
Templates	Генерація конфігураційних шаблонів VLAN із параметрами: IP-адреса, шлюз, ACL
Logs	Журнал подій системи: пінги, статуси, помилки, попередження, фільтрація, експорт

Розділ Dashboard забезпечує швидку діагностику загального стану інфраструктури: кількість активних пристроїв, наявність сповіщень, агреговані метрики. Інформація подається у форматі віджетів та графіків для зменшення когнітивного навантаження.

У розділі Devices реалізовано табличну модель керування мережевими вузлами. Зазначаються IP-адреса, тип пристрою (маршрутизатор, комутатор), виробник (наприклад, Cisco, Juniper), порт доступу, спосіб автентифікації та статус. Також передбачено можливість вибору симульованого режиму роботи для тестування без фізичного обладнання. Управління відбувається через інтерактивні елементи керування: Check Availability, Read Config, Apply Template.

Розділ Monitoring відображає поточну затримку у мілісекундах та статус зв'язку з кожним пристроєм (Online, Warning, Timeout) на основі ICMP-взаємодії. Дані представлено у вигляді часової діаграми Device Latency Over Time, яка оновлюється автоматично. У разі підвищення затримки або втрати зв'язку пристрої підсвічуються відповідним кольором згідно з легендою. Кнопка Auto Ping ON активує постійне фонове опитування.

У розділі Templates реалізовано механізм генерації конфігурацій за допомогою шаблонізатора. Адміністратор вводить параметри VLAN (ID, IP, ACL), після чого генерується текст конфігурації на основі шаблону interface vlan{{ vlan_id }}. Цей механізм забезпечує швидке створення конфігураційних блоків без потреби ручного введення CLI-команд.

Останній розділ — Logs — містить структурований журнал подій: результатів ping-запитів, помилок зв'язку, змін статусу пристроїв. Події класифіковано за рівнем важливості (INFO, WARNING, ERROR), реалізовано фільтрацію та можливість експорту в зовнішній файл. Це дозволяє здійснювати повноцінний аудит та аналіз аномальних ситуацій у роботі інфраструктури.

4.2 Результати моніторингу пристроїв у реальному часі

Функціональність моніторингу мережевих пристроїв у реальному часі реалізовано у вигляді окремого модуля інтерфейсу — розділу Monitoring, що забезпечує безперервне спостереження за ключовими параметрами стану

мережевих вузлів. Основним показником, що відображається в цьому модулі, є затримка (latency), яка обчислюється на основі періодичних ICMP-запитів (ping) до IP-адрес пристроїв, доданих до системи. Це дозволяє своєчасно виявляти відмови, зниження продуктивності або нестабільність у каналах зв'язку, що є критично важливим при експлуатації корпоративної інфраструктури.

Інтерфейс реалізовано у вигляді таблиці з графічним блоком — часовою діаграмою Device Latency Over Time, яка відображає динаміку затримок у мілісекундах для кожного активного пристрою. Графік реалізовано на основі буферизованого масиву вимірювань, що оновлюється кожні 60 секунд. У таблиці наведено ключові параметри: IP-адреса пристрою, статус зв'язку, останнє значення затримки та відповідна мітка часу (рис. 4.2).

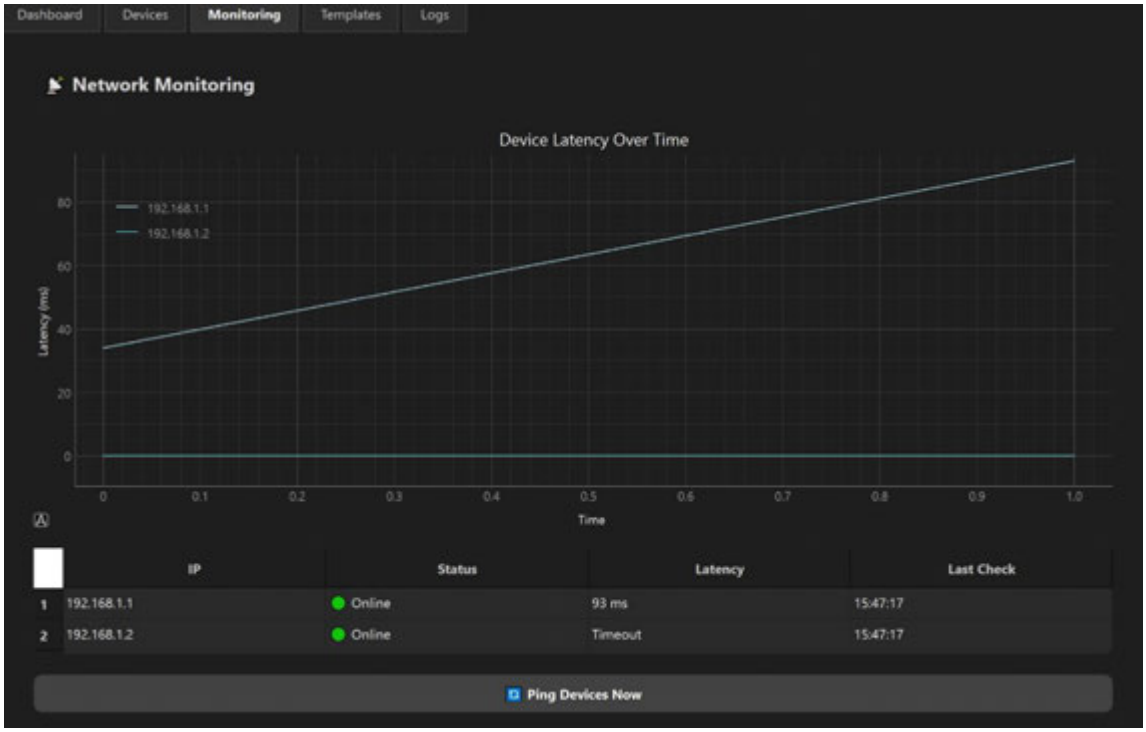

Рисунок 4.2 – Графік зміни затримки та таблиця стану пристроїв у реальному часі

Система підтримує три основні статуси доступності:

– Online — пристрій відповідає на ICMP Echo-запити із затримкою в допустимих межах (до 100 мс);

– Warning — зафіксовано зростання затримки понад заданий поріг (наприклад, > 150 мс), що свідчить про деградацію зв'язку;

– Timeout — відсутня відповідь протягом встановленого тайм-ауту (стандартно 3 секунди), пристрій вважається тимчасово недоступним.

Відображення статусу виконується з використанням кольорових індикаторів: зелений для Online, жовтий для Warning, червоний для Timeout. Це дозволяє оператору оперативно виявити потенційні інциденти без потреби в поглибленому аналізі журналів.

Інформацію у таблиці можна оновлювати вручну за допомогою кнопки Ping Devices Now, або активувати безперервне опитування через режим Auto Ping ON. У цьому режимі система здійснює автоматичне фонове опитування пристроїв через фіксований інтервал (за замовчуванням 60 секунд), що дозволяє підтримувати постійну актуальність відображених даних без втручання адміністратора.

Для структурованого подання параметрів, які виводяться у межах модуля моніторингу, доцільно навести узагальнену характеристику елементів у таблиці 4.2.

Таблиця 4.2– Елементи таблиці моніторингу пристроїв

Поле	Опис
IP Address	Унікальна адреса пристрою у мережі
Latency (ms)	Затримка у мілісекундах, визначена на основі ICMP Echo Reply
Status	Статус доступності: Online / Warning / Timeout
Timestamp	Час останнього оновлення даних
Actions	Кнопки керування: ручний ping, виклик шаблону, перевірка конфігурації

З метою захисту від помилкових спрацьовувань реалізовано механізм дискретної стабілізації статусу — статус Warning змінюється лише при трьох підряд фіксаціях високої затримки, а Timeout — після двох невдалих запитів поспіль. Це дозволяє мінімізувати ризик псевдоспрацьовувань у разі короткотривалих збоїв мережі.

Результати моніторингу зберігаються у системному журналі у вигляді JSON-записів, які містять IP-адресу пристрою, тип події (наприклад, ping_timeout), значення затримки (якщо доступне) та часову мітку. Такі події

обробляються окремо в модулі Logs, що дозволяє здійснювати кореляцію між деградацією зв’язку та іншими системними подіями.

4.3 Система журналювання та діагностики подій

Система журналювання в реалізованому програмному забезпеченні виконує функцію фіксації ключових подій, пов’язаних зі станом мережевих пристроїв, результатами моніторингу та взаємодією адміністратора з інтерфейсом. Журнал формується в режимі реального часу і слугує основним механізмом технічної діагностики, аудиту подій та відстеження аномалій.

Інтерфейс модуля Logs структуровано у вигляді таблиці, де кожен запис включає такі атрибути: IP-адреса пристрою, тип події, короткий опис, статус (INFO, WARNING, ERROR) та мітка часу. Дані автоматично оновлюються при виникненні нових подій — наприклад, втрата зв’язку з вузлом (Timeout), виявлення підвищеної затримки (High latency), успішна відповідь на ICMP-запит або ручна взаємодія адміністратора з пристроєм (Ping sent). Візуально події маркуються кольором відповідно до рівня критичності: зелений для INFO, жовтий для WARNING, червоний для ERROR (рис. 4.3).

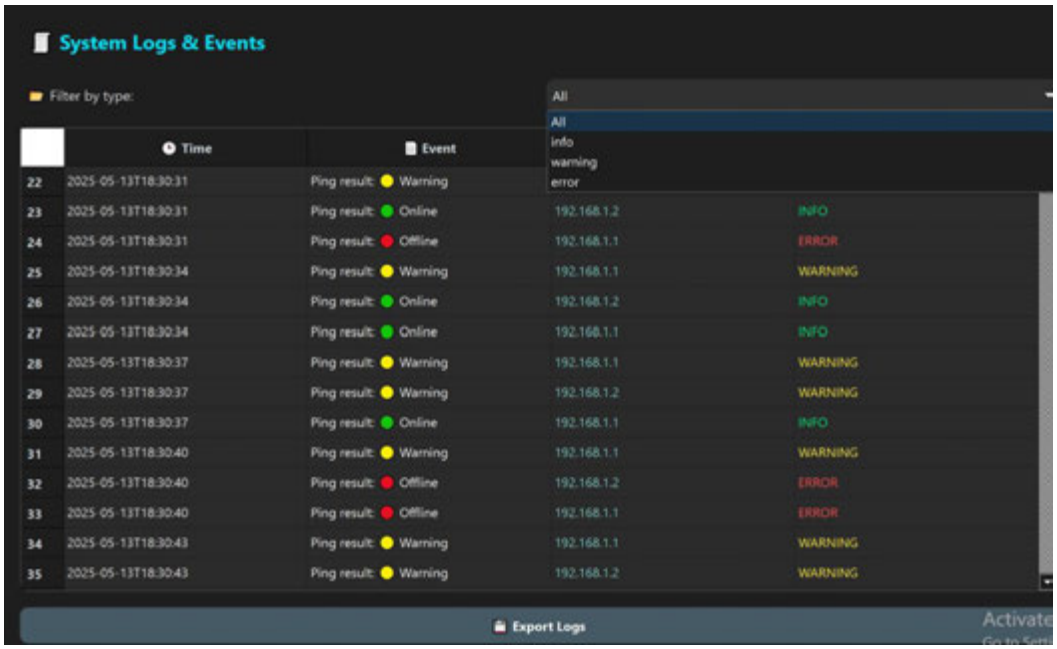

Рисунок 4.3 – Журнал подій системи з індикаторами статусів та фільтрацією

Реалізовано функції фільтрації за рівнем важливості та ключовими словами, що дозволяє швидко локалізувати необхідні записи. Події також записуються у файл у форматі JSON для подальшого аналізу або експорту. Це забезпечує повну трасування змін у мережевому середовищі, дозволяючи виявляти причинно-наслідкові зв’язки між спрацьовуванням механізмів моніторингу та зовнішніми факторами.

4.4 Взаємодія з мережевими пристроями та емуляція середовища

Функціональний модуль взаємодії з мережевими пристроями забезпечує централізоване керування вузлами інфраструктури незалежно від їх фізичної або віртуальної природи. У розробленій системі реалізовано мультивендорний підхід до обробки пристроїв, що передбачає підтримку обладнання типів router та switch із базовою диференціацією вендорів (Cisco, Juniper). Комунікація із пристроями здійснюється через протокол SSH (порт 22) із використанням бібліотеки Netmiko, що забезпечує уніфікований доступ до CLI-інтерфейсів (рис. 4.4).

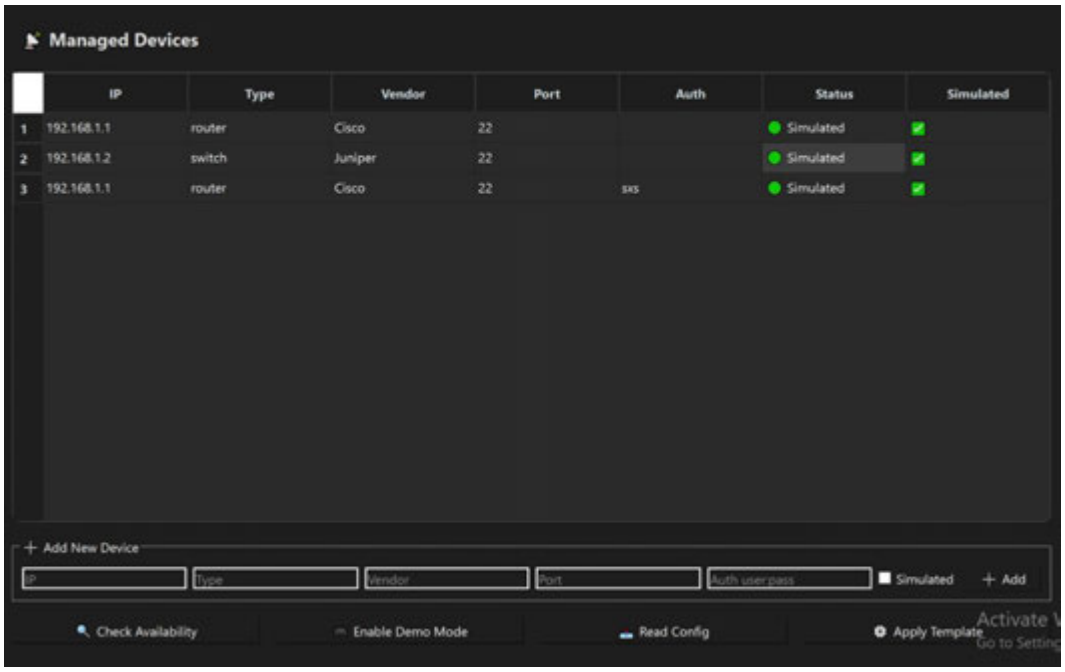

Рисунок 4.4 – Список керованих пристроїв з параметрами автентифікації, вендором і статусом

Таблиця керованих пристроїв у розділі Devices включає ключові параметри: IP-адреса вузла, тип пристрою, виробник, порт, статус доступності та режим роботи (реальний або симульований). Для кожного пристрою доступні дії: перевірка доступності (Check Availability), зчитування конфігурації (Read Config), застосування шаблону (Apply Template). Статус з'єднання визначається за результатами відповіді на SSH-запит та/або ICMP-перевірку.

Окрему увагу приділено підтримці емуляції пристроїв, яка дозволяє перевіряти функціональність системи без наявності фізичного обладнання. Пристрій у симульованому режимі (Simulated) імітує стандартні відповіді на запити, не потребуючи реального SSH-з'єднання. Це дає змогу тестувати сценарії, логіку шаблонізації та моніторинг без впливу на реальну мережу. Усі симульовані пристрої чітко позначені в таблиці, що забезпечує контроль за розмежуванням продуктивного та тестового середовища.

Пристрої, додані до системи, можуть взаємодіяти з модулями шаблонів (Templates) та моніторингу (Monitoring), забезпечуючи наскрізну інтеграцію логіки конфігурування, зчитування статусу та реагування на аномалії. Це дозволяє оператору повністю управляти інфраструктурою з єдиного інтерфейсу, використовуючи стандартизовані методи доступу, незалежно від вендора чи фізичної доступності пристрою.

Застосування такого підходу гарантує масштабованість та гнучкість системи, а також підтримку принципів DevOps та Infrastructure as Code, що передбачає автоматизоване керування конфігураціями, уніфікацію сценаріїв та швидке тестування змін у симульованому середовищі.

4.5 Генерація шаблонів конфігурацій VLAN

Функціональність створення шаблонів конфігурацій реалізовано у спеціалізованому модулі *Templates*, який забезпечує параметризовану генерацію налаштувань для мережевих пристроїв. Основна мета модуля полягає в

автоматизації процесу конфігурування інтерфейсів VLAN, зменшенні ймовірності помилок під час ручного введення команд, а також прискоренні розгортання типових мережевих сценаріїв.

Інтерфейс шаблонізатора реалізовано у вигляді форм, що дозволяють задати значення змінних:

VLAN ID, IP Address, Gateway, ACL Name. На основі цих параметрів відбувається формування конфігураційного блоку у форматі CLI-команд. Внутрішньо система використовує шаблонізатор Jinja2, який дозволяє підставляти значення змінних у текст шаблону за визначеним синтаксисом (рис. 4.5).

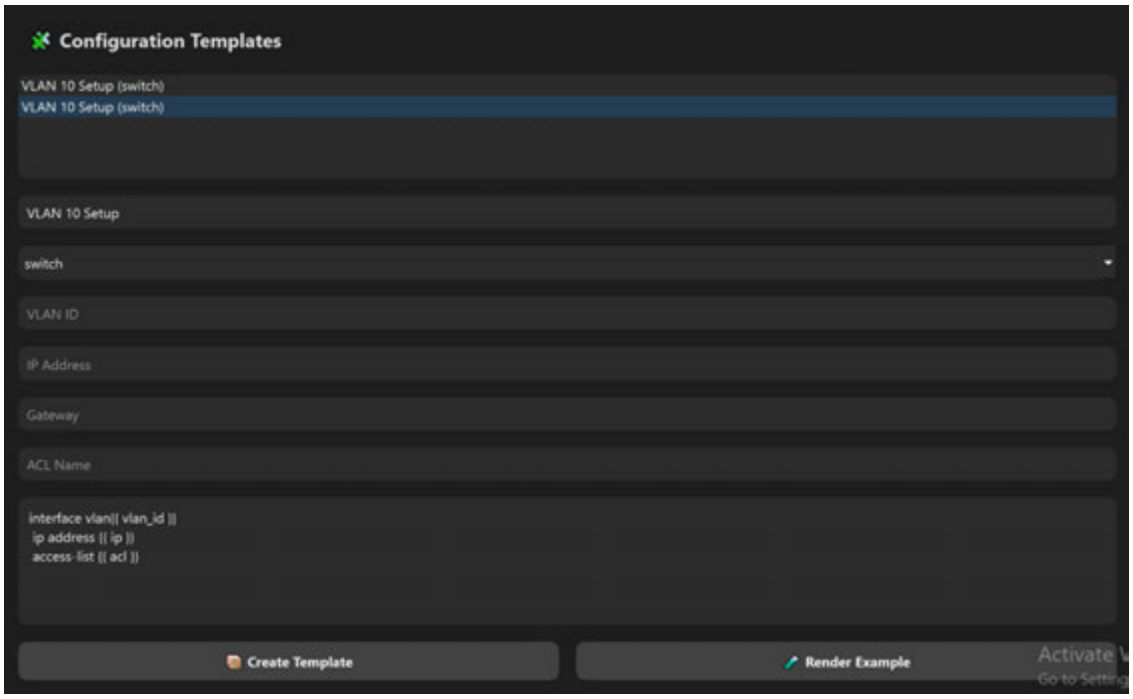

Рисунок 4.5 - Інтерфейс генерації шаблонів конфігурацій VLAN

Після заповнення полів користувач може натиснути Render Example, щоб переглянути згенерований шаблон, або Create Template, щоб зберегти його для подальшого використання. Така реалізація дозволяє створювати та застосовувати стандартизовані конфігурації до пристроїв незалежно від їх вендора, за умови підтримки базового синтаксису командного рядка.

Система також передбачає інтеграцію з модулем Devices, що дозволяє застосовувати шаблони безпосередньо до обраних пристроїв після підтвердження через кнопку Apply Template. При цьому відбувається

автоматичне SSH-з'єднання з пристроєм та надсилання сформованих CLI-команд у сесію, з подальшим збереженням результату у системному журналі.

Застосування шаблонів конфігурації значно знижує трудовитрати при обслуговуванні мережі, підвищує відтворюваність налаштувань та спрощує адаптацію системи до змін у політиках доступу, адресації або структурі VLAN.

4.6 Результати тестування функціональних модулів системи

З метою перевірки працездатності реалізованого програмного забезпечення було проведено комплексне тестування ключових функціональних модулів: моніторингу пристроїв, генерації шаблонів, журналювання подій, взаємодії з пристроями та роботи в симульованому середовищі. Тестування проводилось у контрольованих умовах, із залученням як реальних сценаріїв (SSH-з’єднання, ICMP-запити), так і емуляторів. Метою тестування було оцінити коректність виконання функцій, відповідність результатів очікуваним параметрам та надійність у випадку граничних ситуацій (втрата з’єднання, помилка шаблону тощо).

Результати тестування подано в таблиці 4.3. Для кожного модуля вказано перевірені функції, тип введених даних або сценаріїв, очікувану та фактичну поведінку системи, а також статус виконання.

Таблиця 4.3 – Результати тестування основних функціональних модулів

№	Тестований модуль	Опис дії / сценарію	Очікуваний результат	Фактичний результат	Статус
1	Моніторинг пристроїв	Відповідь від 2 активних IP (ping)	Статус Online, затримка < 100 мс	Online, 71/79 мс	Успішно
2	Моніторинг пристроїв	Вимкнення вузла з IP 192.168.1.2	Статус Timeout, запис у журнал	Timeout, Warning у Logs	Успішно

3	Журналювання подій	Поява 3-х типів подій (INFO/WARNING/ERROR)	Відображення з кольоровими мітками	Відображено коректно	Успішно

Продовження таблиці 4.3

4	Шаблонізація VLAN	Генерація конфігурації VLAN 10	Команди з параметрами IP, ACL	Згенеровано без помилок	Успішно
5	Взаємодія з пристроєм	Застосування шаблону до реального пристрою	SSH-з’єднання, передача команд	Конфігурація застосована	Успішно
6	Емуляція пристроїв	Відповідь симульованого пристрою на ping	Статус Simulated, Online	Simulated, Latency 0	Успішно
7	Захист від помилок	Створення шаблону з пустим ACL	Повідомлення про помилку, блокування запису	Валідація спрацювала	Успішно
8	Автопінг	Періодичне опитування кожні 60 сек	Оновлення затримки без перезавантаження інтерфейсу	Оновлюється динамічно	Успішно

Під час тестування не було виявлено критичних збоїв або зупинок роботи. Система коректно реагувала на відсутність з’єднання, некоректні вхідні дані та переключення між емулятором і реальним режимом. Механізм журналювання виконує трасування всіх подій із точністю до секунди, що забезпечує можливість ретроспективного аналізу роботи.

ВИСНОВКИ

У ході виконання дипломної роботи реалізовано повний цикл дослідження, проєктування та розробки програмного засобу конфігураційного управління мережею, що забезпечує централізовану автоматизацію адміністрування, контроль змін конфігурації та відповідність сучасним стандартам безпеки. Робота охоплює як теоретичний аналіз, так і практичну реалізацію функціонального прототипу, що відповідає заявленим у вступі завданням.

1. Проаналізовано сучасні підходи та засоби конфігураційного управління мережами. Проведено порівняльний аналіз найбільш поширених систем управління (Cisco DNA Center, Juniper Contrail, Huawei eSight, Ansible, Puppet), визначено їх переваги та обмеження в контексті інтеграції, масштабованості, автоматизації та безпеки (див. табл. 1.2). Окреслено сучасні виклики галузі — потребу в мультивендорному доступі, централізації конфігурацій, моніторингу та аудиті змін.

2. Сформовано концептуальну модель системи конфігураційного управління. Визначено архітектуру рішення, що охоплює модулі взаємодії з пристроями (SSH/RESTCONF), моніторингу (ICMP), шаблонізації конфігурацій (Jinja2), журналювання подій, авторизації (LDAP/OAuth 2.0), а також базу даних для зберігання конфігурацій. Відповідно до вимог (табл. 1.3–1.6) розроблено логічну, структурну, функціональну та монтажну схеми мережі.

3. Реалізовано програмний прототип системи. Здійснено розробку графічного інтерфейсу (розділи Dashboard, Devices, Monitoring, Templates, Logs), реалізовано автоматичне пінгування пристроїв, створення шаблонів VLAN-конфігурацій з параметрами, зчитування конфігурацій з реальних або симульованих пристроїв через SSH. Підтримано мультивендорний режим та емуляцію для тестового середовища.

4. Проведено тестування ефективності реалізованих функцій.

5. За результатами тестів (табл. 4.3) підтверджено коректну роботу всіх модулів, включно з виявленням недоступних пристроїв, динамічним оновленням статусів, логуванням подій, застосуванням шаблонів до реального обладнання. Усі функціональні сценарії виконані успішно без критичних збоїв.

6. Здійснено порівняльний аналіз з існуючими системами. Розроблене рішення забезпечує функціональну відповідність основним корпоративним платформам, але відрізняється нижчим порогом входу, відкритістю архітектури, підтримкою емуляції, що робить його придатним для застосування у малих та середніх організаціях, лабораторіях та освітніх закладах.

Усі поставлені завдання дипломної роботи були виконані у повному обсязі, а розроблений програмний засіб підтвердив свою ефективність у сценаріях моніторингу, конфігурування та аудиту конфігураційних змін. Запропоноване рішення може стати основою для подальшого розширення функціоналу (наприклад, підтримки Netconf/YANG), масштабування до хмарних середовищ або інтеграції з CMDB-системами.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сіваковська О. М. До проблем координування конфігурацій продуктів та їх проектів: монографія. Луцьк: ЛНТУ, 2018. 196 с.lib.lntu.edu.ua

2. Бурячок В. Л. Технології забезпечення безпеки мережевої інфраструктури. Київ: Київський університет імені Бориса Грінченка, 2019. 120 с.

3. Клушин Ю. С., Цап'як М. А. Принцип побудови кіберфізичної системи контролю роботи розумної теплиці. Комп'ютерні системи та мережі. 2021. № 1. С. 49–59.Science Lviv Polytechnic

4. Бйорклунд М. YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020. 2010.Wikipedia

5. Björklund M. The YANG 1.1 Data Modeling Language. RFC 7950. 2016. Wikipedia

6. Enns R., Björklund M., Schönwälder J., Bierman A. Network Configuration Protocol (NETCONF). RFC 6241. 2011.Wikipedia

7. Bierman A., Björklund M., Watsen K. RESTCONF Protocol. RFC 8040. 2017.Wikipedia

8. Boutaba R., Aib I. Policy-Based Management: A Historical Perspective. Journal of Network and Systems Management. 2007. Vol. 15, No. 4. P. 447–480. Wikipedia+1Wikipedia+1

9. Sloman M. S. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management. 1994. Vol. 2, No. 4. P. 333–360. Wikipedia

10. Verma D. Simplifying network administration using policy-based management. IEEE Network. 2002. Vol. 16, No. 2. P. 20–26. Wikipedia+2Wikipedia+2Wikipedia+2

11. Agrawal D., Calo S., Lee K., Lobo J., Verma D. Policy Technologies for Self Managing Systems. IBM Press, 2008.Wikipedia

12. Loyola J. R., Serrat J., Charalambides M., Flegkas P., Pavlou G. A Methodological Approach toward the Refinement Problem in Policy-Based Management Systems. IEEE Communications Magazine. 2006. Vol. 44, No. 10. P. 60–68.Wikipedia

13. Bandara A. K., Lupu E. C., Russo A., Dulay N., Sloman M., Flegkas P., Charalambides M., Pavlou G. Policy Refinement for IP Differentiated Services Quality of Service Management. IEEE Transactions on Network and Service Management. 2006. Vol. 2, No. 2. P. 1–10.Wikipedia

14. Aib I., Boutaba R. Business-driven optimization of Policy Based Management Solutions; A Web Application Hosting SLA Use Case. In: Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM'2007), Munich, Germany, May 2007.Wikipedia

15. Aib I., Boutaba R. On leveraging policy-based management for maximizing business profit. IEEE Transactions on Network and Service Management. 2007. Vol. 4, No. 3. P. 163–176.Wikipedia

16. Galis A. Multi-Domain Communication Management. CRC Press LLC, 2000.Wikipedia

17. Pavlou G. On the evolution of management approaches, framework and protocols: A historical perspective. Journal of Network and Systems Management. 2007. Vol. 15. P. 425–445.Wikipedia+1Wikipedia+1

18. Pras A., van Beijnum B.-J., Sprenkels R. Introduction to TMN. University of Twente, Enschede, The Netherlands, CTIT Technical Report 99-09, Apr. 1999.Wikipedia

19. Foley C., Balasubramaniam S., Power E., Ponce de Leon M., Botvich D., Dudkowski D., Nunzi G., Mingardi C. A Framework for In-Network Management in

Heterogeneous Future Communication Networks. In: Proceedings of MACE 2008, Samos Island, Greece, September 22–26, 2008.Wikipedia

20. Dudkowski D., Brunner M., Nunzi G., Mingardi C., Foley C., Ponce de Leon M., Meirosu C., Engberg S. Architectural Principles and Elements of In-Network Management. In: Mini-conference at IFIP/IEEE Integrated Management symposium, New York, USA, 2009.Wikipedia

21. Gonzalez Prieto A., Dudkowski D., Meirosu C., Mingardi C., Nunzi G., Brunner M., Stadler R. Decentralized In-Network Management for the Future Internet. In: IEEE International Workshop on the Network of the Future at IEEE ICC'09, Dresden, Germany, 2009.Wikipedia

22. Sherwin J., Sreenan C. J. Software-Defined Networking for Data Centre Network Management: A Survey. arXiv preprint arXiv:2106.10014. 2021. arxiv.org+1Wikipedia+1

23. Cuppens F., Cuppens-Boulahia N., Garcia-Alfaro J. Misconfiguration Management of Network Security Components. arXiv preprint arXiv:1912.07283. 2019.arxiv.org

24. Gedia D., Perigo L. NetO-App: A Network Orchestration Application for Centralized Network Management in Small Business Networks. arXiv preprint arXiv:1808.01519. 2018.

25. Zhang Y., Xue Y., Liu Q., Choi N., Han T. RoNet: Toward Network Configuration Management with Reinforcement Learning. *IEEE Access*. 2020. Vol. 8. P. 225360–225374.

26. Kim H., Feamster N. Improving network management with software defined networking. *IEEE Communications Magazine*. 2013. Vol. 51, No. 2. P. 114–119.

27. Kreutz D., Ramos F. M. V., Verissimo P. E., Rothenberg C. E., Azodolmolky S., Uhlig S. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*. 2015. Vol. 103, No. 1. P. 14–76.

28. Tootoonchian A., Ganjali Y. HyperFlow: A Distributed Control Plane for OpenFlow. *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. 2010. P. 3–3.

29. Anderson C. Network Programmability with YANG: The Structure of Network Automation. *Addison-Wesley Professional*, 2019. 368 p.

30. Wang L., Chen Y., Zhang Z. A Survey of Network Configuration Management. *Journal of Network and Computer Applications*. 2018. Vol. 86. P. 1–10.

31. ISO/IEC 19086-1:2016. Cloud computing — Service level agreement (SLA) framework — Part 1: Overview and concepts. International Organization for Standardization, 2016.

32. ISO/IEC 27033-1:2015. Information technology — Security techniques — Network security — Part 1: Overview and concepts. International Organization for Standardization, 2015.

33. Lopes C., Monteiro E., Granville L. Z. A Survey of Management in Future Internet Architectures. *IEEE Communications Surveys & Tutorials*. 2015. Vol. 17, No. 3. P. 1199–1228.

34. Turner J., Taylor D. Diversifying the Internet. *Proceedings IEEE GLOBECOM 2005*. Vol. 2. P. 6–11.

35. Kurose J. F., Wetherall D. P. Computer Networking: A Top-Down Approach. 8th ed. Pearson, 2021. 864 p.

36. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Pearson, 2010. 960 p.

37. Bishop M. Computer Security: Art and Science. 2nd ed. Addison-Wesley, 2018. 1440 p.

38. Hunt C. TCP/IP Network Administration. 3rd ed. O’Reilly Media, 2002. 748 p.

39. Barrett D., King C., Wang B. 21st Century C: C Tips from the New School. O'Reilly Media, 2014. 424 p.

40. Cisco Systems. Cisco Networking Basics. Cisco Press, 2020. 350 p.

ДОДАТОК А

Програмний код моніторингу мережі

```
import json
import random
from datetime import datetime
from pathlib import Path

from PyQt6.QtWidgets import (
    QWidget, QVBoxLayout, QLabel, QPushButton, QTableWidget,
    QTableWidgetItem, QHeaderView, QMessageBox, QHBoxLayout
)
from PyQt6.QtCore import Qt, QTimer
import pyqtgraph as pg
from ping3 import ping

DEVICES_FILE = Path("data/devices.json")
LOGS_FILE = Path("data/logs.json")


class MonitoringTab(QWidget):
    def __init__(self):
        super().__init__()

        self.latency_data = {}  # {ip: [values]}

        layout = QVBoxLayout()
        layout.setContentsMargins(24, 24, 24, 24)
        layout.setSpacing(20)

        title = QLabel("· Network Monitoring")
        title.setStyleSheet("font-size: 18px; font-weight: bold;")
        layout.addWidget(title)

        self.plot = pg.PlotWidget()
        self.plot.setBackground("#1e1e1e")
        self.plot.showGrid(x=True, y=True)
        self.plot.addLegend()
        self.plot.setTitle("Device Latency Over Time", color="white",
size="12pt")
        self.plot.setLabel("left", "Latency (ms)", **{"color": "white"})
        self.plot.setLabel("bottom", "Time", **{"color": "white"})
        layout.addWidget(self.plot)

        self.table = QTableWidget()
        self.table.setColumnCount(4)
        self.table.setHorizontalHeaderLabels(["IP", "Status", "Latency", "Last
Check"])

self.table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretc
h)
        self.table.setStyleSheet("""
            QTableWidget {
                background-color: #2a2a2a;
                color: #ffffff;
```

```
                border-radius: 8px;
            }
            QHeaderView::section {
                background-color: #1e1e1e;
                color: #cccccc;
                padding: 6px;
                font-weight: bold;
            }
        """)
        layout.addWidget(self.table)

        button_layout = QHBoxLayout()

        self.check_button = QPushButton("· Ping Devices Now")
        self.check_button.setFixedHeight(36)
        self.check_button.setStyleSheet("""
            QPushButton {
                background-color: #424242;
                color: white;
                border-radius: 8px;
                font-weight: bold;
            }
            QPushButton:hover {
                background-color: #616161;
            }
        """)
        self.check_button.clicked.connect(self.perform_ping)
        button_layout.addWidget(self.check_button)

        self.toggle_auto = QPushButton(" Auto Ping: ON")
        self.toggle_auto.setCheckable(True)
        self.toggle_auto.setChecked(True)
        self.toggle_auto.setFixedHeight(36)
        self.toggle_auto.setStyleSheet("""
            QPushButton {
                background-color: #37474f;
                color: white;
                border-radius: 8px;
                font-weight: bold;
            }
            QPushButton:checked {
                background-color: #00897b;
            }
        """)
        self.toggle_auto.clicked.connect(self.toggle_auto_ping)
        button_layout.addWidget(self.toggle_auto)

        layout.addLayout(button_layout)
        self.setLayout(layout)

        self.auto_timer = QTimer()
        self.auto_timer.timeout.connect(self.perform_ping)
        self.auto_timer.start(3000)  # кожні 5 хвилин

        self.perform_ping()

    def toggle_auto_ping(self):
        if self.toggle_auto.isChecked():
            self.auto_timer.start(300_000)
            self.toggle_auto.setText(" Auto Ping: ON")
        else:
            self.auto_timer.stop()
            self.toggle_auto.setText(" Auto Ping: OFF")
```

```
    def perform_ping(self):
        try:
            with open(DEVICES_FILE, "r", encoding="utf-8") as f:
                devices = json.load(f)
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Failed to read devices:
{str(e)}")
            return

        if not LOGS_FILE.exists():
            LOGS_FILE.write_text("[]", encoding="utf-8")

        self.table.setRowCount(len(devices))
        current_time = datetime.now().strftime("%H:%M:%S")
        log_entries = []

        for i, dev in enumerate(devices):
            ip = dev.get("ip", "")
            is_simulated = dev.get("is_simulated", False)

            if is_simulated:
                latency = random.randint(20, 100)
                packet_loss = random.random() < 0.1
            else:
                try:
                    result = ping(ip, timeout=1)
                    latency = int(result * 1000) if result else None
                    packet_loss = result is None
                except:
                    latency = None
                    packet_loss = True

            # Статус
            if packet_loss:
                status = "· Offline"
                level = "error"
            elif latency < 50:
                status = "· Online"
                level = "info"
            else:
                status = "· Warning"
                level = "warning"

            latency_str = f"{latency} ms" if latency is not None else "Timeout"
            self.table.setItem(i, 0, QTableWidgetItem(ip))
            self.table.setItem(i, 1, QTableWidgetItem(status))
            self.table.setItem(i, 2, QTableWidgetItem(latency_str))
            self.table.setItem(i, 3, QTableWidgetItem(current_time))

            if ip not in self.latency_data:
                self.latency_data[ip] = []
            self.latency_data[ip].append(latency if latency is not None else 0)
            self.latency_data[ip] = self.latency_data[ip][-10:]

            log_entries.append({
                "timestamp": datetime.now().isoformat(timespec="seconds"),
                "event": f"Ping result: {status}",
                "ip": ip,
                "level": level
            })

        self.plot.clear()
```

```
        for ip, latencies in self.latency_data.items():
            self.plot.plot(
                list(range(len(latencies))),
                latencies,
                name=ip,
                pen=pg.mkPen(color=(random.randint(100, 255),
random.randint(100, 255), 255), width=2)
            )

        try:
            with open(LOGS_FILE, "r", encoding="utf-8") as f:
                logs = json.load(f)
            logs.extend(log_entries)
            with open(LOGS_FILE, "w", encoding="utf-8") as f:
                json.dump(logs[-500:], f, indent=2)
        except Exception as e:
            QMessageBox.warning(self, "Log Error", f"Could not write logs:
{str(e)}")
```

ДОДАТОК Б

Лістинінг коду конфігурування мережі

```
import json
from pathlib import Path
from PyQt6.QtWidgets import (
    QWidget, QVBoxLayout, QLabel, QLineEdit, QTextEdit, QPushButton,
    QListWidget, QHBoxLayout, QMessageBox, QComboBox
)
from PyQt6.QtCore import Qt
from jinja2 import Template

TEMPLATES_FILE = Path("data/templates.json")

DEFAULT_TEMPLATE = {
    "name": "Default VLAN Template",
    "device_type": "switch",
    "template": (
        "interface vlan{{ vlan_id }}\n"
        " ip address {{ ip }} 255.255.255.0\n"
        " description Gateway: {{ gateway }}\n"
        " access-group {{ acl }} in"
    )
}


class TemplatesTab(QWidget):
    def __init__(self):
        super().__init__()
        self.templates = []

        layout = QVBoxLayout()
        layout.setContentsMargins(24, 24, 24, 24)
        layout.setSpacing(16)

        title = QLabel("· Configuration Templates")
        title.setStyleSheet("font-size: 18px; font-weight: bold;")
        layout.addWidget(title)

        self.list_widget = QListWidget()
        self.list_widget.setStyleSheet("background-color: #2b2b2b; color: white;
border-radius: 8px;")
        self.list_widget.itemClicked.connect(self.on_template_selected)
        layout.addWidget(self.list_widget)

        # Поля для створення шаблону
        self.name_input = QLineEdit()
        self.name_input.setPlaceholderText("Template Name")

        self.device_type_input = QComboBox()
        self.device_type_input.addItems(["switch", "router", "firewall"])
        self.device_type_input.setStyleSheet("background-color: #2c2c2c; color:
white;")
```

```
        self.vlan_input = QLineEdit()
        self.vlan_input.setPlaceholderText("VLAN ID")

        self.ip_input = QLineEdit()
        self.ip_input.setPlaceholderText("IP Address")

        self.gateway_input = QLineEdit()
        self.gateway_input.setPlaceholderText("Gateway")

        self.acl_input = QLineEdit()
        self.acl_input.setPlaceholderText("ACL Name")

        self.template_body = QTextEdit()
        self.template_body.setPlaceholderText("Jinja2 Template Body")
        self.template_body.setFixedHeight(120)

        # Додаємо стилі і поля в layout
        for widget in [
            self.name_input, self.device_type_input,
            self.vlan_input, self.ip_input, self.gateway_input,
            self.acl_input, self.template_body
        ]:
            layout.addWidget(widget)
            widget.setStyleSheet("background-color: #2c2c2c; color: white;
border-radius: 8px; padding: 6px;")

        # Кнопки
        button_row = QHBoxLayout()
        self.create_btn = QPushButton("· Create Template")
        self.render_btn = QPushButton("· Render Example")

        self.create_btn.clicked.connect(self.save_template)
        self.render_btn.clicked.connect(self.render_template)

        for btn in [self.create_btn, self.render_btn]:
            btn.setFixedHeight(36)
            btn.setStyleSheet("""
                QPushButton {
                    background-color: #424242;
                    color: white;
                    border-radius: 8px;
                    font-weight: bold;
                }
                QPushButton:hover {
                    background-color: #616161;
                }
            """)
            button_row.addWidget(btn)

        layout.addLayout(button_row)
        self.setLayout(layout)

        self.load_templates()

    def load_templates(self):
        if not TEMPLATES_FILE.exists():
            TEMPLATES_FILE.write_text(json.dumps([DEFAULT_TEMPLATE], indent=2),
encoding="utf-8")

        try:
            with open(TEMPLATES_FILE, "r", encoding="utf-8") as f:
                self.templates = json.load(f)
```

```
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Failed to load templates:
{str(e)}")
            return

        self.list_widget.clear()
        for tpl in self.templates:
            name = tpl["name"]
            dtype = tpl.get("device_type", "-")
            self.list_widget.addItem(f"{name} ({dtype})")

    def save_template(self):
        name = self.name_input.text().strip()
        device_type = self.device_type_input.currentText()
        body = self.template_body.toPlainText().strip()

        if not name or not body:
            QMessageBox.warning(self, "Warning", "Please enter template name and
body.")
            return

        tpl = {
            "name": name,
            "device_type": device_type,
            "template": body
        }

        self.templates.append(tpl)
        try:
            with open(TEMPLATES_FILE, "w", encoding="utf-8") as f:
                json.dump(self.templates, f, indent=2)
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Failed to save template:
{str(e)}")
            return

        self.name_input.clear()
        self.template_body.clear()
        self.load_templates()

        QMessageBox.information(self, "Saved", "Template saved successfully.")

    def render_template(self):
        try:
            tpl = Template(self.template_body.toPlainText())
            output = tpl.render(
                vlan_id=self.vlan_input.text(),
                ip=self.ip_input.text(),
                gateway=self.gateway_input.text(),
                acl=self.acl_input.text()
            )
            QMessageBox.information(self, "Rendered Output", output)
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Failed to render template:
{str(e)}")

    def on_template_selected(self, item):
        text = item.text()
        name = text.split(" (")[0]
        for tpl in self.templates:
            if tpl["name"] == name:
                self.name_input.setText(tpl["name"])
                self.template_body.setPlainText(tpl["template"])
```

```
                index = self.device_type_input.findText(tpl.get("device_type",
"switch"))
                self.device_type_input.setCurrentIndex(index if index >= 0 else
0)
                break
```