**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Сумський державний університет**
**Факультет електроніки та інформаційних технологій**
**Кафедра інформаційних технологій**

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня магістр**

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-наукової програми «Інформаційні технології проєктування»

на тему: Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів

Здобувача групи ІТ.м-21н. Шевченка Данила Олександровича
(шифр групи) (прізвище, ім'я, по батькові)

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

_____ Данило ШЕВЧЕНКО
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник ____ к.т.н., доц. Вікторія АНТИПЕНКО_____ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

**Суми – 2024**

**Сумський державний університет**
**Факультет** електроніки та інформаційних технологій
**Кафедра** інформаційних технологій
**Спеціальність** 122 «Комп'ютерні науки»
**Освітньо-наукова програма** «Інформаційні технології проєктування»

**ЗАТВЕРДЖУЮ**
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«____» _____ 2024 р.

# ЗАВДАННЯ

**на кваліфікаційну роботу магістра студентові**

*Шевченку Данилу Олександровичу*
**(прізвище, ім'я, по батькові)**

**1 Тема кваліфікаційної роботи** Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів

затверджена наказом по університету від «01» лютого 2024 р. № 0096-VI

**2 Термін здачі студентом кваліфікаційної роботи** «___» __травня___ 2024 р.

**3 Вхідні дані до кваліфікаційної роботи** план робіт, технічна документація Ansible, технічна документація Terraform, специфікація мережевого обладнання.

**4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)** аналіз предметної області, постановка задачі, методи дослідження, проєктування інформаційної технології автоматизації конфігурації мережевих пристроїв та сервісів, розробка вебплатформи, тестування вебплатформи.

**5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації)** актуальність дослідження, постановка задачі, задачі дослідження, огляд технологій автоматизації розгортання конфігурацій, результати проведеного огляду технологій, функціональні вимоги, засоби реалізації, методи дослідження, архітектура вебплатформи, структурно-функціональне моделювання, діаграма варіантів використання, практична реалізація, демонстрація роботи вебплатформи, апробація, висновки.

**6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:**

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

## КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Планування робіт | 20.02.2024 | |
| 2 | Проведення аналізу предметної області | 26.02.2024 | |
| 3 | Проєктування структури інформаційної технології | 11.03.2024 | |
| 4 | Розробка функціональної частини вебплатформи | 10.04.2024 | |
| 5 | Тестування роботи вебплатформи | 15.04.2024 | |

Магістрант _____ Данило ШЕВЧЕНКО

Керівник роботи _____ к.т.н., доц. Вікторія АНТИПЕНКО

# АНОТАЦІЯ

Тема роботи: «Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 45 найменувань, додатків. Загальний обсяг роботи − 134 сторінок, у тому числі 76 сторінок основного тексту, 5 сторінки списку використаних джерел, 53 сторінок додатків.

Мета роботи: автоматизація процесів конфігурації мережевих пристроїв і сервісів за рахунок розробки інформаційної технології у вигляді відповідної вебплатформи для зменшення ймовірності виникнення ризику появи помилок через людський фактор і скорочення часу на зміну налаштувань.

У цій роботі детально проаналізовано поточний стан процесів керування та конфігурації мережі, включаючи останні дослідження, публікації та існуючі програмні рішення в цій галузі. Чітко визначені завдання та цілі розробки, а також вибір методів та інструментів для реалізації. До них відносяться створення бази даних, модулів для автоматизованої генерації та перевірки конфігурацій, інтеграції з системами контролю версій та механізмів автоматизованого розгортання налаштувань. Усі ці компоненти були синтезовані в комплексне інформаційно-технологічне рішення, спрямоване на зменшення ймовірності виникнення ризику появи помилок через людський фактор і скорочення часу на зміну налаштувань.

Практичне значення даного проєкту полягає в удосконаленні процесів конфігурації та керування мережевими пристроями та службами за допомогою їх автоматизації. Розроблена інформаційна технологія у вигляді відповідної вебплатформи сприяє швидкому та безпомилковому виконанню процесів налаштування, а саме підтримує інтеграцію з системами контролю версій для спрощеного керування змінами конфігурації та підвищує загальну надійність і

гнучкість управління мережею, зменшуючи ймовірність виникнення людської помилки та скорочення часу на здійснення зміни налаштувань.

Ключові слова: КОНФІГУРАЦІЯ МЕРЕЖЕВОГО ПРИСТРОЮ, АВТОМАТИЗАЦІЯ, ІНТЕГРАЦІЯ КОНТРОЛЮ ВЕРСІЙ, ПЕРЕВІРКА КОНФІГУРАЦІЇ, АВТОМАТИЗОВАНЕ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ, КЕРУВАННЯ МЕРЕЖЕЮ, ЕКСПЛУАТАЦІЙНА НАДІЙНІСТЬ, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, ВЕБПЛАТФОРМА.

# ЗМІСТ

# ВСТУП

**Актуальність**. Актуальність даного проєкту зумовлена стрімким розвитком інформаційних технологій, які вимагають від компаній не тільки швидкості реакції на зміни в IT-середовищі, але й забезпечення високої надійності та безпеки мережевих ресурсів. Сучасні бізнес-моделі інтенсивно впроваджують цифрові технології, що підсилює потребу в автоматизації управління мережевими конфігураціями.

Мережева інфраструктура компаній стає все більш складною, що збільшує кількість помилок, здатних спричинити серйозні збої. Автоматизація конфігурацій дозволяє мінімізувати людські помилки, забезпечуючи стабільність та ефективність мережевих систем. Це важливо не тільки для підтримки безперервної роботи існуючих систем, але і для швидкого масштабування і впровадження нових технологій без додаткового ризику збоїв.

Із огляду на ці виклики, створення ефективних інструментів для автоматизації конфігурації мережевих пристроїв та сервісів стає критично необхідним. Розробка таких інструментів допоможе компаніям не тільки підвищити ефективність роботи своїх мережевих відділів, але й забезпечить вищий рівень безпеки та адаптивності до змін у технологічному ландшафті. Враховуючи ці аспекти, дана робота займає актуальне місце в ряду наукових досліджень і практичних застосувань в сфері інформаційних технологій.

**Тема дослідження.** Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів.

**Об’єкт дослідження**. Процес автоматизації конфігурації мережевих пристроїв та сервісів.

**Предмет дослідження.** Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів

**Мета**. Автоматизація процесів конфігурації мережевих пристроїв і сервісів за рахунок розробки інформаційної технології у вигляді відповідної вебплатформи для зменшення ймовірності виникнення ризику появи помилок через людський фактор і скорочення часу на зміну налаштувань.

Для досягнення мети проєкту необхідно виконати наступні задачі:

- провести дослідження предметної області, визначити актуальність роботи, здійснити огляд останніх публікацій, виконати аналіз існуючих технологій автоматизації розгортання конфігурацій;
- визначити функціональні можливості та засоби реалізації проєкту;
- виконати проєктування гнучкої та масштабованої архітектури вебплатформи для її адаптації під різні мережеві середовища та конфігурації, а також алгоритму виконання процесів конфігурації та керування мережевими пристроями та службами;
- розробити модулі для генерації та валідації конфігурацій, створити інструменти, які автоматизують впровадження мережевих конфігурацій;
- виконати інтеграцію з системами контролю версій, забезпечити інтеграцію зі звичайними системами контролю версій для ведення історії змін та налагодження співпраці між розробниками;
- розробити механізми автоматизованого розгортання інфраструктури;
- реалізувати функціонал для автоматизованого розгортання мережевих налаштувань для швидкого впровадження змін у мережеву інфраструктуру;
- провести тестування власної розробки.

**Практична цінність.** Практична цінність цього проєкту полягає в удосконаленні процесів конфігурації та керування мережевими пристроями та службами за допомогою їх автоматизації. Розроблена інформаційна технологія у вигляді відповідної вебплатформи сприяє швидкому та безпомилковому виконанню процесів налаштування, а саме підтримує повну інтеграцію з

системами контролю версій для спрощеного керування змінами конфігурації та підвищує загальну надійність і гнучкість управління мережею, зменшуючи ймовірність виникнення людської помилки та скорочення часу на здійснення зміни налаштувань.

**Наукова новизна.** Розробка інформаційної технології для автоматизації конфігурацій мережевих пристроїв та сервісів та її впровадження у вигляді відповідної вебплатформи дозволить підвищити надійність та швидкість налаштування мережевої інфраструктури, знижуючи кількість помилок, спричинених людським фактором, і удосконалюючи виконання процесів управління конфігураціями. Дослідження використання даної технології дозволить виявити найкращі стратегії її впровадження, що відповідає сучасним вимогам мережевого управління та підтримує постійне оновлення мережевих налаштувань.

Результати даної роботи були апробовані англійською мовою на науково-практичній конференції IMA-2024.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Огляд останніх досліджень і публікацій

У сучасному світі, де швидкість розвитку бізнесу та технологій стрімко зростає, потреба в ефективних інструментах для управління мережевими сервісами та конфігураціями стає дедалі актуальнішою. Розробка вебплатформи для автоматизації конфігурації мережевих пристроїв та сервісів адресує низку специфічних технологічних і бізнес-викликів, які мають важливе значення для сучасних організацій.

Багато процесів у мережевій інфраструктурі обслуговуються переважно вручну, наприклад, вхід у маршрутизатори, комутатори, балансування навантаження, тестування, виявлення пристроїв, інвентаризація тощо. Автоматизація мережі усуває всі ручні операції та впроваджує технічні інструменти для керування мережевою інфраструктурою та сервісами. Автоматизація мережі допомагає підприємствам економити час і гроші, а також робити процеси більш гнучкими [1].

Робота [2] розглядає важливість автоматизації мереж для підвищення ефективності бізнесу. Вона акцентує увагу на тому, як автоматизація мережі може допомогти компаніям оптимізувати процедури, знизити витрати та мінімізувати помилки, пов’язані з людським фактором. Автор аналізує, як швидке впровадження змін у мережі та автоматичне управління конфігураціями дозволяє раціональніше використовувати ресурси, що призводить до зниження витрат на обслуговування та управління мережею. Важливим аспектом є також підвищення надійності мережевої інфраструктури, що досягається за рахунок зменшення помилок через людський фактор, що забезпечує стабільне та безпечне середовище. Стаття підкреслює необхідність інтеграції інструментів автоматизації з існуючими бізнес-процесами і системами управління для

забезпечення максимальної ефективності та демонструє приклади успішного впровадження таких інновацій. Завершуючи, автор наголошує на критичному значенні автоматизації мереж для сучасних компаній, які прагнуть зберегти свою конкурентоспроможність на ринку.

REST API, як визначено в [3], використовується для автоматизації мереж з метою забезпечення більшої ефективності управління мережевими ресурсами. Використання REST API дозволяє адміністраторам мереж впроваджувати складні процеси управління, такі як контроль інвентарю, автоматизація розгортання застосунків і покращення процедур логування та відладки. Стаття також акцентує на необхідності інтеграції різних систем за допомогою API для поліпшення адаптивності та швидкості відповіді мережі на зміни, що значно покращує загальну ефективність мережевих операцій.

Робота [4] підкреслює значення автоматизації мереж у сучасному бізнес-середовищі, де швидкість та ефективність є критично важливими. Автор зазначає, що автоматизація може допомогти управляти складністю мережі, зменшити помилки, пов’язані з людським фактором, та покращити відгук на інциденти безпеки. Основні переваги включають підвищення продуктивності, краще управління ресурсами та зниження витрат на утримання мережі.

Дослідження [5] надає результати щодо обговорення концепції «Configuration as Code», яка є ключовою у сучасній автоматизації мереж і IT-інфраструктур. Вона розглядає переваги використання коду для управління конфігурацією, такі як покращене відновлення після збоїв, легше впровадження змін і збільшення відповідальності через контроль версій. Основний акцент робиться на те, як такий підхід може зменшити помилки, забезпечити відтворюваність налаштувань та підвищити ефективність розгортання інфраструктур.

Стаття [6] аналізує роль автоматизованих систем у підвищенні надійності мережевих операцій. В статті розглядаються різні методології та інструменти, які сприяють кращому керуванню мережею, зокрема програмні засоби та мережеві

конфігурації. Автори обговорюють можливий вплив автоматизації на продуктивність мережі, звертаючи увагу на потенційні прогалини в дослідженні, такі як інтеграція штучного інтелекту та заходи безпеки. Основні міркування стосуються комплексного аналізу сучасних технологій і можливих обмежень у наявних емпіричних даних.

У роботі [7] представлено застосування Python для автоматизації та абстракції мережевих процесів. Дослідження підкреслює універсальність та потужність різноманітних інструментів автоматизації, завдяки якій можна спростити складні мережеві задачі, підвищити ефективність і мінімізувати помилки через автоматизовані сценарії. Особлива увага приділяється практичним способам застосування автоматизації, що має велике значення для мережевих інженерів та адміністраторів.

Робота [8] розглядає стратегічне впровадження технологій автоматизації для оптимізації мережевого середовища. У статті обговорюються інструменти та платформи, такі як Ansible та Cisco DNA Center, які допомагають автоматизувати повторювані мережеві задачі. Автори детально аналізують переваги автоматизації, включаючи швидкість розгортання конфігурації, точніші та розширені можливості моніторингу, а також виклики, пов’язані з інтеграцією та навчанням персоналу.

Стаття [9] висвітлює проєктування нової плагінної системи, яка використовує Python для автоматизації мережі, зосереджуючись на високій конкурентоспроможності та надійності. Ця система покликана вирішити проблеми традиційних інструментів автоматизації, забезпечуючи гнучкість і масштабованість через модульну архітектуру. Автори демонструють, як така структура може поліпшити управління мережевими конфігураціями і ефективно розподіляти завдання на різних пристроях, підвищуючи загальну продуктивність мережі.

Зі зростанням технологічних можливостей та еволюцією мережевої інфраструктури з’являється чимало викликів, які вимагають вдосконалення та

оновлення традиційних підходів до управління мережами. Ця робота акцентує увагу на необхідності автоматизації як засобу зменшення ймовірності виникнення ризику появи помилок через людський фактор і скорочення часу на зміну налаштувань.

Із розвитком глобальних мереж та збільшенням кількості підключених пристроїв, компанії зіштовхуються з величезними обсягами даних та управлінських задач. Сучасні мережеві інфраструктури включають безліч компонентів, які мають бути належно налаштовані для забезпечення безперебійної роботи і високої продуктивності. Ручне управління такими комплексними системами не тільки потребує багато часу, але й є схильним до помилок, що може призвести до серйозних збоїв в роботі та фінансових втрат. Бізнес-середовище вимагає швидкої адаптації до змін у ринкових умовах, включаючи нові вимоги безпеки, регуляторні зміни, та розвиток технологій. Потреба в оперативному впровадженні змін у мережеву інфраструктуру є критичною для забезпечення неперервності бізнесу.

Сучасні технології надають потужні інструменти, які можуть бути інтегровані для автоматизації управління мережами. Використання таких інструментів, як Ansible, Terraform та інших платформ автоматизації, дозволяє адміністраторам ефективно реалізувати стандартні процедури, швидко розгортати нові сервіси та вчасно реагувати на зміни у мережевій інфраструктурі.

Мережева автоматизація та управління інфраструктурою через використання інструментів, таких як Ansible та Terraform, відіграють ключову роль у сучасних IT-операціях. Одним з яскравих прикладів є [10], яка показує, як Ansible може використовуватися для налаштування EIGRP у середовищі GNS3, значно зменшуючи людські помилки та час на конфігурацію мережі. Це дослідження підкреслює потенціал автоматизації у зменшенні людських помилок та підвищенні точності процесів конфігурації.

Інше дослідження [11], розглядає використання Ansible для налаштування протоколів маршрутизації в маршрутизаторах Cisco та Mikrotik через Raspberry PI, демонструючи інноваційний підхід до мережевої автоматизації [2]. Це дослідження висвітлює, як використання відкритого програмного забезпечення та недорогих контролерів може ефективно замінити традиційні, більш дорогі рішення для управління мережею.

У роботах [12-13] підкреслюється важливість простої конфігурації мережевих пристроїв, обговорюється використання Ansible для базової конфігурації маршрутизаторів Cisco. Підкреслюється ефективність Ansible в автоматизації повторюваних завдань, зменшенні помилок та підвищенні оперативності налаштування мережі. Описуються основні аспекти інтеграції автоматизації в мережеві процеси. В посібниках надається детальний огляд функцій та можливостей Ansible, які можуть допомагати мережевим адміністраторам оптимізувати їхні робочі процеси.

Джерела [14-15] висвітлюють практичне застосування Ansible у мережевих середовищах, показуючи різні сценарії та приклади коду, які можуть бути використані для автоматизації мережевих завдань, підкреслюють інтеграцію Ansible, pyATS, Docker, і Twilio API для забезпечення ефективного мережевого моніторингу та управління. Це є надзвичайно корисним для технічних фахівців, які хочуть застосувати автоматизацію на практиці різні інструменти та методики, що є ключовими для розробників та інженерів, які прагнуть підвищити ефективність мережевої інфраструктури.

У контексті Terraform, стаття [16] підкреслює гнучкість та масштабованість Terraform у керуванні інфраструктурою різних хмарних постачальників, відкриваючи двері для широкомасштабного управління IT-ресурсами. Це дослідження підкреслює переваги використання декларативних методів конфігурації для удосконалення контролю та зменшення помилок у розгортанні інфраструктури.

Інтеграція CI/CD Terraform з хмарними середовищами, описана [17], показує, як Terraform може оптимізувати процеси налаштування та оновлення інфраструктури, що є важливим для сучасних хмарних середовищ. Ця інтеграція підкреслює важливість автоматизації у безперервному впровадженні та управлінні, забезпечуючи стабільність та надійність в розгортанні інфраструктурних змін.

Впровадження автоматизації в процеси управління мережею зумовлює переосмислення існуючих підходів і розробку нових методів. Це включає аналіз поточних проблем у мережах, вивчення найкращих практик управління та інтеграцію нових технологічних рішень, що сприятиме не тільки покращенню продуктивності, але й забезпеченню більшої безпеки і надійності систем.

Враховуючи сучасні тенденції та виклики у сфері мережевого управління, дослідження та розробка ефективних інструментів автоматизації є актуальною та важливою задачею. Автоматизація мережевого управління відкриває нові можливості для підприємств, зокрема, надаючи можливість швидкого розгортання нових послуг, знижуючи загальні витрати на виконання IT-операцій та підвищуючи загальну ефективність і надійність систем. Це дозволяє компаніям швидше адаптуватися до ринкових умов та забезпечити кращий досвід для своїх клієнтів.

## 1.2 Огляд сучасного стану технологій

Автоматизація мережевих конфігурацій стала важливою темою з появою складних мережевих інфраструктур, які потребують постійного управління та оновлення. Раніше інженери виконували більшість налаштувань вручну, що потребує багато часу та не застраховане від помилок. Поява таких інструментів, як Puppet [18] (заснований у 2005 році), Chef [19] (заснований у 2009 році), і

Ansible [20] (заснований у 2012 році) значно змінила підхід до управління мережевими конфігураціями, дозволяючи інженерам застосовувати декларативні або імперативні методи для автоматизації рутинних завдань.

Terraform [21], що став доступним на ринку у 2014 році, приніс інновації у сфері інфраструктури як коду (IaC), дозволяючи користувачам ефективно управляти як фізичними, так і віртуальними ресурсами в хмарному середовищі.

Ansible – це потужний інструмент автоматизації, який використовує просту мову опису стану системи на основі YAML (YAML Ain’t Markup Language). Це робить Ansible особливо доступним для системних адміністраторів, які можуть не мати глибоких знань у програмуванні. Він розроблений для автоматизації конфігурації, управління задачами, а також для автоматизації розгортання програмного забезпечення. Загальна архітектура представлена на рисунку 1.1.

Переваги Ansible є такими:

– легкість використання: Ansible використовує YAML для написання плейбуків, що робить його простим для розуміння та використання;

– немає необхідності в агентах: Ansible використовує SSH для з’єднань, уникаючи необхідності встановлення додаткового програмного забезпечення на цільові системи;

– модульність: велика кількість доступних модулів для різних систем і платформ.

Однак Ansible має такі недоліки:

– масштабування: управління досить великими інфраструктурами може бути складнішим без використання додаткових інструментів або налаштувань;

– Ansible не відстежує стан вузлів, на яких він керує конфігураціями, що може ускладнити розробку складних змін у конфігурації.

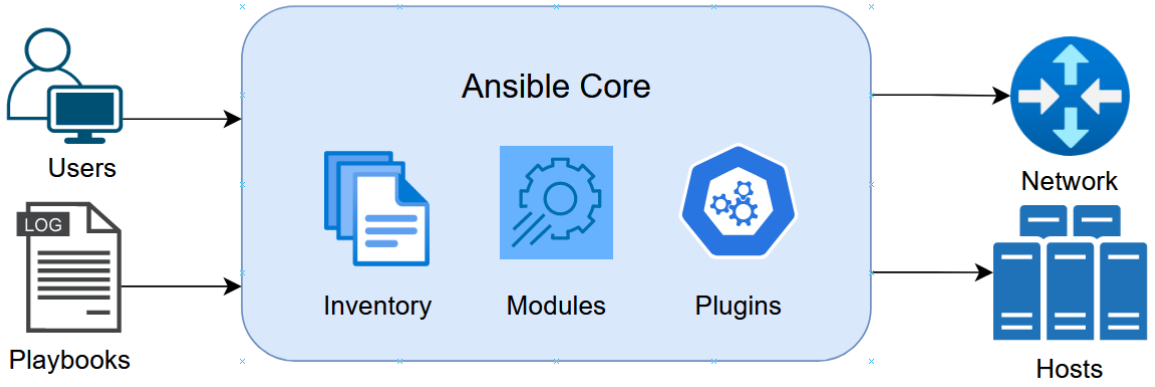



Рисунок 1.1 – Загальна архітектура Ansible

*Джерело*: побудовано автором (знімок з екрану)

Puppet – це інструмент управління конфігурацією, який дозволяє автоматично управляти інфраструктурою, забезпечуючи однаковість середовища і відтворення стану на багатьох серверах. Він використовує власну декларативну мову для опису налаштувань, що дозволяє користувачам легко визначити «яким повинен бути стан» різних систем і сервісів. Загальна архітектура представлена на рисунку 1.2.

Puppet має такі переваги:

– модель даних: Puppet використовує модель стану ресурсу, що дозволяє користувачам легко визначати і управляти станом інфраструктури;

– підтримка великої інфраструктури: добре підходить для масштабних розгортань завдяки потужній централізованій архітектурі.

Однак Puppet має наступні недоліки:

– складність: Puppet може бути складнішим для освоєння порівняно з Ansible через свій DSL (Domain-Specific Language);

– час виконання: Puppet може сповільнюватися при управлінні великими інфраструктурами та мати затримки в оновленні конфігурацій.

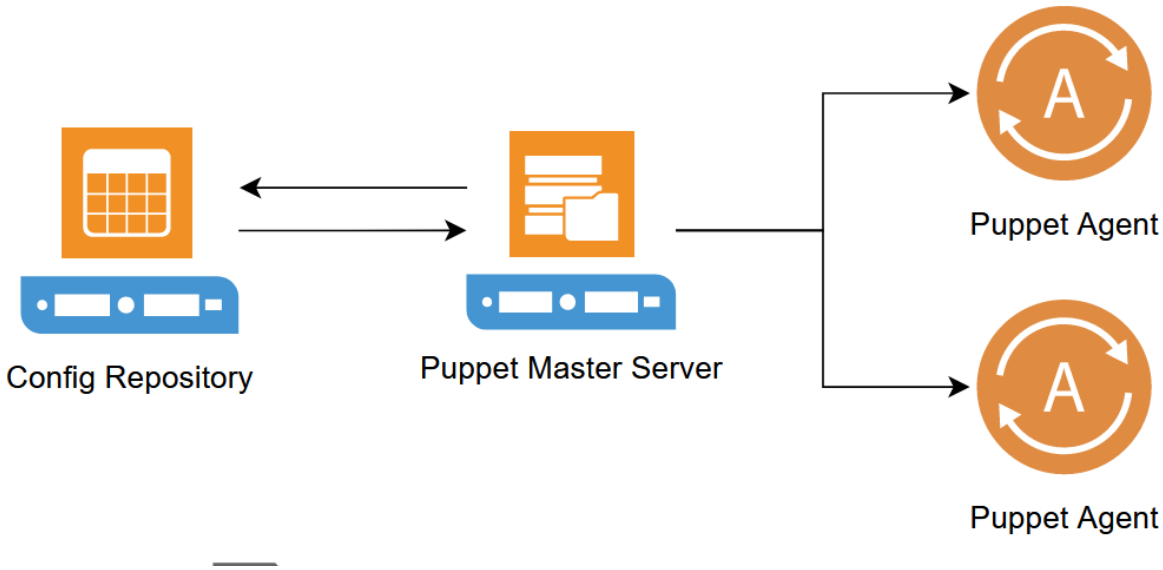



Рисунок 1.2 – Загальна архітектура Puppet

*Джерело*: побудовано автором (знімок з екрану)

Chef – інструмент для управління конфігураціями, який дозволяє розробникам та системним адміністраторам автоматизувати процес управління інфраструктурою як кодом (Infrastructure as Code). Chef використовує мову програмування Ruby для написання «рецептів» та «кухонних книг» (cookbooks), які описують бізнес-процеси та задачі управління конфігурацією в модульній формі. Загальна архітектура представлена на рисунку 1.3.

Chef має такі переваги:

– гнучкість: Chef використовує Ruby як основу для своїх рецептів (recipes), надаючи велику гнучкість в автоматизації;

– сильні інтеграційні можливості: інтеграція з багатьма хмарними платформами та сервісами.

Проте Chef має наступні недоліки:

– висока складність: навчання та освоєння Chef може вимагати більше часу та розуміння програмування;

– виконання рецептів: Іноді виконання рецептів може бути повільним, особливо в великих інфраструктурах.

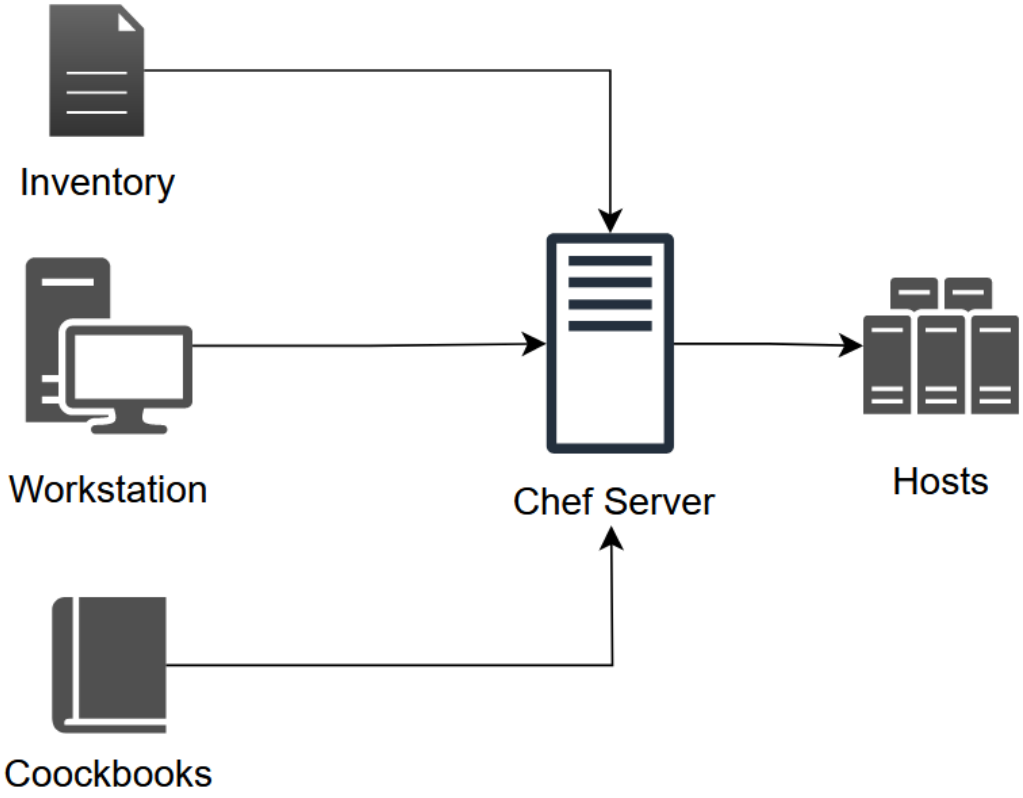



Рисунок 1.3 – Загальна архітектура Chef

*Джерело*: побудовано автором (знімок з екрану)

Terraform – інструмент від HashiCorp для створення, зміни та версіонування інфраструктури безпечно та ефективно. Terraform використовує декларативну мову конфігурації для опису інфраструктури як код, що дозволяє користувачам визначати ресурси хмарних провайдерів, віртуальних машин, мережевих пристроїв тощо. Його особливість полягає в можливості управління не тільки фізичними ресурсами, але й віртуальними, розподіленими по різних платформах і сервісах. Загальна архітектура представлена на рисунку 1.4.

Terraform має такі переваги:

– ідемпотентність: Terraform забезпечує ідентичність результатів при кожному запуску, що робить розгортання інфраструктури передбачуваним;

– підтримка хмарних сервісів: широка підтримка хмарних платформ, з можливістю управління всією хмарною інфраструктурою з одного місця.

Однак Terraform має наступні недоліки:

– залежність від стану: Управління станом інфраструктури може стати складним, особливо в динамічних середовищах;

– не висока функціональність в управлінні конфігураціями.

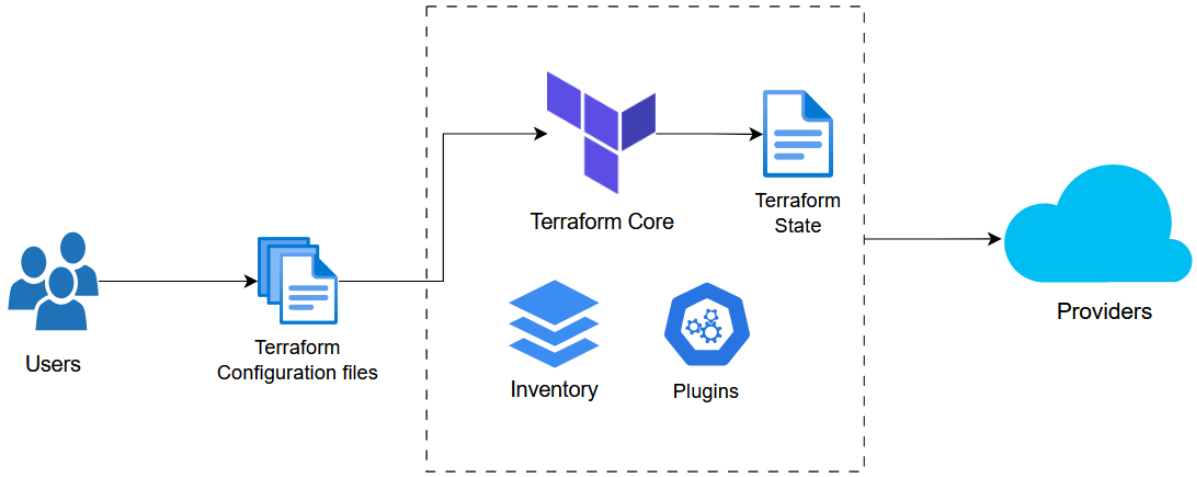



Рисунок 1.4 – Загальна архітектура Terraform

*Джерело*: побудовано автором (знімок з екрану)

Результати проведеного аналізу представлено у таблиці 1.1

Таблиця 1.1 – Порівняльна таблиця інструментів автоматизації налаштування мережі

| Інструмент | Метод конфігурації | Вимога до агента | Мова опису | Основна функція | Підтримка хмарних платформ |
|---|---|---|---|---|---|
| Puppet | Декларативний | Так | Власна DSL | Управління конфігурацією | Висока |
| Chef | Декларативний | Так | Ruby | Управління конфігурацією | Висока |

Продовження табл. 1.1

| | | | | | |
|---|---|---|---|---|---|
| Ansible | Декларативний/ Імперативний | Ні | YAML | Управління конфігурацією/ Розгортання | Середня |
| Terraform | Декларативний | Ні | Власна HCL | Управління інфраструктурою | Висока |

*Джерело*: побудовано автором

На основі результатів із таблиці 1.1 видно, що кожен із розглянутих інструментів має свої переваги та недоліки. Вибір використання конкретного з них часто залежить від специфіки задачі, масштабу інфраструктури та вподобань команди. Для даного проєкту було обрано Ansible і Terraform як основні інструменти автоматизації через їхню високу ефективність і широкі можливості управління конфігураціями. Terraform застосовується для деплойменту інфраструктури, зокрема в хмарних сервісах, що дозволяє з легкістю масштабувати рішення [22]. Ansible використовується для автоматизації налаштувань мережевих пристроїв. Його легко застосовувати без необхідності встановлення додаткових агентів.

## 1.3 Проблеми інтеграції та взаємодії

Інтеграція різних мережевих інструментів в єдину координовану систему представляє собою великий виклик через низку таких проблем:

– сумісність протоколів: різні інструменти часто використовують різні протоколи для зв’язку та обміну даними, що може ускладнити їхню інтеграцію.

Наприклад, один інструмент може використовувати SNMP для моніторингу, тоді як інший – REST API для конфігурації;

− інтерфейси користувача: кожен інструмент зазвичай має свій унікальний інтерфейс користувача, що може ускладнити процес управління та моніторингу для користувачів, які мають працювати з кількома системами одночасно;

− управління конфігурацією: забезпечення послідовності конфігурацій між різними системами може бути складним, особливо коли системи вимагають різних форматів файлів конфігурації;

− масштабування: інтегровані системи повинні ефективно масштабуватися, щоб впоратися зі зростаючою кількістю завдань та розширенням інфраструктури без втрати продуктивності та ефективності.

Для ефективного вирішення проблем інтеграції та забезпечення безпеки в складних мережевих середовищах важливо використовувати низку наступних передових методів і практик:

− використання стандартизованих протоколів: при інтеграції інструментів важливо вибирати такі, які підтримують стандартизовані протоколи (наприклад, REST API), що спрощує їх інтеграцію та взаємодію;

− розширені системи управління ідентифікацією: використання централізованих систем управління ідентифікацією (наприклад, LDAP або Active Directory) допомагає управляти доступом до різних інструментів з єдиної точки;

− застосування шлюзів безпеки та мережевих екранів: використання шлюзів безпеки та мережевих екранів допомагає контролювати трафік між інструментами та запобігати несанкціонованому доступу.

## 1.4 Визначення наявних проблем

Сучасні мережеві інфраструктури зазвичай включають велику кількість пристроїв і сервісів, які потребують регулярних налаштувань. Комплексність таких налаштувань може значно збільшити ризик людських помилок, які, в свою чергу, можуть призвести до серйозних збоїв у роботі мережі.

Причини помилок у комплексних налаштуваннях є такими:

- багато кроків налаштування: чим складніша мережа, тим більше кроків потрібно для її налаштування. Кожен додатковий крок збільшує ймовірність помилки;
- різноманітність технологій: мережі часто складаються з обладнання та програмного забезпечення від різних виробників, що вимагає різних методів конфігурації;
- тиск часу: під тиском термінів адміністратори можуть припускатися помилок.

Автоматизація знижує ризик виникнення помилок через впровадження таких заходів:

- стандартизація процесів: автоматизація дозволяє стандартизувати процеси налаштування мережі, використовуючи заздалегідь визначені шаблони та скрипти, що зменшує можливість людської помилки;
- повторне використання конфігурацій: автоматизація дозволяє зберігати та повторно використовувати перевірені конфігурації, що забезпечує консистентність налаштувань між різними частинами мережі;
- автоматична валідація: багато систем автоматизації мають вбудовані засоби для перевірки та валідації конфігурацій перед їх застосуванням, що дозволяє виявити та виправити помилки до того, як вони стануть причиною збоїв.

Мережеві збої, спричинені помилками в ручних налаштуваннях, часто призводять до великих фінансових втрат та зниження довіри споживачів. Розгляд

таких конкретних випадків допоможе краще зрозуміти, які саме помилки стають причиною проблем:

- неправильне розподіл IP-адрес: наприклад, подвійне призначення IP-адреси двом пристроям у мережі може призвести до конфліктів IP та недоступності важливих сервісів;
- помилки у конфігурації файрволу: неправильні налаштування правил файрволу можуть блокувати легітимний трафік або, навпаки, дозволити доступ потенційно небезпечним запитам;
- помилки при оновленні програмного забезпечення: некоректне або неповне оновлення мережевих компонентів може залишити систему вразливою до атак або збоїв.

Вирішення цих проблем через автоматизацію не тільки знижує ризик помилок, але й значно покращує загальну надійність мережевої інфраструктури.

У сучасному динамічному бізнес-середовищі, швидкість впровадження змін у мережеві налаштування стає критично важливою. Здатність швидко адаптуватися та реагувати на зміни в бізнес-потребах може значно впливати на успішність та конкурентоспроможність компанії.

Потреба в швидкому розгортанні мережевих змін вимагає розгляду наступних аспектів:

- бізнес-динаміка: організації постійно стикаються з необхідністю швидко впроваджувати нові технології та сервіси, адаптуватися до змін у регуляторних вимогах або відповідати на виклики конкурентів. Швидке розгортання дозволяє забезпечувати бізнес-вимоги з мінімальними затримками;
- підтримка інновацій: швидкість впровадження змін є ключовою для підтримки інноваційних проєктів, які вимагають частого оновлення мережевих налаштувань для експериментів та тестування нових ідей;
- вимоги до масштабованості: швидкість розгортання також критична для забезпечення масштабування мережевої інфраструктури відповідно до зростаючого навантаження або географічного розширення.

Інструменти автоматизації такі як Ansible, Puppet, Chef і Terraform дозволяють автоматизувати рутинні процедури розгортання та управління мережевими налаштуваннями, що значно знижує час, необхідний для впровадження змін. Вони використовують шаблони та скрипти, які можна налаштовувати і повторно використовувати, що забезпечує швидке розгортання конфігурацій без необхідності кожного разу вручну втручатися у процес. Це дозволяє інженерам зосередитись на більш складних та інноваційних аспектах мережевого управління, підвищуючи ефективність та надійність мережевої інфраструктури.

Інтеграція систем неперервної інтеграції та неперервного розгортання, таких як Jenkins та GitLab CI/CD [23], стає фундаментальним елементом у стратегіях управління мережевими конфігураціями. Ці системи автоматизують не тільки процес розгортання змін, але й забезпечують неперервне тестування цих змін, що допомагає виявляти і усувати потенційні проблеми ще до того, як вони потраплять у продуктивне середовище. Використання CI/CD пайплайнів забезпечує, що кожна зміна, внесена у конфігурацію, проходить через ретельний процес перевірки, значно знижуючи ризики, пов’язані з людським фактором, і скорочуючи час між внесенням змін та їх впровадженням. Це допомагає забезпечити більшу стабільність та безперебійність роботи мережевої інфраструктури, яка є життєво важливою для підтримки безперервної діяльності сучасних організацій.

# 2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

## 2.1 Мета та задачі дослідження

Розробка архітектури вебплатформи для автоматизації конфігурації мережевих пристроїв та сервісів вимагає бути створеною гнучкою, масштабованою та адаптивною, здатною відповідати різноманітним потребам у різних мережевих середовищах. Основною метою є забезпечення стабільності, високої продуктивності, та легкості управління для користувачів, які можуть мати різний технічний досвід.

Компоненти архітектури є такими:

- інтерфейс користувача: розробка інтуїтивно зрозумілого вебінтерфейсу, який адаптується до різних типів користувачів та їх потреб;
- логіка застосунку: розробка серверної частини, яка обробляє бізнес-логіку, взаємодіє з базами даних і інтегрується з зовнішніми системами або API. Використання мов програмування, таких як Python або Node.js, що забезпечують гнучкість та легкість інтеграції;
- автоматизація конфігурацій: розробка модулів для генерації, валідації та розгортання конфігураційних файлів, використовуючи інструменти автоматизації, такі як Ansible, Puppet, Chef або Terraform;
- зберігання даних: вибір відповідної бази даних (реляційної, як PostgreSQL або MySQL, або NoSQL, як MongoDB або Cassandra) для зберігання даних про користувачів, конфігурації та журнали змін.
- безпека даних: забезпечення безпеки даних через шифрування, регулярне резервне копіювання та відновлення;
- API-шлюзи: використання API-шлюзів для управління взаємодіями між різними сервісами та ізоляції бізнес-логіки від клієнтської частини.

− мікросервісна архітектура: проєктування системи у формі мікросервісів для підвищення масштабованості та надійності.

Підхід до розробки повинен враховувати потребу в масштабуванні системи в залежності від зростаючих вимог. Використання хмарних платформ, таких як AWS, Google Cloud або Azure, забезпечує еластичність ресурсів та дозволяє легко масштабувати систему. Це також включає автоматичне горизонтальне масштабування для обробки піків навантаження та забезпечення високої доступності послуг.

Така гнучка та масштабована архітектура не тільки задовольняє поточні потреби користувачів, але й відкриває можливості для розвитку та адаптації системи під майбутні технологічні зміни та бізнес-вимоги.

## 2.2 Методи дослідження

Структурний синтез задачі [24] в рамках проєкту полягає в детальному розбитті комплексної системи на керовані, чітко визначені компоненти та завдання, що дозволяє забезпечити ефективне управління та виконання проєкту. Це дозволяє чітко розуміти свої ролі, відповідальності, і забезпечити виконання кожного аспекту проєкту згідно з визначеними цілями.

Проєкт поділяється на основні компоненти, кожен з яких включає наступні конкретні задачі:

− конфігурація мережевих пристроїв: створення та модифікація налаштувань для маршрутизаторів і комутаторів;

− конфігурація мережевих сервісів: налаштування DNS, DHCP, файрволів;

− фронтенд розробка: розробка користувацького інтерфейсу для забезпечення доступності та інтуїтивності;

− бекенд розробка: інтеграція серверної логіки з базами даних та зовнішніми сервісами;

− інтеграційні завдання: перевірка взаємодії між модулями та сервісами.

− тестування функціональності: перевірка надійності, безпеки, та відповідності системи до вимог;

− створення документації: розробка технічної та користувацької документації;

− підтримка користувачів: надання технічної підтримки та оновлень;

− використання технічних ресурсів: використання обладнання, програмного забезпечення, та інструментів автоматизації;

− використання людських ресурсів: залучення команди розробників, проєктних менеджерів, та тестувальників.

Структура проєкту також вимагає визначення залежностей між різними задачами, щоб забезпечити логічний та послідовний хід виконання робіт. Використання інструментів проєктного управління, таких як Gantt charts та PERT діаграми, допомагає візуалізувати та управляти цими залежностями (Додаток А).

Теоретико-множинний підхід [25] використовується для формалізації взаємодій та залежностей між різними компонентами системи в рамках проєкту. Цей підхід дозволяє чітко визначити та аналізувати структуру, що сприяє більш ефективному плануванню та управлінню ресурсами.

Множини моделі є такими:

− $C$ (Конфігурації): множина всіх можливих конфігурацій, які потрібно реалізувати в системі;

− $R$ (Ресурси): множина ресурсів, які використовуються для реалізації конфігурацій. Це можуть бути фізичні пристрої, програмне забезпечення, час та персонал;

− $U$ (Користувачі): множина користувачів або стейкхолдерів, які взаємодіють із системою;

– $A$ (Дії): множина дій, які користувачі можуть виконувати в системі, таких як створення, оновлення чи видалення конфігурацій.

Функції та Відношення моделі є наступними:

– $f(C, R) \rightarrow A$: функція, яка описує, які дії ($A$) повинні бути виконані для реалізації кожної конфігурації ($C$) із використанням доступних ресурсів ($R$);

– $g(U, A) \rightarrow C$: функція, що визначає, які конфігурації ($C$) може ініціювати або змінювати користувач ($U$) через певні дії ($A$);

– $h(R, U) \rightarrow A$: Ресурси, що використовуються користувачами, сприяють виконанню дій.

– $i(U, R) \rightarrow C$: Взаємодія користувачів із ресурсами може впливати на конфігурації.

Взаємодія та Залежності моделі є такими:

– $R \times C$: відношення між ресурсами та конфігураціями визначає, які ресурси необхідні для виконання кожної задачі; це відношення допомагає планувати використання ресурсів та їх розподіл;

– $U \times A$: відношення, що показує, які дії можуть виконувати користувачі, і як це впливає на систему; це включає дозволи та ролі користувачів у системі.

Обмеження та умови, задані на множини, дозволяють керувати реалізацією проєкту з урахуванням наступних ресурсних обмежень:

– $\{r \in R \mid \exists c \in C : f(c, r) \in A\}$: множина ресурсів, які є критично важливими для виконання хоча б однієї конфігураційної задачі.

– $\{u \in U \mid \exists a \in A : g(u, a) \in C\}$: множина користувачів, які мають можливість ініціювати зміни в системі.

Діаграма на рисунку 2.1 відображає взаємозв’язки між множинами та функціями в системі.

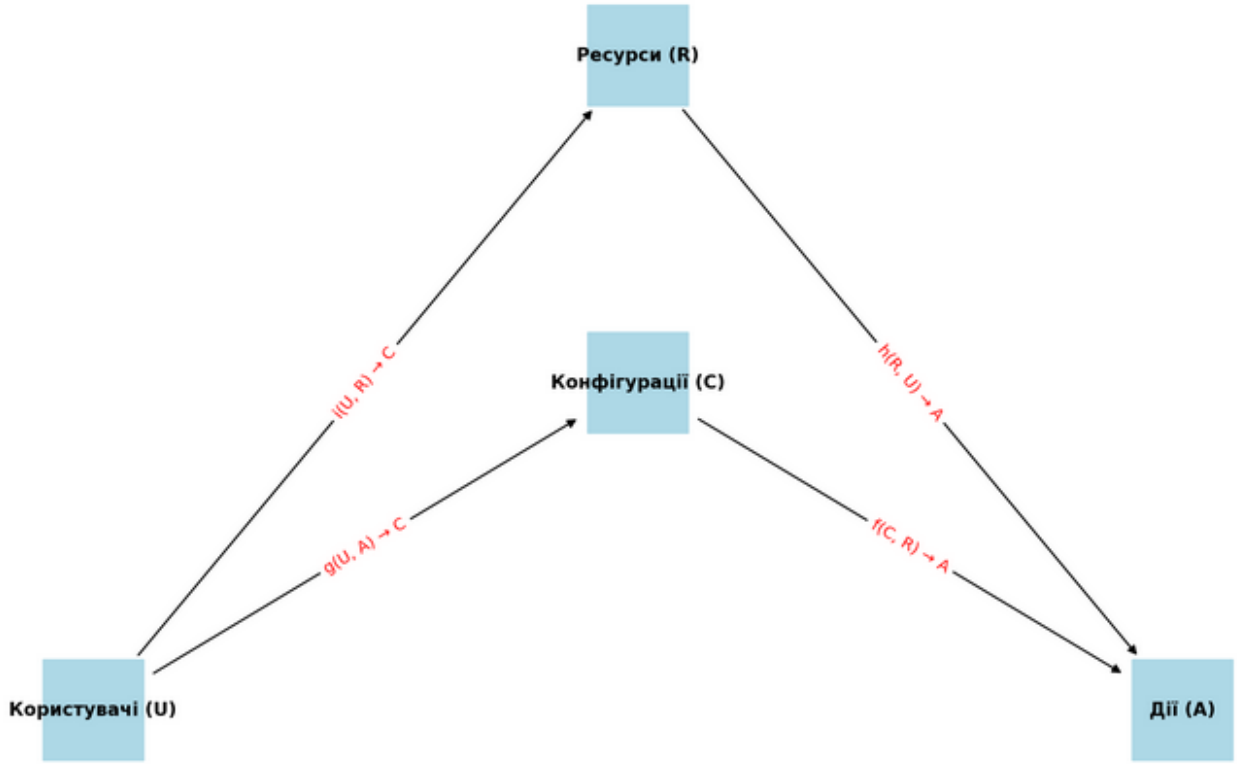



Рисунок 2.1 – Діаграма взаємозв’язків між множинами та функціями

*Джерело*: побудовано автором (знімок з екрану)

Кожна з множин і функцій має чіткі зв’язки, що показують, як вони взаємодіють та впливають одна на одну у рамках проєкту. Наприклад, $j(R) \rightarrow C$ означає вплив ресурсів на конфігурації; $k(A) \rightarrow R$ вказує на те, як дії впливають на ресурси; $l(C) \rightarrow A$ показує взаємозв’язок конфігурацій, які впливають на виконання дій.

Формулювання рівняння для функції $f$, яка відображає взаємодію між множинами конфігурацій $C$ і ресурсів $R$ у дії $A$, представлене формулою (2.1):

$$f: C \times R \rightarrow A, \tag{2.1}$$

де $C$ – множина можливих конфігурацій системи;

$R$ – множина ресурсів, необхідних для реалізації конфігурацій;

$A$ – множина дій, що виконуються в результаті застосування конфігурацій з використанням відповідних ресурсів.

Це означає, що для кожної пари конфігурації та ресурсу визначається відповідна дія.

Формулювання рівняння для функції $g$, яка відображає взаємодію між множиною користувачів $U$ і діями $A$ у конфігурації $C$, представлене формулою (2.2):

$$g: U \times A \rightarrow C, \tag{2.2}$$

де $U$ – множина користувачів, що взаємодіють з системою;
$A$ – множина дій, які можуть виконувати користувачі;
$C$ – множина конфігурацій, які можуть бути створені або змінені в результаті дій користувачів.

Це вказує на те, що взаємодія між користувачами та діями веде до створення або модифікації конфігурацій.

Ці відносини можуть бути детальніше представлені у вигляді матриць, де стовпці і рядки представляють відповідні множини, а елементи матриці показують можливі результати відносин.

Функція $f$ асоціює кожну пару конфігурацій $C$ і ресурсів $R$ з відповідними діями $A$, вона представлена формулою (2.3):

$$f(C,R)=A, \tag{2.3}$$

де $C$ і $R$ є множинами конфігурацій та ресурсів відповідно;
$A$ – множина можливих дій.

Це відображення представлено у вигляді матриці, де кожна комбінація конфігурації і ресурсу вказує на конкретну дію.

Матриця функції *f: C×R→A* (рис. 2.2) представляє відношення між парами конфігурацій *C* і ресурсів *R* та діями *A*. Кожен рядок відповідає конкретній парі конфігурації та ресурсу, а стовпці відповідають діям, які можуть бути виконані для цих пар.


Рисунок 2.2 – Матриця функції *f*

*Джерело*: побудовано автором (знімок з екрану)

Функція g визначає відносини між користувачами *U* і діями *A*, вказуючи які конфігурації *C* можуть бути ініційовані або змінені, це представлено формулою (2.4):

$$g(U,A)=C, \tag{2.4}$$

де *U* – множина користувачів або стейкхолдерів;

*A* – множина дій, що користувачі можуть виконати;

*C* – множина конфігурацій, які можуть бути змінені або створені в результаті цих дій.

Матриця функції g: $U \times A \rightarrow Cg$ (рис. 2.3) представляє відношення між парами користувачів $U$ і дій A до конфігурацій $C$. Кожен рядок відповідає парі користувача та дії, а стовпці відповідають конфігураціям, які можуть бути змінені або створені.

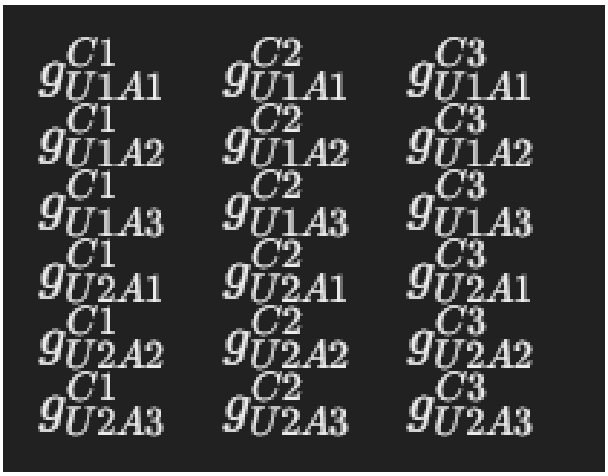



Рисунок 2.3 – Матриця функції g

*Джерело*: побудовано автором (знімок з екрану)

На рисунку 2.4 представлена візуалізація графа, який ілюструє взаємозв’язки між множинами та функціями у вашому проєкті. Кожен вузол представляє множину, а спрямовані зв’язки показують відносини, задані функціями $f$ і $g$.

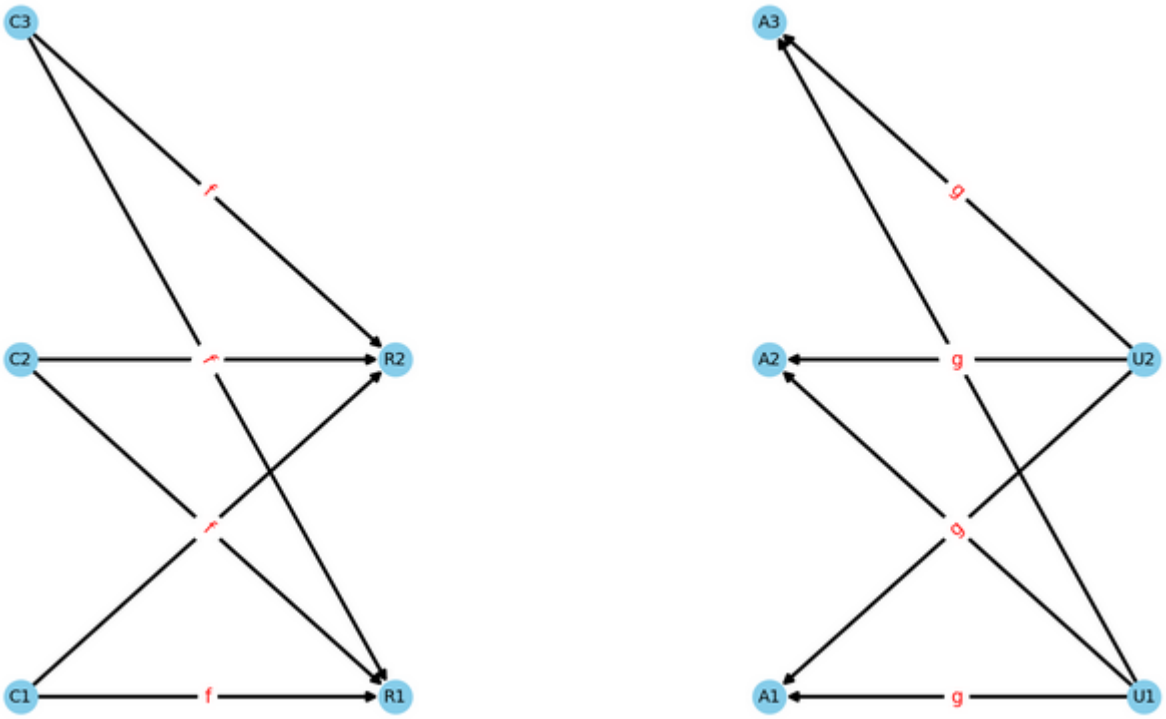



Рисунок 2.4 – Деталізований граф взаємозв’язків між множинами та функціями

*Джерело*: побудовано автором (знімок з екрану)

На візуалізації графа видно вузли, що представляють конфігурації (*C1*, *C2*, *C3*), ресурси (*R1*, *R2*), дії (*A1*, *A2*, *A3*), і користувачів (*U1*, *U2*). Зв’язки між вузлами мають позначення, що відображають функції $f$ та $g$, вказуючи на відносини між різними компонентами системи. Така структура допомагає чітко розуміти взаємодії в системі та можливі шляхи даних і команд.

На цьому етапі можна розглянути декілька наступних кроків для подальшого розвитку:

– інтеграція та тестування – зосередитися на інтеграції розроблених компонентів і проведенні комплексних тестів для перевірки стабільності та ефективності системи;

– запуск пілотної версії – розгорнути пілотну версію системи в обмеженому середовищі для збору зворотного зв’язку від реальних користувачів та їхнього використання в реальних умовах;

– оцінка та покращення – аналізувати зібрані дані для виявлення можливих покращень системи перед повним запуском.

Теоретико-множинний підхід сприяє більшій структурованості і зрозумілості в проєктному управлінні, дозволяючи команді ефективно вирішувати комплексні задачі та оптимізувати використання доступних ресурсів. Завершення структурного синтезу задачі забезпечує основу для подальшого детального планування, виконання, моніторингу та контролю за проєктом, ведучи до його успішної реалізації.

## 2.3 Вибір технологій

При розробці вебплатформи для автоматизації конфігурації мережевих пристроїв та сервісів, формулювання чітких технічних вимог є ключовим етапом, що забезпечує успішність проєкту.

Технічні вимоги охоплюють ряд наступних аспектів, від функціональності інтерфейсу користувача до забезпечення безпеки та інтеграції з іншими системами:

– конфігурованість: інструменти мають дозволяти легке налаштування параметрів конфігурації для відповідності специфічним вимогам мережі; це включає можливість налаштування шаблонів конфігурації, правил валідації та автоматизації розгортання;

– масштабованість: інструменти повинні бути спроєктовані таким чином, щоб вони могли легко масштабуватися від малих до великих мережевих середовищ. Вони мають ефективно обробляти збільшення кількості мережевих вузлів і пристроїв без втрати продуктивності;

– сумісність з різними платформами: інструменти мають підтримувати інтеграцію з різними мережевими обладнаннями та системами, включаючи роутери, перемикачі, файрволи, і хмарні сервіси різних виробників;

– API для зовнішніх інтеграцій: необхідно надати розроблені API, які дозволяють іншим системам та інструментам легко інтегруватися з платформою, забезпечуючи обмін даними і автоматизацію процесів.

Ці вимоги забезпечують основу для розробки потужної, надійної та безпечної платформи автоматизації, яка здатна задовольнити потреби сучасних мережевих середовищ і підвищити загальну ефективність управління мережевою інфраструктурою.

Ansible і Terraform обрано як основні інструменти автоматизації, які забезпечать необхідну гнучкість, масштабованість та безпеку для платформи.

Ansible використовується для автоматизації конфігурацій та управління мережевими пристроями. Його архітектура, заснована на використанні YAML [26] для написання плейбуків, робить його дуже зручним та зрозумілим для системних адміністраторів і розробників. Ansible працює без необхідності встановлення агентів на віддалені машини, що знижує складність управління та вимоги до систем. Це спрощує розгортання та масштабування системи.

Конфігурації, створені за допомогою Ansible, є ідемпотентними, що означає, що повторне виконання плейбуків не змінює стан системи, якщо вона вже знаходиться в бажаному стані. Це забезпечує стабільність і передбачуваність розгортань.

Terraform дозволяє управляти інфраструктурою за допомогою коду. Це включає не тільки мережеві пристрої, але й віртуальні машини, облікові записи хмарних сервісів та інші ресурси. Це дозволяє забезпечити повне управління мережевими ресурсами в рамках однієї системи.

Terraform підтримує модульну структуру, де компоненти інфраструктури можуть бути упаковані в модулі, які легко використовувати повторно в різних середовищах або проєктах.

Terraform ефективно управляє залежностями між ресурсами та дозволяє безпечно впроваджувати зміни, використовуючи план змін, який чітко показує, які зміни будуть застосовані до інфраструктури перед їхнім впровадженням.

Обидва інструменти, Ansible та Terraform, сприяють створенню надійної, безпечної та легко масштабованої платформи, здатної адаптуватися до змінних вимог користувачів та технологічних умов. Їх використання не тільки покращує ефективність процесів управління мережевими конфігураціями, але й забезпечує високий рівень гнучкості для інтеграції нових технологічних рішень у майбутньому.

Ansible є надзвичайно популярним інструментом у спільноті автоматизації, що забезпечується його простотою використання та потужною підтримкою. Велика кількість бібліотек та модулів, які розроблені спільнотою, дозволяють легко інтегрувати Ansible з майже будь-яким мережевим обладнанням або сервісом, незалежно від виробника. Це означає, що незалежно від того, чи потрібно керувати роутерами Cisco, комутаторами Juniper чи файрволами Palo Alto, Ansible може використовувати готові рішення для ефективного управління конфігураціями.

Ansible використовується для автоматизації процесів управління конфігураціями вже розгорнутих пристроїв та систем. Це ідеально підходить для задач, які вимагають швидких змін або оновлень конфігурацій, а також для забезпечення послідовності конфігурацій через різні середовища. Завдяки його модульності Ansible дозволяє легко вносити зміни в конфігурації без ризику порушення стабільності системи.

Terraform є чудовим вибором для розгортання та управління інфраструктурою, особливо в хмарних середовищах. Цей інструмент забезпечує потужні можливості для опису інфраструктури як коду (Infrastructure as Code – IaC), що дозволяє користувачам створювати, змінювати та управляти інфраструктурою в різних провайдерах хмарних послуг (наприклад, AWS, Azure, Google Cloud) із великою точністю та ефективністю. Він дозволяє

автоматизувати розгортання нових сервісів, серверів, мережевих компонентів та інших ресурсів, забезпечуючи управління залежностями та змінами. Terraform дуже добре підходить для сценаріїв, де потрібне швидке та гнучке масштабування інфраструктури.

Використання обох цих інструментів разом в одній платформі дає змогу створити комплексне рішення для управління інфраструктурою. Terraform може використовуватися для розгортання початкових конфігурацій, після чого Ansible може застосовуватися для більш деталізованого управління налаштуваннями та проведення необхідних оновлень. Цей підхід забезпечує максимальну гнучкість та контроль над усією мережевою інфраструктурою.

# 3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ

## 3.1 Структурно-функціональне моделювання

IDEF0 [27] є методологією функціонального моделювання, яка використовується для візуалізації робочих процесів та системних функцій. Вона розроблена в рамках програми ICAM (Integrated Computer Aided Manufacturing). Ця методологія є частиною більшого сімейства IDEF (Integration DEFinition), призначеного для опису різних аспектів інженерних систем.

IDEF0 надає формальний і структурований спосіб представлення процесів та функцій у системі. Вона використовує серію діаграм, що показують, як входи перетворюються на виходи через різні функції. Діаграми IDEF0 складаються з блоків (які представляють функції або процеси) та стрілок, що позначають входи, виходи, механізми виконання та контролі (правила чи вимоги, які керують функціями).

Основні компоненти, які входять до складу IDEF0, є такі:

- входи (Inputs): ресурси, дані, або інші елементи, які потрібні для виконання функції;
- виходи (Outputs): результати або продукти, генеровані функцією.
- механізми (Mechanisms): інструменти, обладнання, технології чи люди, що використовуються для виконання функції;
- керування (Controls): вказівки, правила, або політики, що керують виконанням функції.

Декомпозиція: IDEF0 дозволяє розбивати складні системи на менші, більш керовані частини. Кожна функція на вищому рівні може бути деталізована на більш дрібних рівнях, забезпечуючи глибоке розуміння процесів та їх взаємозв’язків. IDEF0 є стандартизованою методологією, яка дозволяє

консистентно описувати широкий спектр систем, від програмного забезпечення до виробничих процесів.

Методологія широко використовується для аналізу процесів, удосконалення систем, розробки бізнес-процедур і тренінгу. Вона ефективна для забезпечення ясності та однозначності у проєктах, де потрібне глибоке розуміння взаємодій між різними частинами системи.


Рисунок 3.1 – Контекстна діаграма IDEF0

*Джерело*: побудовано автором (знімок з екрану)

Функція A0: Управління мережевими конфігураціями визначає основну функціональну область вашої системи, яка займається управлінням змін у мережевих налаштуваннях. Ця функція включає в себе збір запитів та вимог користувачів, а також відповідне налаштування мережевих параметрів за допомогою автоматизованих інструментів.

Компоненти-входи є наступними:

– запити користувачів: це можуть бути запити на зміну конфігурацій, додавання нових компонентів мережі, або зміни в політиках безпеки;

– параметри конфігурації: деталізовані технічні дані, які необхідні для виконання змін у мережі, включаючи IP-адреси, правила маршрутизації, налаштування безпеки тощо.

Компоненти-виходи є такими:

– оновлена конфігурація: кінцевий результат відображає зміни, що були успішно застосовані до мережевих налаштувань;

– звіти про впровадження конфігурація: документи чи сповіщення, що містять інформацію про успішність виконання змін, виявлені проблеми.

Компоненти-механізми наступні:

– сервер автоматизації: цнтральний компонент, що керує виконанням усіх автоматизованих процесів управління конфігураціями;

– інструменти автоматизації: програмне забезпечення або скрипти, такі як Ansible, Terraform чи інші, що використовуються для автоматизованого налаштування мережі.

– користувач: особа або команда, яка використовує систему.

– база даних: зберігає дані про конфігурації та історію змін.

Компоненти-керування наступні:

– стандарти та протоколи мережі: регулятивні вимоги та стандарти, які визначають правила конфігурації;

– скрипти для автоматизації: програмні скрипти, що автоматизують рутинні задачі та забезпечують впровадження змін.

Ці правила допомагають забезпечити відповідність роботи системи загальноприйнятим стандартам і політикам безпеки.

Декомпозиція IDEF0-діаграми представлена на рисунках 3.2.

Дані для діаграми наступні:

– Компоненти-входи: запити користувачів, параметри конфігурації:

– Компоненти-виходи: оновлена конфігурація звіти про впровадження конфігурація;

– Компоненти-механізми: сервер автоматизації; інструменти автоматизації: користувач: база даних.

– Компоненти-керування: стандарти та протоколи мережі; скрипти для автоматизації.

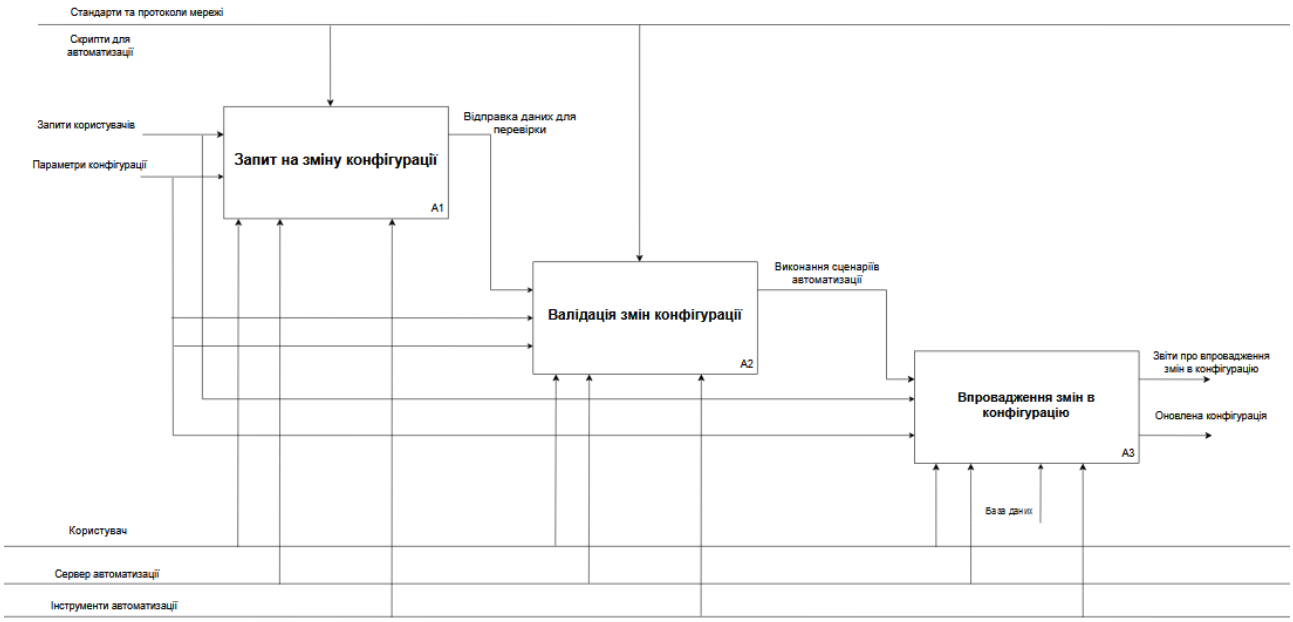



Рисунок 3.2 – Декомпозиція IDEF0-діаграми

*Джерело*: побудовано автором (знімок з екрану)

Функція A1: управління сервісами файрволу. Функція обробляє початкові запити користувачів та параметри конфігурації, ініціюючи процес автоматизації. Вона готує дані для подальшої валідації, підготовляючи їх до перевірки та впровадження.

Функція A2: валідація змін конфігурації. Служить для перевірки відправлених даних і валідації змін перед їх реалізацією. Забезпечує точність і правильність виконання запланованих змін у конфігурації, запобігаючи помилкам у мережевих налаштуваннях.

Функція A3: впровадження змін в конфігурацію. Реалізує затверджені і валідовані зміни у конфігурацію системи. Вона займається активним

впровадженням оновлень та налаштувань, а також створює документацію про зміни та звіти про їх впровадження.

## 3.2 Моделювання варіантів використання

Моделювання варіантів використання (Use Case Modeling) [28] є критичним аспектом аналізу вимог у розробці програмного забезпечення, який допомагає зрозуміти функції системи та взаємодії між різними користувачами (акторами) та системою. Використання діаграм варіантів використання дозволяє ілюструвати, як кінцеві користувачі взаємодіють з системою, визначаючи ключові дії, які можуть виконувати користувачі, та очікувані відповіді системи.
Основні елементи діаграми варіантів використання

Актори (Actors): представляють користувачів або інші системи, які взаємодіють із системою. Актори можуть бути зовнішніми (наприклад, користувачі, зовнішні системи) або внутрішніми (наприклад, компоненти системи).

Варіанти використання (Use Cases): описують послідовності подій, які виконуються системою для досягнення конкретних цілей користувача. Кожен варіант використання представляє завершену функцію або процес.

Зв’язки (Relationships) є настпуними:

- Асоціації (Associations): показують взаємодію між актором і варіантом використання;
- Залежності (Dependencies): описують, як зміни в одному варіанті використання можуть вплинути на інший;
- Узагальнення (Generalizations): використовуються для групування та узагальнення подібних варіантів використання.

Діаграма варіантів використання зображена на рисунку 3.3.

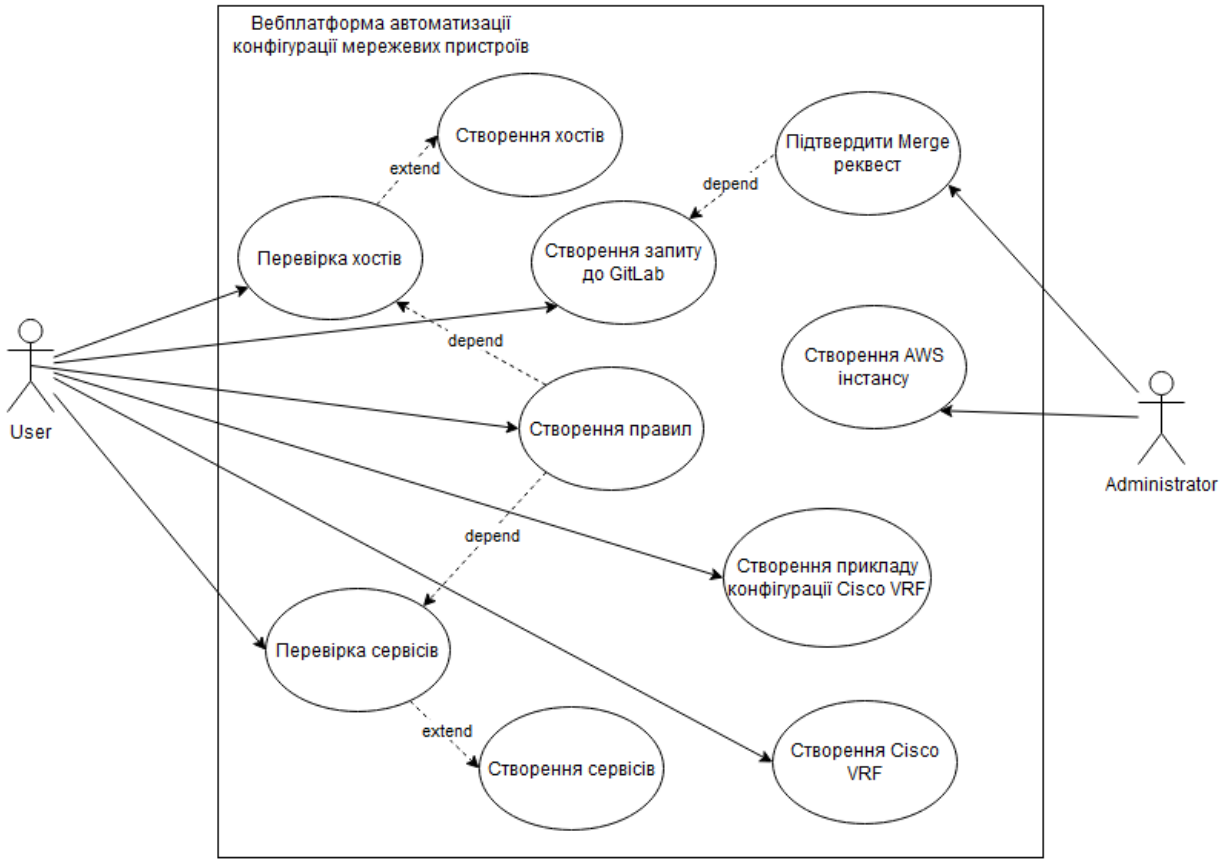



Рисунок 3.3 – Діаграма варіантів використання вебплатформи

*Джерело*: побудовано автором (знімок з екрану)

Для системи автоматизації конфігурації мережевих пристроїв та сервісів було розглянуто наступні ключові варіанти використання:

- для Користувача:
  - перевірка хостів та сервісів;
  - створення сервісів та хостів;
  - створення правил, які залежать від результатів перевірок;
  - створення запитів до GitLab;
  - конфігурація Cisco VRF.
- для Адміністратора:
  - створення інстансів на AWS;
  - підтвердження мердж реквестів у GitLab.

Нижче наведено докладний опис варіантів використання для акторів «Користувач» та «Адміністратор».

Користувач: зазвичай кінцевий користувач системи, який виконує дії з перевірки та створення конфігурацій у мережевій інфраструктурі.

Адміністратор: відповідає за більш складні та критичні операції, такі як управління хмарними ресурсами та затвердження змін.

Користувач системи має можливість перевіряти стан існуючих хостів, що включає запит інформації про їхню доступність та конфігурацію. Ця інформація допомагає користувачеві визначити, які хости доступні для подальших дій. Якщо потрібно, користувач може ініціювати створення нових хостів, введення параметрів для їх конфігурації, після чого система проведе валідацію та створення відповідних хостів.

Аналогічний процес відбувається і з мережевими сервісами. Користувач може перевіряти існуючі сервіси та, за потреби, створювати нові. Система дозволяє користувачеві вводити специфікації для нових сервісів, валідує ці дані, та створює сервіси за визначеними параметрами.

Додатково, користувач має можливість створювати мережеві правила на основі валідованих хостів та сервісів. Цей процес включає вибір компонентів, які будуть інтегровані в нове правило, після чого система застосовує вибрані налаштування.

Користувач також може ініціювати створення merge реквестів у GitLab для реалізації змін у конфігурації системи, де адміністратор згодом може перевірити та затвердити ці запити.

Для роботи з обладнанням Cisco, користувачі мають можливість генерувати та застосовувати конфігурації для віртуальних маршрутизаторів (VRF) [29], що дозволяє їм керувати поділом мережі на сегменти залежно від вимог безпеки та ефективності.

Адміністратор відповідає за створення інстансів у хмарному сервісі AWS [30], де він ініціювати їх створення. Система валідує введені дані та керує процесом їх реалізації.

Крім того, адміністратор має повноваження переглядати та затверджувати мердж реквести в GitLab, які ініційовані користувачами. Цей процес включає перевірку запитів на зміни в конфігурації та їх затвердження для впровадження.

## 3.3 Проєктування моделі бази даних

Система використовує MongoDB [31] для зберігання критичних даних, пов’язаних з управлінням правилами файрволів. Це документо-орієнтована база даних NoSQL, яка відома своєю високою продуктивністю, високою доступністю та легкою масштабованістю. Вона була розроблена у 2009 році компанією MongoDB Inc. та з тих пір стала однією з найпопулярніших баз даних для сучасних застосунків. MongoDB використовує формат BSON (Binary JSON) для зберігання даних, що дозволяє їй ефективно обробляти великі обсяги даних та складні запити.

MongoDB зберігає дані у вигляді документів, що схожі на JSON-об’єкти. Кожен документ складається з пар ключ-значення і може містити вкладені документи та масиви. Ця структура дозволяє зберігати складні ієрархічні відносини у одному документі, що робить їх легшими для читання та обробки. На відміну від реляційних баз даних, вона не вимагає визначення схеми перед зберіганням даних. Структура документів може відрізнятися, дозволяючи додавати нові поля або видаляти існуючі без необхідності зміни всієї схеми бази даних. MongoDB використовує реплікацію для забезпечення відмовостійкості та високої доступності. Реплікаційні набори в MongoDB включають один

первинний вузол, який обробляє усі операції з читання та запису, та кілька вторинних вузлів, які виконують копіювання даних з первинного вузла.

MongoDB була обрана для проєкту з автоматизації конфігурацій мережевих пристроїв і сервісів завдяки її винятковій гнучкості і масштабованості. Ця документо-орієнтована база даних не вимагає жорсткої схеми, що дозволяє легко адаптуватися до змін у мережевих конфігураціях без потреби в перепроєктуванні бази даних. MongoDB вражає своєю швидкістю і продуктивністю при роботі з великими обсягами даних, що є критично важливим для систем, які вимагають високого рівня відгуку та оперативності.

Завдяки своїм механізмам реплікації, MongoDB забезпечує високу відмовостійкість, що є необхідним для забезпечення неперервності бізнес-процесів, особливо у критичних застосунках. Масштабованість через шардинг дозволяє розподіляти навантаження по кількох серверах, що значно підвищує загальну продуктивність системи. MongoDB також підтримує широкий спектр типів даних, включаючи вкладені документи і масиви, що ідеально підходить для управління складними мережевими налаштуваннями. Ця база даних легко інтегрується з більшістю сучасних технологічних стеків, забезпечуючи гнучкість та легкість управління для розробників і системних адміністраторів.

MongoDB має унікальну, гнучку структуру (рис. 3.4) для зберігання та управління даними, яка відрізняється від традиційних реляційних баз даних. Основні складові структури MongoDB розкладаються на кілька рівнів ієрархії: бази даних, колекції, документи та поля. Детальний опис кожного рівня наступний:

– база даних у MongoDB функціонує як контейнер для колекцій і може містити декілька колекцій. Кожна база даних є окремим фізичним набором файлів на диску і ізольована від інших баз даних; у рамках одного сервера MongoDB може бути створено багато таких баз, кожна з яких має власний набір прав доступу;

– колекція в MongoDB еквівалентна таблиці у реляційній базі даних, але вона не має жорстко визначеної схеми. Колекція містить один або більше документів; документи в одній колекції можуть мати різні поля, що надає велику гнучкість при проєктуванні бази даних;

– документ є базовою одиницею даних в MongoDB і аналогічний рядку в таблиці реляційної бази даних; усі дані в документі зберігаються у форматі BSON, що є бінарною формою JSON. Документи в колекції можуть мати різні поля, набори полів і структури даних;

– поля в документах схожі на колонки в таблицях реляційних баз даних. Кожне поле має ключ (або назву) і значення, яке може бути будь-яким типом даних, що підтримується MongoDB, включаючи інші документи, масиви та вкладені документи; це дозволяє зберігати складні структури даних всередині одного документа;

– для покращення продуктивності запитів, MongoDB дозволяє створювати індекси на одному або кількох полях у документах; індекси покращують швидкість доступу до даних за вказаними критеріями пошуку.

Ця структура дозволяє MongoDB бути надзвичайно гнучкою і динамічною системою управління базами даних, здатною задовольняти різноманітні потреби сучасних додатків. MongoDB ефективно справляється з великими обсягами неструктурованих даних і комплексними запитами, що робить її популярним вибором для багатьох розробників та компаній.

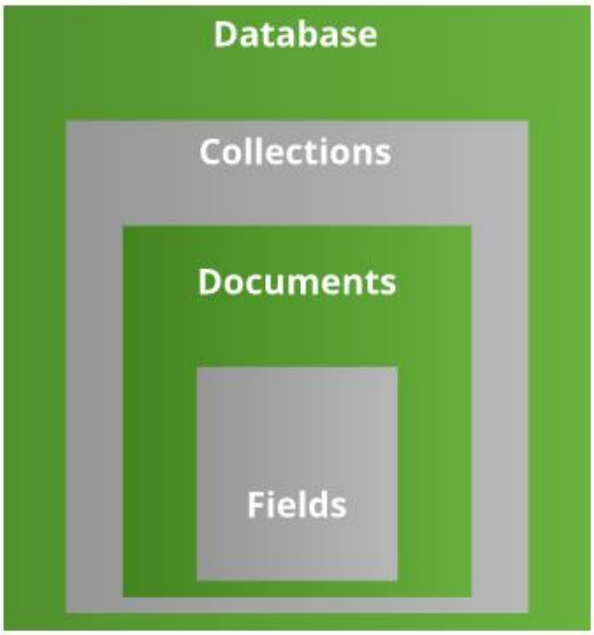



Рисунок 3.4 – Структура mongodb

*Джерело*: побудовано автором (знімок з екрану)

Дані зберігаються у колекції records в базі даних mydatabase. Кожен запис у цій колекції представляє окреме правило файрволу з наступними атрибутами:

- _id: унікальний ідентифікатор запису, що використовується для внутрішнього відслідковування і посилання в базі даних;
- src_ip: IP-адреса джерела трафіку;
- src_uid: унікальний ідентифікатор об’єкта джерела;
- src_type: тип об’єкта джерела (наприклад, ’host’);
- dst_ip: IP-адреса призначення трафіку;
- dst_uid: унікальний ідентифікатор об’єкта призначення;
- dst_type: тип об’єкта призначення (наприклад, ’host’);
- port: назва порту чи служби, що використовується (наприклад, ’service-udp_udp_555’);
- service_uid: унікальний ідентифікатор сервісу;
- created_at: час створення правила, який зберігається у форматі ISODate.

Ця структура дозволяє системі швидко звертатися до даних для виконання запитів на перевірку або зміну правил файрволу. Операції читання, вставки, оновлення та видалення (CRUD) виконуються через API, розроблені на основі Flask, що спілкуються з MongoDB. Автоматизація взаємодії з базою даних здійснюється за допомогою інструментів управління, як-от Ansible, які використовуються для надсилання та виконання налаштувань на мережевому обладнанні.

Використання MongoDB як сховища для правил файрволу надає системі велику гнучкість і швидкість при роботі з динамічними даними. Це дозволяє системі ефективно адаптуватися до змін в мережевій конфігурації та забезпечує високий рівень безпеки завдяки точному контролю доступу до мережевих ресурсів.

Ця модель зберігання даних сприяє забезпеченню прозорості та ефективності в управлінні мережевими конфігураціями, що є критично важливим для сучасних мережевих середовищ. На рисунку 3.5 показано приклад створеного запису в базі даних.

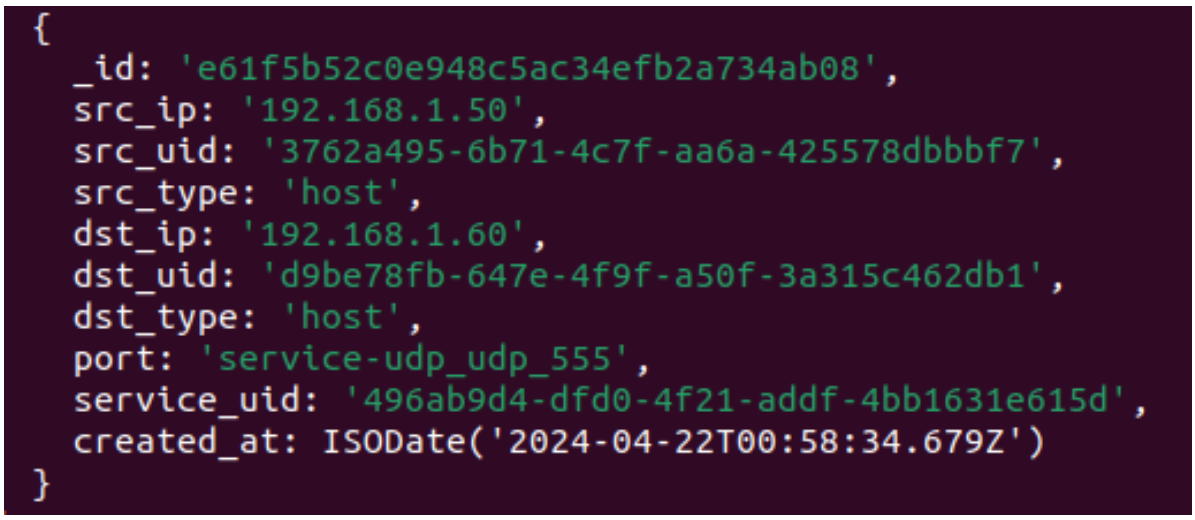



Рисунок 3.5 – Приклад запису в базі даних

*Джерело*: побудовано автором (знімок з екрану)

# 4 РОЗРОБКА ВЕБПЛАТФОРМИ АВТОМАТИЗАЦІЇ КОНФІГУРАЦІЇ МЕРЕЖЕВИХ ПРИСТРОЇВ ТА СЕРВІСІВ

## 4.1 Архітектура вебплатформи

Розробка інформаційної технології автоматизації конфігурації мережевих пристроїв та сервісів у вигляді вебплатформи вимагає інтеграції низки сучасних технологій. Це забезпечують високу ефективність, гнучкість і надійність системи. Вибір технологій для цього проєкту обумовлений потребою забезпечення зручного інтерфейсу для клієнтів, масштабованості архітектури та здатності інтегруватися з різноманітними мережевими сервісами й обладнанням.

Основа користувацького інтерфейсу платформи будується на HTML і JavaScript. Перший створює структурний каркас вебсторінок, визначаючи розміщення тексту, форм вводу, кнопок, і зображень. Це забезпечує основні візуальні елементи, які є необхідними для будь-якої вебсторінки.

JavaScript відіграє роль в динамізації цих сторінок, дозволяючи користувачам взаємодіяти з вебплатформою в режимі реального часу. Завдяки JavaScript, власна розробка може виконувати асинхронні запити до сервера, обновляючи інформацію на сторінці без необхідності її перезавантаження. Це важливо для забезпечення плавного та швидкого користувацького досвіду.

На бекенді основою служить Python із використанням мікро-фреймворка Flask [32]. Python було обрано через свою широку підтримку, чистоту синтаксису та ефективність у написанні серверного коду. Flask дозволяє обробляти запити HTTP від користувачів, маршрутизувати ці запити до відповідних обробників, а також відправляти відповіді назад до споживача. Він також надає розробникам гнучкість у керуванні сесіями клієнтів, безпеці та інтеграції з іншими вебсервісами та базами даних. Легкість Flask та наявність здатності швидко

розгортати прототипи робить його ідеальним для стартапів та проєктів із швидкими ітераціями.

Фронтенд і бекенд інтегруються через RESTful API [33]. Вони створені за допомогою Flask. Ці API служать мостом, по якому передаються дані між користувацьким інтерфейсом і сервером у форматі JSON. Це дозволяє системі ефективно взаємодіяти зі споживачами, забезпечуючи їх актуальною інформацією та можливістю керування конфігураціями в реальному часі.

Завдяки застосуванню цих технологій, платформа може забезпечувати високу продуктивність, швидкість реагування на запити користувачів та високий рівень персоналізації для кожного користувача. Це робить її не тільки технічно справною, але й адаптивною до потреб клієнтів. Це є ключовим для успішного впровадження та використання у великих організаціях. High-level design архітектура представлена схемою на рисунку 4.1.

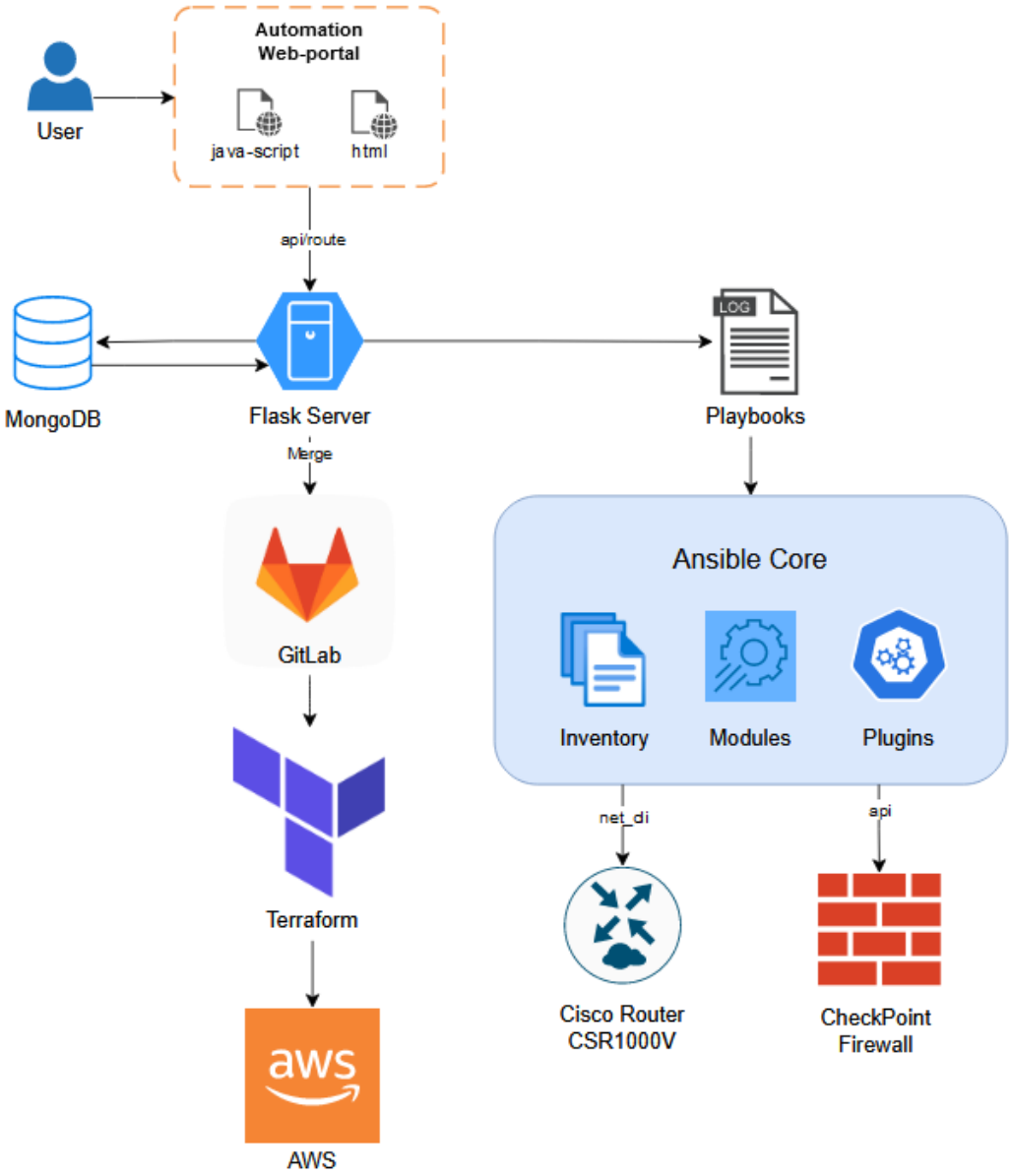



Рисунок 4.1 – HLD архітектура вебплатформи

*Джерело*: побудовано автором (знімок з екрану)

Діаграма послідовностей UML (рис. 4.2) ілюструє взаємодії, пов’язані з автоматизацією налаштувань мережевих пристроїв через дану вебплатформу. Вона детально показує інтеракцію між користувачем, вебінтерфейсом, сервером, базою даних та мережевими пристроями. Кожен етап цієї взаємодії послідовно відображений на діаграмі. Від введення даних користувачем через вебінтерфейс до обробки сервером та застосування конфігурацій до мережевих пристроїв.

Діаграма включає синхронні та асинхронні повідомлення для демонстрації обробки даних у реальному часі та прийняття рішень і з анотаціями для пояснення складних взаємодій або значущих точок у потоці.

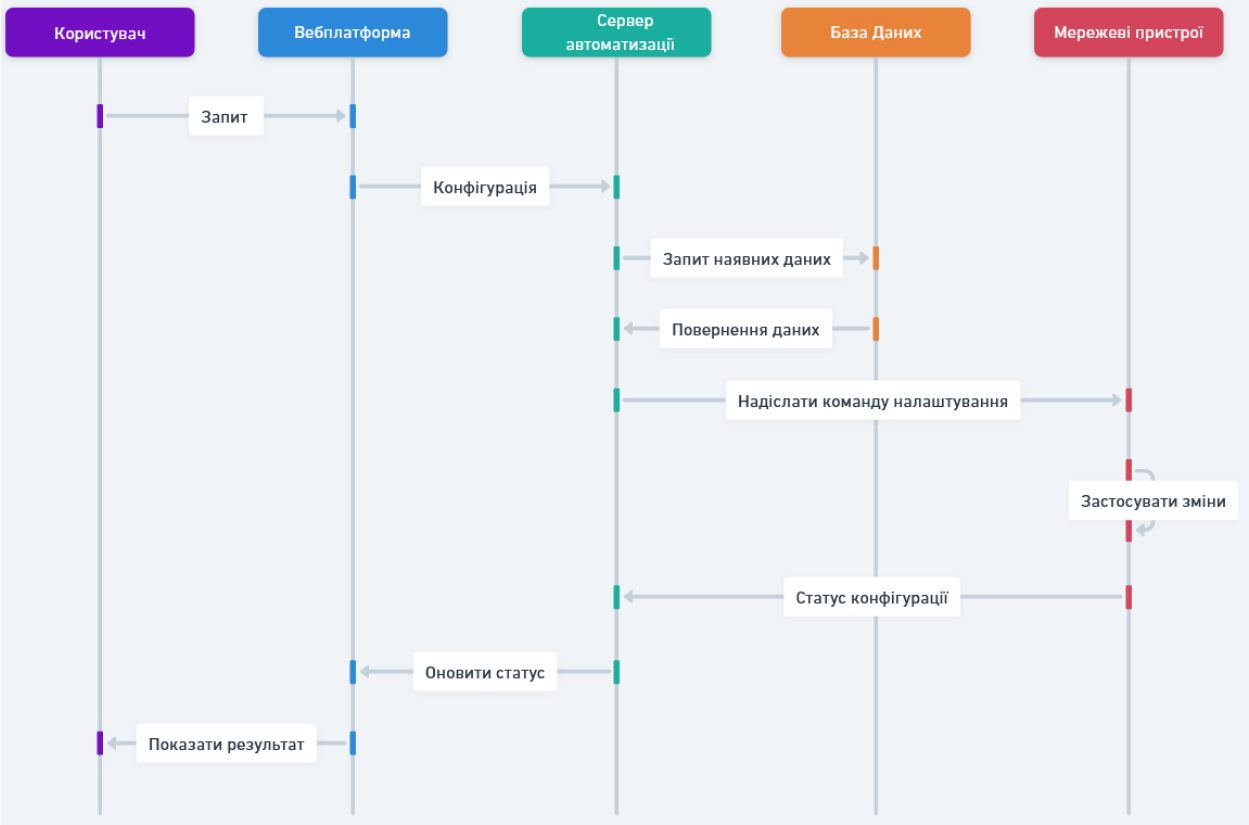



Рисунок 4.2 – Діаграма послідовностей UML

*Джерело*: побудовано автором (знімок з екрану)

## 4.2 Реалізація вебплатформи

Дана платформа забезпечує користувацький інтерфейс із навігаційною панеллю, дозволяючи легко вибирати потрібні функції. Це надає клієнтам можливість легко переходити між останніми. Таким чином забезпечується інтуїтивне та зручне управління мережевими конфігураціями. Інтерфейс

підтримує високий рівень доступності та зрозумілості, спрощуючи користувачам процес вибору та застосування необхідних інструментів.

Кожна функція на цій платформі включає форму, де користувачі вводять параметри, такі як IP-адреси, порти та обирають типи протоколів через радіокнопки. Нижче наведено приклад HTML-коду для функції перевірки та створення правил файрволу:

```
    <div class="form-container">
      <form id="ruleForm">
        <div class="input-group">
          <label for="src-ip">Source IP:</label>
          <input type="text" id="src-ip" name="src-ip" required>
          <div class="radio-group">
            <input   type="radio"   id="src-host"   name="src-type"
value="host" checked>
            <label for="src-host">Host</label>
            <input  type="radio"  id="src-network"  name="src-type"
value="network">
            <label for="src-network">Network</label>
            <input   type="radio"   id="src-group"   name="src-type"
value="group">
            <label for="src-group">Group</label>
          </div>
        </div>
        <button type="submit">Check</button>
        <button   type="button"   id="runPlaybook"   disabled>Create
Rule</button> <!-- New button for running playbook -->
      </form>
```

JavaScript використовується для взаємодії з сервером за допомогою методу fetch [34]. Це дозволяє відправляти асинхронні запити до бекенду без необхідності перезавантажувати сторінку. Коли користувач заповнює форму та надсилає її, скрипт зчитує введені дані, такі як IP-адреси, порт та протокол, і формує JSON-об’єкт, який відправляється на серверний кінець за визначеною URL-адресою. Метод POST [35] застосовується для відправлення даних на сервер, де обробляється запит і повертається відповідь, яку можна обробити у подальшому. Наприклад, для виведення повідомлення про статус або для збереження ідентифікатора в місцевому сховищі. Нижче наведено приклад js-коду для функції перевірки та створення правил файрволу:

```
        fetch('http://127.0.0.1:5000/api/run_playbook', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ src_ip: srcIp, dst_ip: dstIp, port:
port, protocol: protocol })
        })
           .then(response => response.json())
        .then(data => {
          if (data.status === 'success') {
            alert(data.message);
            localStorage.setItem('lastRecordId',
data.db_record._id);
                document.getElementById('runPlaybook').disabled    =
false;
          } else {
            alert(data.message);
                document.getElementById('runPlaybook').disabled    =
true;
          }
        })
```

Flask використовується як бекенд для вебплатформи з автоматизації конфігурації мережевих пристроїв. Цей мікрофреймворк надає функціонал для застосування REST API для обробки HTTP запитів від клієнта. Завдяки Flask, створено ендпойнти для прийому даних від користувачів, валідації цих даних та виконання необхідних скриптів автоматизації. Фреймворк забезпечує просту інтеграцію з базами даних та іншими вебсервісами, спрощуючи створення складних вебдодатків з різноманітними бізнес-логіками.

Функція «run_playbook» обробляє POST-запити на сервері Flask і виконує роль обробника запитів для запуску Ansible плейбуків. Вона парсить вхідні JSON дані для отримання IP-адрес, портів, і типів протоколів від користувача. Ця інформація застосовується для формування параметрів, які передаються в Ansible плейбуки для знаходження об'єктів в мережі та сервісів за допомогою їх IP і портів. Команди Ansible повертають унікальні ідентифікатори об'єктів, які перевіряються на існування. Результати останніх і дані використовуються для створення запису в базі даних, який включає ідентифікаційний номер, IP-адреси, типи джерел, порти, і таймстемп. У залежності від того, чи були знайдені всі

елементи, відповідь містить статус успіху або помилки. Нижче наведено приклад python коду для функції перевірки правил файрволу:

```
@app.route('/api/run_playbook', methods=['POST'])
     def run_playbook():
       data = request.json
       src_ip = data['src_ip']
       dst_ip = data['dst_ip']
       protocol_type = data['protocol'] # Assuming the protocol
(TCP/UDP) is also sent
       port_name = f"{protocol_type}_{data['port']}"
       #port_name = data['port']
       port_type = f"service-{data['protocol']}" if data['protocol']
in ['tcp', 'udp'] else data['protocol']
       source_uid                                                =
parse_ansible_output(run_ansible_playbook('FindObject_Playbook.yml
', f"filter_ip={src_ip}"))
       destination_uid                                           =
parse_ansible_output(run_ansible_playbook('FindObject_Playbook.yml
', f"filter_ip={dst_ip}"))
       service_uid                                               =
parse_ansible_output(run_ansible_playbook('FindService_Playbook.ym
l', f"filter_port={port_name} filter_type={port_type}"))
       all_exist = source_uid and destination_uid and service_uid #
Only true if all are found
```

Для парсингу JSON даних використовується наступна функція:

```
def parse_ansible_output(output):
  """Parse the output from Ansible to extract the UID."""
  pattern                                                       =
r'\{\s*"objects_result\.ansible_facts\.objects\.objects":\s*\[\s*"
([\w-]+)"\s*\]\s*\}'
  match = re.search(pattern, output)
  return match.group(1) if match else None
```

Інтеграція з MongoDB здійснюється через створення з'єднання з базою даних за допомогою бібліотеки pymongo [36]. Спочатку налаштовується зв'язок із сервером MongoDB. При цьому вказу'ться рядок з'єднання, який містить адресу сервера і порт. Далі, обирається конкретна база даних (mydatabase), у якій створюються або використовуються колекції records та users для зберігання даних. Колекція records застосовується для зберігання записів про операції в системі, в той час як users може зберігати інформацію про користувачів системи.

Це дозволяє ефективно зберігати, оновлювати та витягувати дані залежно від потреб проєкту. Нижче наведено код для підключення бази даних:

```
    client = MongoClient('mongodb://localhost:27017/') # Adjust
connection string if needed
    db = client['mydatabase'] # Use or create a database
    records = db.records # Use or create a collection
    users = db.users
```

Нижче наведено код для створення запису в базі даних:

```
  # Record creation with a unique identifier and timestamp
  record = {
    #'_id': str(uuid.uuid4()), # Generate a unique identifier
    '_id': str(uuid.uuid4()).replace('-', ''),
    'src_ip': src_ip,
    'src_uid': source_uid,
    'src_type': data.get('src_type', 'host'), # Default to 'host' if
not specified
    'dst_ip': dst_ip,
    'dst_uid': destination_uid,
    'dst_type': data.get('dst_type', 'host'), # Default to 'host'
    'port': f"{port_type}_{port_name}",
    'service_uid': service_uid,
    'created_at': datetime.now() # Timestamp of creation
  }
```

Створення плейбуків для автоматизації конфігурації мережі здійснюється через використання YAML-файлів. Плейбук «InstallRule_Playbook.yml» призначений для додавання правил доступу. Він налаштований на роботу з певним хостом через httpapi [37]. Плейбук включає змінні, які передаються під час виконання, що визначають об'єкти джерела, призначення та сервісу. Він містить задачі для додавання правила в систему управління, визначаючи параметри, такі як позиція, джерело, призначення, сервіс і дія. Ці налаштування автоматизують процеси встановлення правил із високим рівнем контролю та здійснюють автоматичну публікацію сесій. Нижче наведено вміст плейбуку:

```
    ---
    - name: "Add acces rule"
     hosts: smc
     connection: httpapi
     tasks:
      - name: "add rule"
```

```
check_point.mgmt.cp_mgmt_access_rule:
 layer: network
 enabled: true
 name: "{{ object_name }}"
 relative_position:
  top: "Automation"
 source: "{{ object_src }}"
 destination: "{{ object_dst }}"
 service: "{{ object_service }}"
 action: accept
 auto_publish_session: true
 install_on: "CP_Cluster_1"
```

Тестове середовище включає інтеграцію платформи автоматизації з сервером управління [38] та шлюзом безпеки [39], а також окремо з маршрутизатором CSR 1000V [40]. Це дозволяє тестувати роботу скриптів автоматизації у наближених до реальності умовах. З'єднання між платформою та сервером управління безпекою забезпечує управління через шлюз безпеки, тоді як зв'язок із маршрутизатором CSR 1000V дозволяє тестувати мережеві налаштування на цьому роутері. Структура тестового середовища показана на рисунку 4.3.

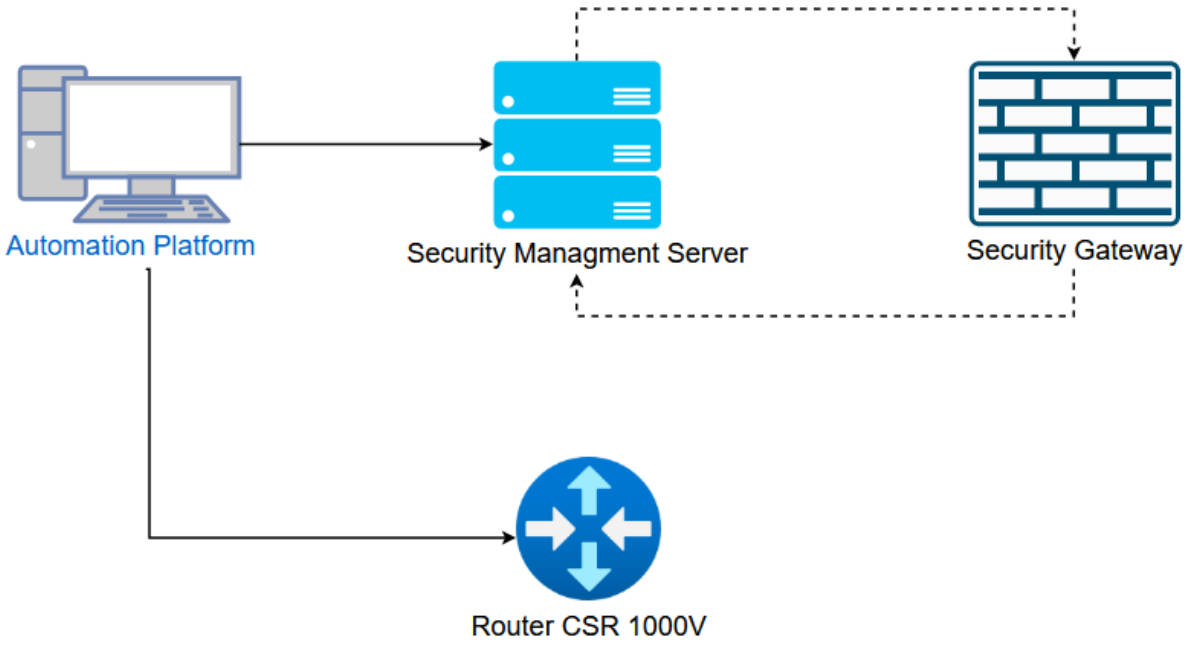



Рисунок 4.3 – Структура тестового середовища

*Джерело*: побудовано автором (знімок з екрану)

Для інтеграції з Amazon Web Services, спочатку потрібно створити аккаунт AWS і встановити основні параметри безпеки. Після цього, у розділі IAM AWS [41], необхідно створити нового користувача та згенерувати ключі доступу (рис. 4.4), які застосовуватимуться для аутентифікації в GitLab. Ці ключі слід додати в налаштування CI/CD GitLab (рис. 4.5), щоб забезпечити безперервну інтеграцію. Початкове налаштування проєкту в GitLab включає створення першого коміту та завантаження його в репозиторій, що є перевіркою правильності налаштувань інтеграції.

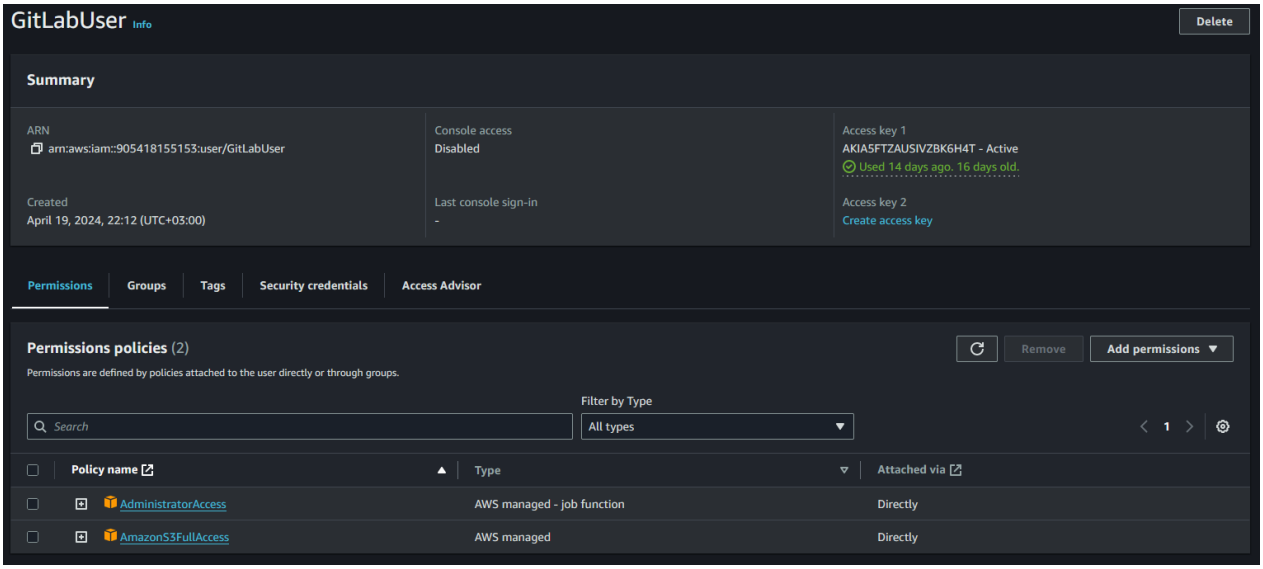



Рисунок 4.4 – Створений AWS користувач

*Джерело*: побудовано автором (знімок з екрану)

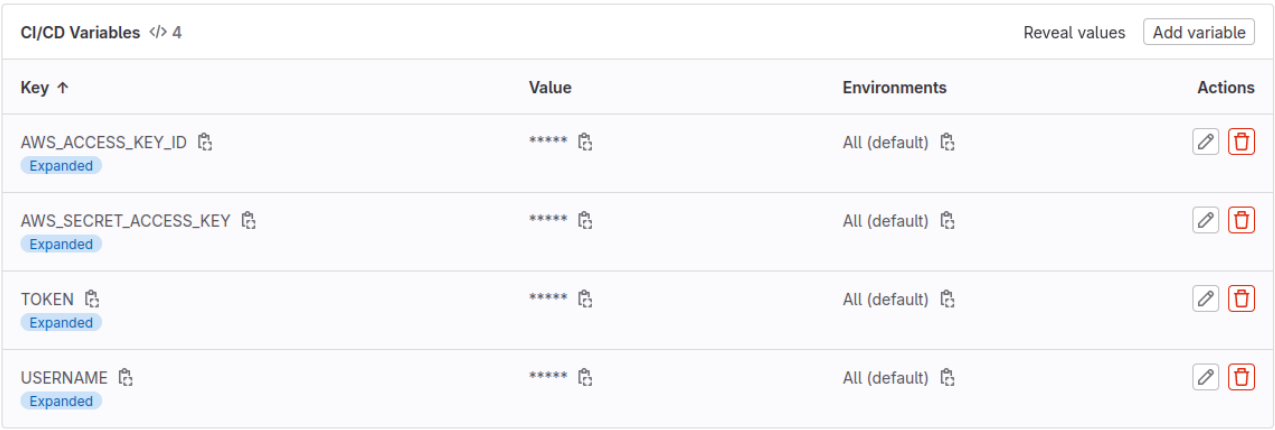



Рисунок 4.5 – Ключі GITLAB CI/CD

*Джерело*: побудовано автором (знімок з екрану)

Файл .gitlab-ci.yml у проєкті налаштовано для управління процесами CI/CD за допомогою Terraform із GitLab [42]. Він включає різні етапи такі, як валідація, планування, застосування та знищення конфігурацій. Кожен етап має свої скрипти для виконання відповідних дій. Наприклад, для ініціалізації Terraform використовуються параметри backend, такі як адреса, ім’я користувача, і пароль, які налаштовані для роботи зі станом у GitLab. Керування версіями здійснюється через різні правила, які дозволяють автоматично запускати пайплайни на головній гілці або при подіях злиття запитів, та вимагають ручного запуску для змін, що впливають на реальні середовища.

Код для виконання стадії валідації показаний нижче:

```
validate:
 stage: validate
 script:
  - terraform validate
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull-push
```

Код для виконання стадії планування показаний нижче:

```
plan:
 stage: plan
 script:
  - terraform plan
 dependencies:
  - validate
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull
```

Код для виконання стадії застосування показаний нижче:

```
apply:
 stage: apply
 script:
  - terraform apply -auto-approve
 rules:
  - if: '$CI_COMMIT_BRANCH == "main"'
    when: manual
```

```
 dependencies:
  - plan
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull
 when: manual
```

Код для виконання стадії знищення конфігурації показаний нижче:

```
destroy:
 stage: destroy
 script:

  - terraform destroy -auto-approve
 rules:
  - if: '$CI_COMMIT_BRANCH == "main"'
   when: manual
 dependencies:
  - plan
  - apply
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull
 when: manual
```

Код Terraform дозволяє автоматично розгорнути віртуальну машину на AWS, надаючи можливість швидкого розгортання та управління віртуальними машинами у хмарі. Така автоматизація значно спрощує процеси управління інфраструктурою. Конфігурацію Terraform, що створює екземпляр EC2 на AWS [43-44]:

```
resource "aws_instance" "Ubuntu_test_instance" {
 ami      = "ami-023adaba598e661ac"
 instance_type = "t2.micro"
}
```

Цей блок коду Terraform визначає ресурс типу aws_instance під назвою Ubuntu_test_instance. Він використовує AMI з ідентифікатором ami-023adaba598e661ac, який є образом віртуальної машини Ubuntu. Обраний тип інстансу, t2.micro, належить до мікро інстансів, що забезпечують мінімальний

рівень обчислювального ресурсу за доступною ціною, ідеально підходять для розробки та тестування з мінімальним навантаженням.

## 4.3 Демонстрація роботи вебплатформи

На головній сторінці вебплатформи (рис. 4.6) користувачам пропонується навігаційна панель. Вона дозволяє легко вибирати потрібні функції для управління конфігураціями. Дана панель дозволяє виконувати різні дії. Наприклад, налаштування мережевих екранів, управління сервісами маршрутизації і інтеграція зі сторонніми системами. Візуальне представлення платформи розроблене так, щоб забезпечити інтуїтивно зрозуміле та зручне користування нею.

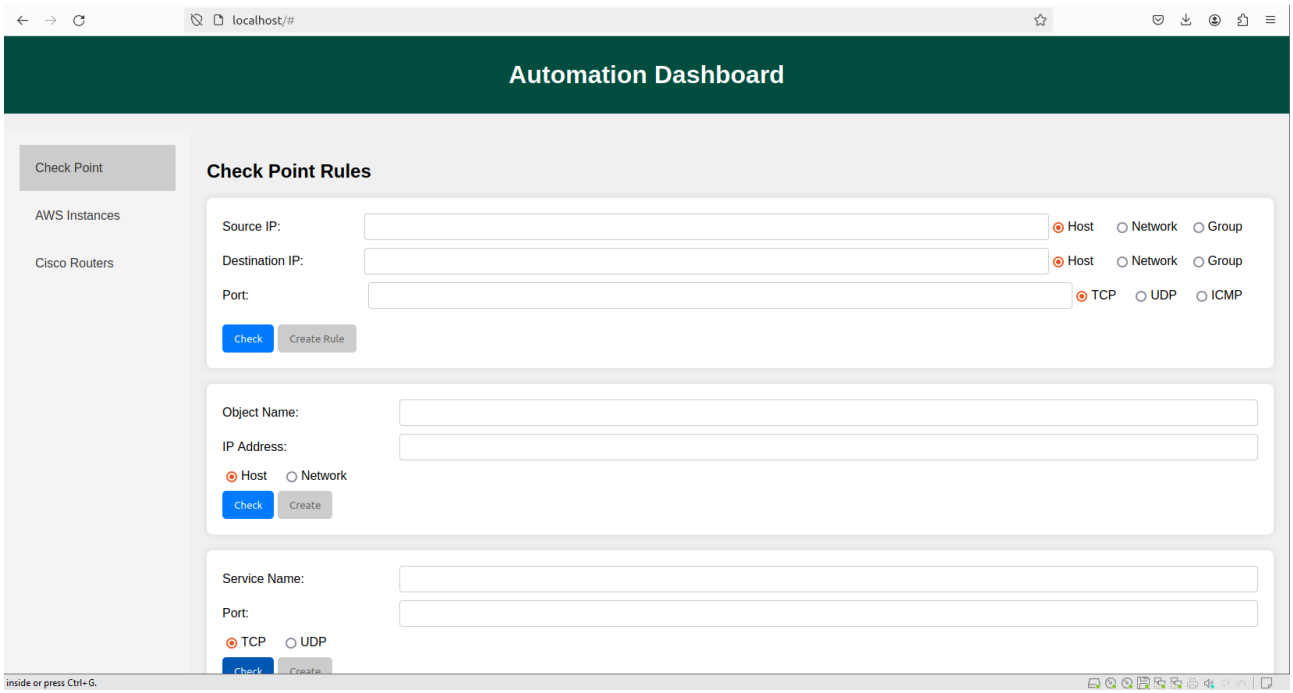



Рисунок 4.6 – Загальний вигляд вебплатформи

*Джерело*: побудовано автором (знімок з екрану)

Функція перевірки хост-об’єкта (рис. 4.7) на файрволі дозволяє клієнтам виконувати запити для виявлення й аналізу хостів у мережевому середовищі. Завдяки інтегрованим інструментам автоматизації на платформі, користувачі можуть із легкістю задавати параметри пошуку, такі як IP-адреси хостів, та отримувати деталізовані відомості про кожен об’єкт. Це значно спрощує процес моніторингу та управління безпекою мережі.

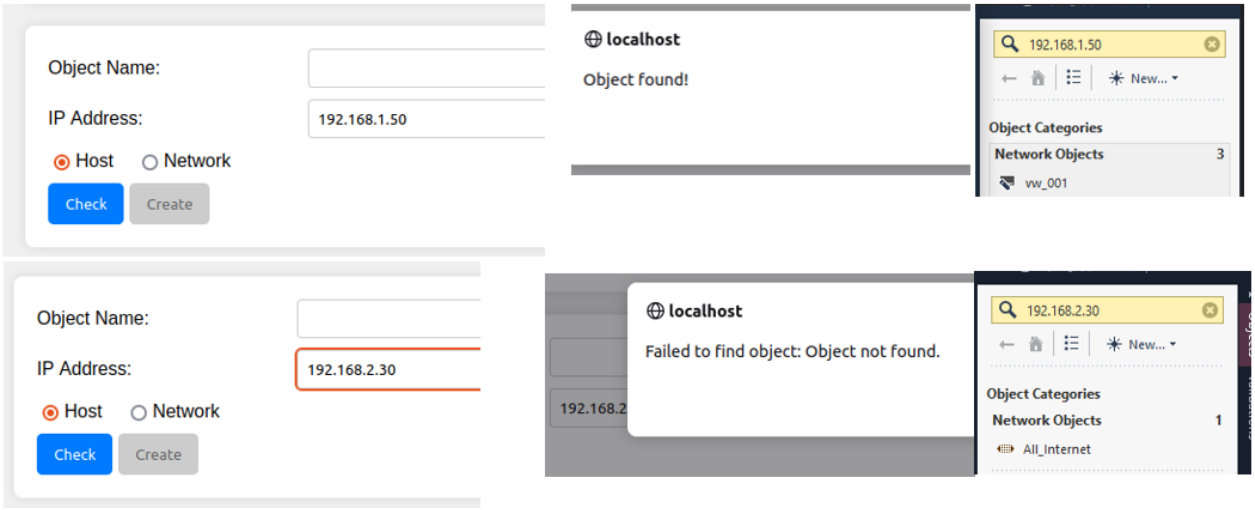



Рисунок 4.7 – Функція перевірки хост-об’єкта

*Джерело*: побудовано автором (знімок з екрану)

Функція перевірки сервіс-об’єктів (рис. 4.8) на файрволі дозволяє ідентифікувати та аналізувати сервіси, такі як порти та протоколи, які використовуються в мережевому середовищі. Цей процес включає запуск заздалегідь сконфігурованих сценаріїв, які дозволяють виявити всі активні сервіси.

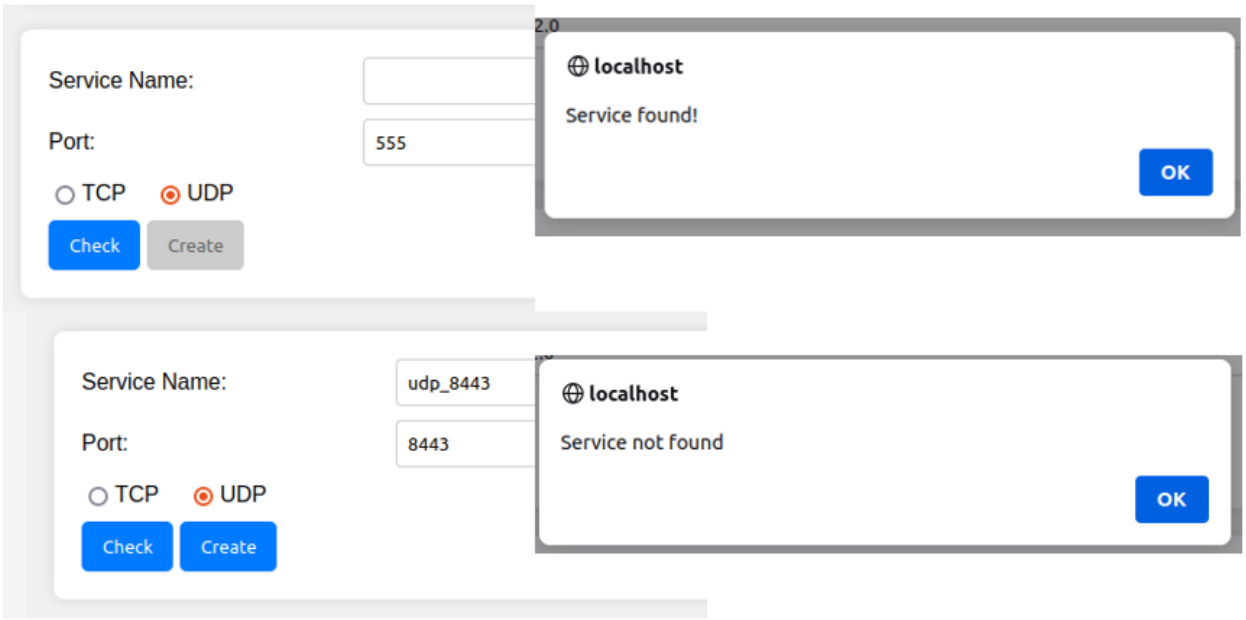



Рисунок 4.8 – Функція перевірки сервіс-об’єкта

*Джерело*: побудовано автором (знімок з екрану)

Функція створення об’єктів (рис. 4.9) активується, коли під час перевірки виявляється, що потрібні об’єкти відсутні у системі. У такому випадку користувач має можливість вручну їх створити, вказуючи їхні унікальні дані. Це дозволяє клієнту налаштувати параметри для кожного об’єкта, що забезпечує необхідну гнучкість у конфігурації мережі та адаптацію до специфічних потреб.

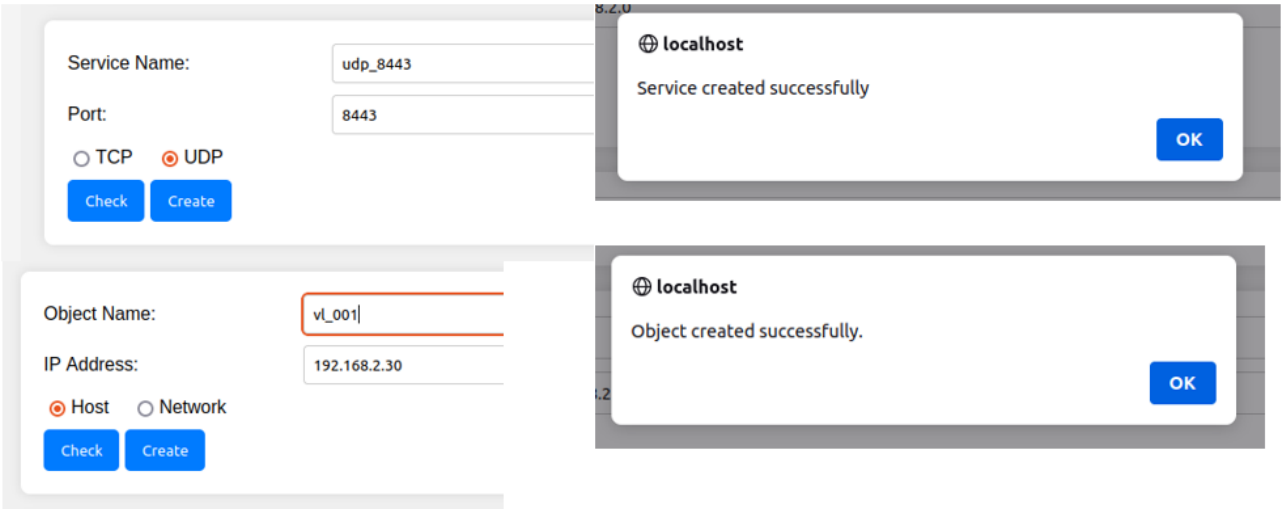



Рисунок 4.9 – Функція створення об’єктів

*Джерело*: побудовано автором (знімок з екрану)

Функція створення правил на файрволі (рис. 4.10) дозволяє користувачу формувати доступні правила після перевірки існування необхідних об’єктів у системі. Цей процес починається з валідації наявності всіх необхідних елементів. Якщо перевірка підтверджує їхню наявність, користувач отримує дозвіл на створення правила. Це забезпечує точність та відповідність конфігурації безпеки актуальним вимогам інфраструктури.

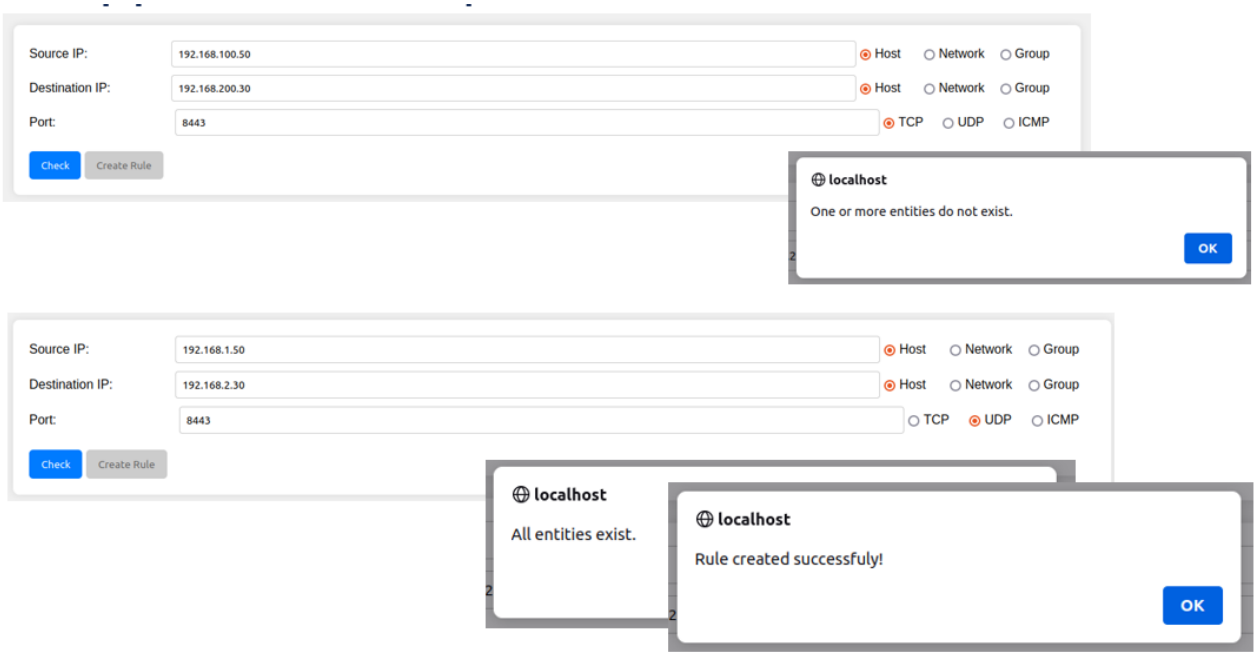



Рисунок 4.10 – Функція створення правил на файрволі

*Джерело*: побудовано автором (знімок з екрану)

На рисунку 4.11 показано усі об’єкти та правила, що були попередньо створені. Ця візуалізація представляє кінцевий вигляд мережевих налаштувань і політик безпеки, які застосовано через автоматизовані процедури.

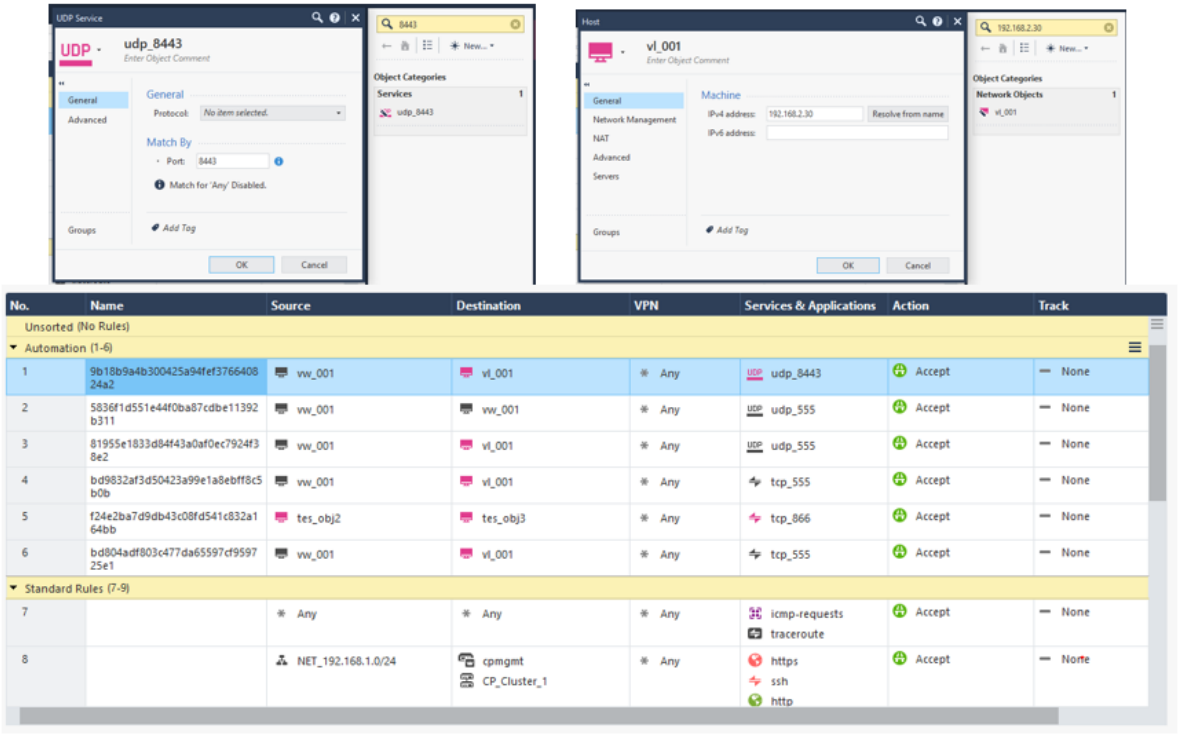



Рисунок 4.11 – Новостворені об’єкта та правила

*Джерело*: побудовано автором (знімок з екрану)

Функція створення нового VRF (рис. 4.12) на роутері дозволяє користувачу задати всі необхідні унікальні параметри для налаштування мережі. Після введення даних, новий VRF створюється, надаючи можливість контролювати трафік у мережі з більшою гнучкістю. Крім того, система пропонує функціонал генерації текстової конфігурації, що дозволяє переглянути потенційні зміни без їх реального застосування на обладнанні. Це забезпечує додаткову перевірку та безпеку перед остаточним впровадженням налаштувань у мережеве середовище. На рисунку 4.13 показано створений VRF на роутері.

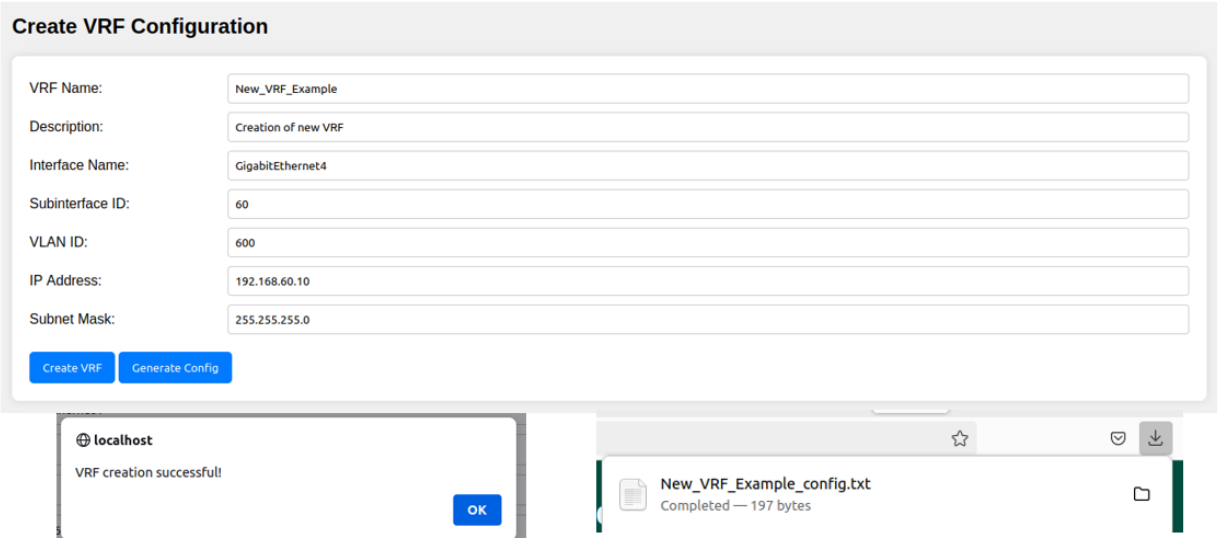



Рисунок 4.12 – Функція створення VRF

*Джерело*: побудовано автором (знімок з екрану)

```
Router_IOS#show vrf br
  Name                             Default RD          Protocols   Interfaces
  NewVRF                           100:1
  New_VRF_Example                  <not set>
  test_vrf                         <not set>           ipv4,ipv6   Gi4
  test_vrf_3                       <not set>
  test_vrf_4                       <not set>
  test_vrf_5                       <not set>
Router_IOS#show ip int br
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       unassigned      YES NVRAM  administratively down down
GigabitEthernet2       192.168.1.10    YES NVRAM  up                    up
GigabitEthernet3       unassigned      YES NVRAM  administratively down down
GigabitEthernet4       192.168.1.31    YES manual administratively down down
GigabitEthernet4.60    192.168.60.10   YES manual administratively down down
GigabitEthernet5       192.168.2.32    YES manual administratively down down
GigabitEthernet5.10    192.168.10.1    YES manual administratively down down
GigabitEthernet5.20    192.168.20.1    YES manual administratively down down
GigabitEthernet5.50    192.168.50.1    YES manual administratively down down
Router_IOS#_
```

Рисунок 4.13 – Новостворений VRF

*Джерело*: побудовано автором (знімок з екрану)

Функція створення AWS інстансу (рис. 4.14) на платформі дозволяє користувачу автоматизувати процес розгортання віртуальних машин. За допомогою одного кліку клієнт може ініціювати створення мердж реквесту на GitLab (рис. 4.15). Після цього, адміністратор переглядає та затверджує зміни перед тим, як запустити інстанс, забезпечуючи контроль і безпеку процесу

розгортання. Ця функція значно спрощує управління ресурсами у хмарному середовищі та забезпечує швидке впровадження необхідних сервісів.

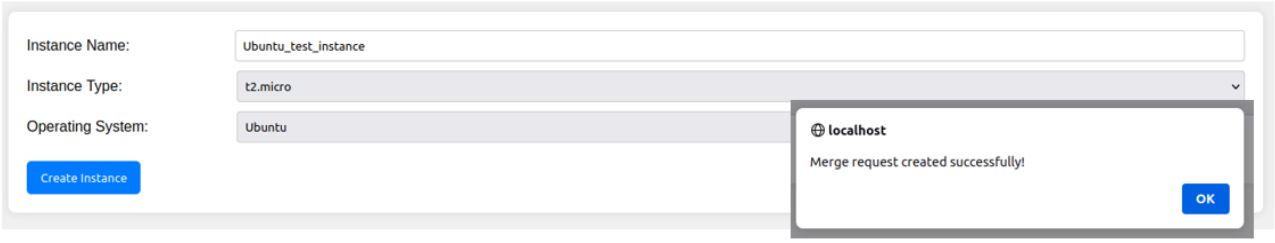



Рисунок 4.14 – Функція створення AWS інстансу

*Джерело*: побудовано автором (знімок з екрану)

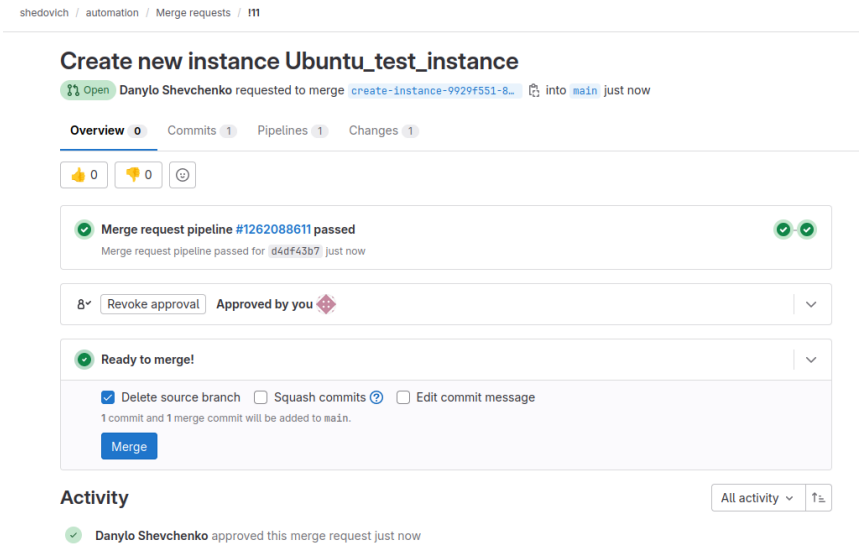



Рисунок 4.15 – Створений мердж реквест

*Джерело*: побудовано автором (знімок з екрану)

Процес створення AWS інстансу включає створення коду Terraform (рис. 4.16), який конфігурує інстанс. Потім адміністратор підтверджує та виконує пайплайни Terraform Plan та Terraform Apply через GitLab для реалізації змін у хмарному середовищі (рис. 4.17). На рисунку 4.17 показано завершений процес на AWS, де показані створені інстанси.

Overview 0 Commits 1 Pipelines 1 **Changes 1**

Compare main and latest version

terraform/Cloud/Ubuntu_test_instance.tf 0 → 100644

```
1  +
2  + resource "aws_instance" "Ubuntu_test_instance" {
3  +   ami           = "ami-023adaba598e661ac"
4  +   instance_type = "t2.micro"
5  + }
```

Рисунок 4.16 – Створений код terrafrom

*Джерело*: побудовано автором (знімок з екрану)

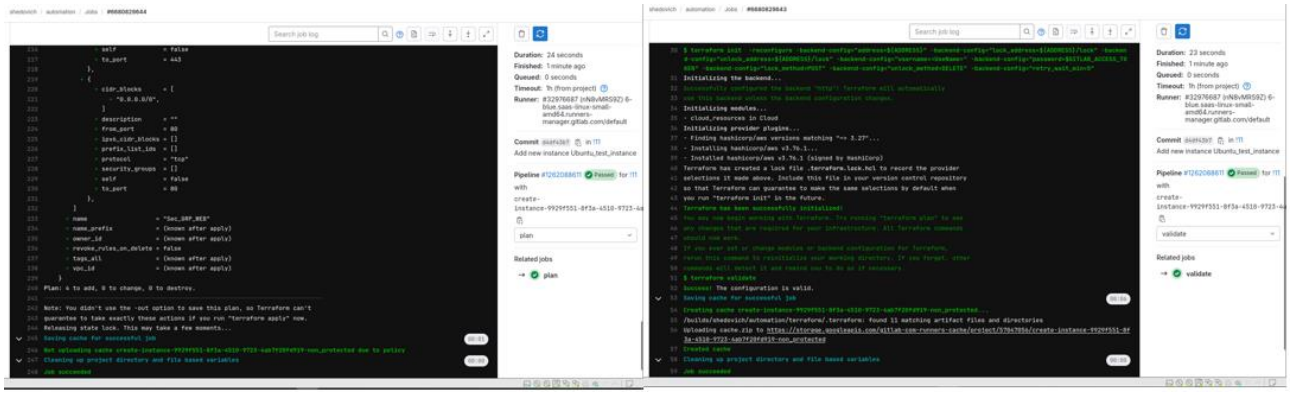


Рисунок 4.17 – Виконаний пайплайн

*Джерело*: побудовано автором (знімок з екрану)

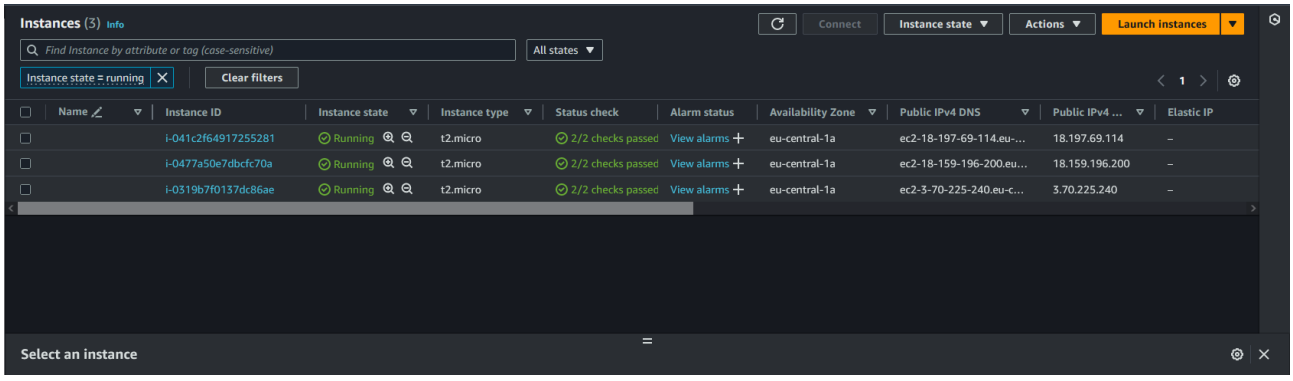


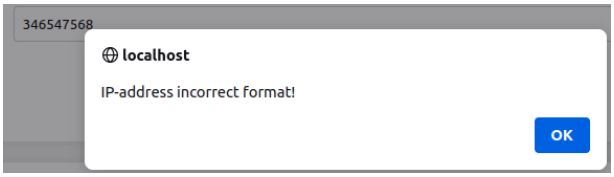
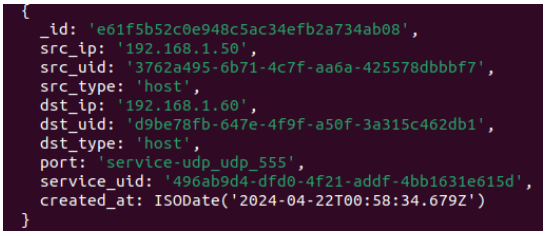

Рисунок 4.18 – Створений AWS інстанс

*Джерело*: побудовано автором (знімок з екрану)

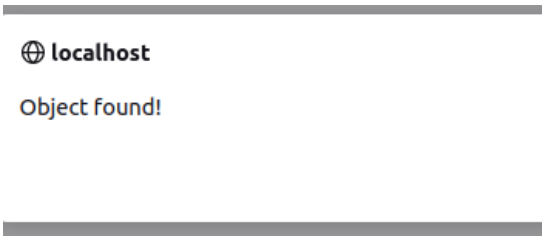
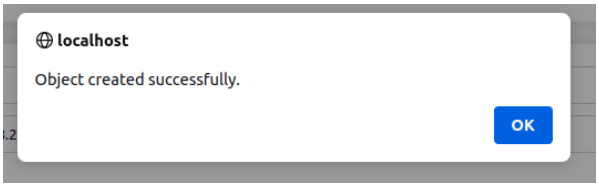
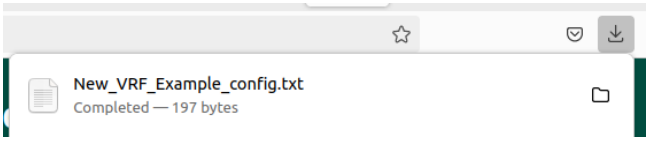
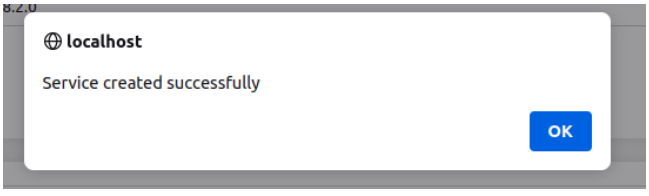
## 4.4 Тестування вебплатформи

Після завершення програмної реалізації вебплатформи, надзвичайно важливим є забезпечення належної якості роботи її функціоналу та відповідності поставленим вимогам. Тому було проведене ретельне тестування власної розробки. Воно включало перевірку коректності роботи функцій вебплатформи, включно з належною реакцією на введення невалідних даних та перевірку інтеграцій з зовнішніми сервісами через API. Використовуючи метод «чорного ящика» [45], було досліджено зовнішню поведінку програмного продукту без доступу до його коду. Основні результати тестування надано в таблиці 4.1.

Таблиця 4.1 – Результати проведеного тестування

| № | Назва | Очікуваний результат | Фактичний результат | 0/1 |
|---|---|---|---|---|
| 1 | Перевірка валідності IP адрес | Система приймає тільки валідні IP | Валідація працює коректно   | 1 |
| 2 | Відправлення даних через API | API передає дані коректно | Дані успішно передаються | 1 |
| 3 | Перевірка з’єднання з базою даних | З’єднання має бути стабільним та запис має створюватись | Запис створено   | 1 |

Продовження табл. 4.1

| | | | | |
|---|---|---|---|---|
| 4 | Тест перевірки об’єктів | Перевірка об’єктів має виконуватись коректно | Об’єкти перевіряються успішно   | 1 |
| 5 | Тест створення об’єктів | Об’єкти мають автоматично створитись | Об’єкти створені успішно   | 1 |
| 6 | Тест автоматизації правил | Правила мають автоматично створитись | Правила створені успішно | 1 |
| 7 | Перевірка функції створення VRF | Успішне створення VRF | VRF створено без помилок | 1 |
| 8 | Перевірка генерації файлів конфігурації | Файли генеруються автоматично | Файли успішно сформовані   | 1 |
| 9 | Перевірка інтерактивних сповіщень | Сповіщення з’являються у відповідь на події | Сповіщення коректно відображаються   | 1 |

Продовження табл. 4.1

| | | | | |
|---|---|---|---|---|
| 10 | Тестування Terraform Plan | Terraform Plan виконується без помилок | Plan виконано, помилок не виявлено | 1 |
| 11 | Тестування Terraform Apply | Terraform Apply виконує зміни коректно | Зміни застосовано, система стабільна | 1 |

*Джерело*: побудовано автором

Таблиця демонструє результати різних тестів, які були проведені для забезпечення надійності та коректності роботи вебплатформи. Це включало перевірки на правильність вводу даних, стабільність з’єднань із зовнішніми ресурсами, а також належну роботу автоматизованих процесів. У результаті проведеного тестування, критичних помилок не виявлено. Функціонал вебплатформи працює належним чином.

# ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра було розроблено інформаційну технологію автоматизації конфігурації мережевих пристроїв та сервісів у вигляді відповідної вебплатформи.

Проведено аналіз предметної області. Він підтвердив потребу в розробці інформаційної технології автоматизації конфігурації мережевих пристроїв та сервісів у вигляді відповідної вебплатформи. Наразі дійсно існує відчутний дефіцит повноцінно реалізованих рішень, які б вирішували всі нагальні проблеми в даній сфері. Здійснений аналіз та порівняння технологій автоматизації розгортання конфігурацій. Це дозволило визначити, які з них найкраще підходять для досягнення мети даної роботи, забезпечуючи належне поєднання гнучкості, ефективності та сумісності з існуючою інфраструктурою.

Проведене детальне планування робіт представленого проєкту (Додаток А).

Було сформовано мету дослідження, функціональні та нефункціональні вимоги до розроблюваної інформаційної технології автоматизації мережевих конфігурацій. Також визначено завдання для реалізації даного проєкту та здійснено детальне планування його робіт, що дозволило встановити часові рамки для їх виконання.

Наступний етап полягав у проєктуванні структури розроблюваної вебплатформи для автоматизації мережевих конфігурацій, використовуючи IDEF0 діаграми для визначення основних процесів і деталізації через декомпозицію. Застосування Use Case діаграм надало можливість демонстрації різних сценаріїв використання власної розробки.

Була розроблена інформаційна технологія у вигляді вебплатформи для автоматизації мережевих конфігурацій із використанням таких інструментів, як Flask, JavaScript, MongoDB, Ansible та Terraform. Усі функціональні вимоги, визначені в теоретичній частині проєкту, були успішно реалізовані. Ansible

використовувався для автоматизації мережевих налаштувань, а Terraform – для управління інфраструктурою. Проведено тестування розробленого програмного продукту методом «чорного ящика», і значних проблем у роботі платформи не виявлено. Результати тестування підтвердили високу надійність та ефективність реалізації основних функцій продукту даного проєкту.

Із впровадженням вебплатформи для автоматизації конфігураційних процесів мережі в діяльність компанії Infopulse (Додаток Б), мережеві інженери значно скоротили час необхідний для виконання своїх задач. Спочатку, задачі ручної конфігурації займали від 1 до 3 годин, але зараз, завдяки застосуванню розробленої вебплатформи, цей час зменшився на 10%. Це позитивно вплинуло на продуктивність роботи, оскільки тепер інженери мають можливість зосередитися на більш складних та стратегічних завданнях. Використання даної платформи не лише спростило виконання вищезазначених процесів, а й забезпечило зниження виникнення помилок у конфігураціях, що є важливим для сучасних мережевих середовищ.

Результати даної роботи були апробовані англійською мовою на науково-практичній конференції ІМА-2024 (Додаток В).

Усі вихідні коди зібрані та представлені в Додатку Г.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Are the Benefits of Network Automation? URL: https://www.cprime.com/resources/blog/what-are-the-benefits-of-network-automation/ (дата звернення: 10.03.2024)

2. Network Automation: Streamlining processes for better business agility URL: https://www.linkedin.com/pulse/network-automation-streamlining-processes-better-business-6tf0c/ (дата звернення: 10.03.2024)

3. Network Automation - A Deep Dive into Modern Network Automation by Using REST APIs URL: https://www.researchgate.net/publication/372875266_Network_Automation_-_A_Deep_Dive_into_Modern_Network_Automation_by_Using_REST_APIs (дата звернення: 10.03.2024)

4. The Power of Network Automation: What You Need to Know URL: https://www.cprime.com/resources/blog/what-are-the-benefits-of-network-automation/ (дата звернення: 10.03.2024)

5. Configuration as Code – moving in the right direction URL: https://codilime.com/blog/configuration-as-code-moving-in-right-direction/ (дата звернення: 10.03.2024)

6. Muhammad, T., & Munir, M. (2023). Network Automation. European Journal of Technology, 7(2), 23 - 42. DOI: https://doi.org/10.47672/ejt.1547

7. Mihaila, Paul, Titus Constantin Balan, Radu Curpen and Florin Sandu. “Network Automation and Abstraction using Python Programming Methods.” MACRo 2015 2 (2017): 103 - 95.

8. Datta, Anirban, A. T. M. Asif Imran and Chinmay Biswas. “Network Automation: Enhancing Operational Efficiency Across the Network Environment.” ICRRD Quality Index Research Journal (2023). DOI: https://doi.org/10.53272/icrrd.v4i1.1

9. Z. Li, B. Zhou, Z. Xiong, X. Zhang and W. Zhang, "Design of a Highly Concurrent Plug-in System for Network Automation," 2023 10th International Forum on Electrical Engineering and Automation (IFEEA), Nanjing, China, 2023, pp. 788-793, DOI: 10.1109/IFEEA60725.2023.10429677.

10. Fuzi, Mohd Faris Mohd, K. Abdullah, Iman Hazwam Abd Halim and Rafiza Ruslan. “Network Automation using Ansible for EIGRP Network.” Journal of Computing Research and Innovation (2021). URL: https://api.semanticscholar.org/CorpusID:240599269 (дата звернення: 10.03.2024)

11. Islami, M.F. , Musa , P. and Lamsani , M. 2020. Implementation of Network Automation using Ansible to Configure Routing Protocol in Cisco and Mikrotik Router with Raspberry PI: Array. Jurnal Ilmiah Komputasi. 19, 2 (Jun. 2020), 127–134. DOI:https://doi.org/10.32409/jikstik.19.2.80.

12. Wijaya, J. (2018). Network Automation using Ansible for Cisco Routers Basic Configuration. URL: https://osf.io/gxhjb/download (дата звернення: 10.03.2024)

13. Choi, Brendan and Erwin Medina. “Introduction to Ansible Network Automation: A Practical Primer.” Introduction to Ansible Network Automation (2023). DOI:10.1007/978-1-4842-9624-0

14. Choi, Brendan. “Python Network Automation Labs: Ansible, pyATS, Docker, and the Twilio API.” (2021). DOI:10.1007/978-1-4842-6806-3_16

15. Jason Edelman, “Network automation with ansible”, (2016). URL: https://www.oreilly.com/content/network-automation-with-ansible/ (дата звернення: 10.03.2024)

16. Howard, Michael. “Terraform - Automating Infrastructure as a Service.” ArXiv abs/2205.10676 (2022). DOI: 10.48550/arXiv.2205.10676

17. Modi, Ritesh. “CI/CD with Terraform.” Deep-Dive Terraform on Azure (2021). DOI:10.1007/978-1-4842-7328-9_7

18. Introduction to Puppet URL: https://www.puppet.com/docs/puppet/6/puppet_overview.html (дата звернення: 10.03.2024)

19. Chief Automation URL: https://chiefautomation.com/ (дата звернення: 10.03.2024)

20. Ansible. Ansible Documentation. URL: https://docs.ansible.com/ansible/latest/index.html (дата звернення: 10.03.2024).

21. Terraform. Terraform Documentation. URL:: https://terraform-docs.io/user-guide/configuration/ (дата звернення: 10.03.2024).

22. HashiCorp. Terraform AWS Provider Documentation. URL: https://registry.terraform.io/providers/hashicorp/aws/latest/docs (дата звернення: 10.03.2024).

23. Create and run your first GitLab CI/CD URL: https://docs.gitlab.com/ee/ci/quick_start/ (дата звернення: 10.03.2024)

24. Chainikov S. Information technology of software architecture structural synthesis of information system / S. Chainikov, A. Solodovnikov // EUREKA: Physics and Engineering. – 2016. – Volume 4(5). – P. 25–32. URL: http://openarchive.nure.ua/handle/document/5399 (дата звернення: 10.03.2024)

25. ТЕОРЕТИКО-МНОЖИННИЙ ПІДХІД ДО ОПИСУ СИСТЕМ. URL: https://stud.com.ua/109947/informatika/teoretiko_mnozhinniy_pidhid_opisu_sistem (дата звернення: 10.03.2024)

26. What is YAML? URL: https://www.redhat.com/en/topics/automation/what-is-yaml (дата звернення: 10.03.2024)

27. The Use of IDEF0 for the Design and Specification of MethodologiesUse case URL: https://www.researchgate.net/publication/2447898_The_Use_of_IDEF0_for_the_Design_and_Specification_of_Methodologies (дата звернення: 10.03.2024)

28. UML Use Case Diagram URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 10.03.2024)

29. Virtual Routing and Forwarding URL: https://secure.cisco.com/secure-firewall/docs/virtual-routing-and-forwarding (дата звернення: 10.03.2024)

30. Cloud computing with AWS URL: https://aws.amazon.com/what-is-aws/?nc1=h_ls (дата звернення: 10.03.2024)

31. What is MongoDB? Features and how it works URL: https://www.techtarget.com/searchdatamanagement/definition/MongoDB (дата звернення: 10.03.2024)

32. What is Flask Python URL: https://pythonbasics.org/what-is-flask-python/ (дата звернення: 10.03.2024)

33. What is a REST API? URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата звернення: 10.03.2024)

34. Using the Fetch API URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (дата звернення: 10.03.2024)

35. HTTP request methods POST URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST (дата звернення: 10.03.2024)

36. PyMongo 4.7.1 Documentation URL: https://pymongo.readthedocs.io/en/stable/ (дата звернення: 10.03.2024)

37. CHECK POINT API Reference URL: https://sc1.checkpoint.com/documents/latest/api_reference/index.html (дата звернення: 10.03.2024)

38. Introduction to Security Management URL: https://sc1.checkpoint.com/documents/R81/WebAdminGuides/EN/CP_R81_SecurityManagement_AdminGuide/Topics-SECMG/Welcome.htm (дата звернення: 10.03.2024)

39. Check Point Next Generation Security Gateway Solution URL: https://sc1.checkpoint.com/documents/R80.30SP/WebAdminGuides/EN/CP_R80.30SP_Maestro_NextGenSecurityGateway_Guide/html_frameset.htm?topic=documents/R80.30SP/WebAdminGuides/EN/CP_R80.30SP_Maestro_NextGenSecurityGateway_Guide/136970 (дата звернення: 10.03.2024)

40. Cisco Cloud Services Router 1000v URL: https://www.cisco.com/c/en/us/support/routers/cloud-services-router-1000v/model.html (дата звернення: 10.03.2024)

41. Using IAM with CodeCommit: Git credentials, SSH keys, and AWS access keys URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_ssh-keys.html (дата звернення: 10.03.2024)

42. Infrastructure as Code with Terraform and GitLab URL: https://docs.gitlab.com/ee/user/infrastructure/iac/ (дата звернення: 10.03.2024)

43. Amazon EC2 URL: https://aws.amazon.com/ec2/?nc1=h_ls (дата звернення: 10.03.2024)

44. Creating Your First Instance with Terraform URL: https://medium.com/@rafael_muller/creating-your-first-instance-with-terraform-20334f3023ef (дата звернення: 10.03.2024)

45. Black Box Testing URL: https://www.imperva.com/learn/application-security/black-box-testing/ (дата звернення: 10.03.2024)

# ДОДАТОК А

## Планування робіт

**Деталізація мети проєкту методом SMART.** Продуктом дипломного проєкту є вебплатформа, призначена для автоматизації процесів конфігурації мережевих пристроїв та сервісів. Результати деталізації методом SMART розміщені у таблиці А.1.

Таблиця А.1 − Деталізація мети методом SMART

| | |
|---|---|
| Specific (конкретна) | Розробка інформаційної технології у вигляді відповідної вебплатформи для генерації конфігураційних файлів для мережевих пристроїв, таких як файрволи, роутери та балансувальники навантаження, яка дозволить користувачам легко вибирати параметри та шаблони для автоматизації їх мережевих конфігурацій. |
| Measurable (вимірювана) | Впроваджена вебплатформа, яка здатна автоматизувати конфігурацію пристроїв. |
| Achievable (досяжна) | Для реалізації проєкту необхідно забезпечити ресурси, такі як доступ до серверів, необхідне програмне забезпечення (Ansible, Terraform, MongoDB), а також час для розробки і тестування. Передбачається, що команда складатиметься з кваліфікованих розробників та тестувальників. |
| Relevant (реалістична) | Проєкт спрямований на зменшення часу та помилок при здійсненні конфігурації мережевих установок, підвищення ефективності оперативного управління мережевою інфраструктурою. |
| Time-framed (обмежена у часі) | Проєкт має бути завершений і готовий до запуску протягом наступних 6 місяців з чіткими проміжними етапами розробки, тестування та впровадження. |

*Джерело*: побудовано автором (знімок з екрану)

**Планування змісту структури робіт ІТ-проєкту (WBS).** WBS (Work Breakdown Structure) є важливим інструментом у проєктному менеджменті, який допомагає в організації та виконанні проєктів через систематичне розбиття цілого проєкту на менші, легше керовані компоненти, відомі як робочі пакети. Цей підхід дозволяє командам краще розуміти обсяг робіт і ефективніше розподіляти ресурси.

Основна мета WBS – забезпечити структурований поділ робіт проєкту, який допомагає управлінню проєктами на кожному етапі від планування до виконання. WBS перетворює основні проєктні цілі на конкретні завдання, що можуть бути присвоєні окремим членам команди або групам.

Ієрархічна структура: WBS має деревоподібну структуру, де кожен наступний рівень є більш деталізованим розбивом вищестоящого. Вершина структури є сам проєкт, і вона поступово дробиться на фази, завдання, підзавдання, і так до найдрібніших компонентів.

Використання WBS сприяє кращій координації між членами команди, оскільки кожен знає свої завдання і розуміє, як вони вписуються в загальну картину проєкту. Це сприяє ефективнішій взаємодії та співпраці всередині команди.

Один з важливих аспектів управління проєктами є управління змінами. WBS дозволяє ефективно впорядковувати зміни, що вносяться до проєкту, оскільки зміни у будь-якій частині проєкту можна чітко ідентифікувати та визначити їх вплив на інші частини проєкту. WBS дозволяє менеджерам проєктів визначити пріоритетні завдання та виділяти ресурси, забезпечуючи, що критично важливі частини проєкту мають необхідні ресурси та увагу. Інтеграція з іншими інструментами управління проєктами: WBS ефективно інтегрується з іншими інструментами, такими як Gantt Charts, Project Management Software (наприклад, Microsoft Project), та Agile Boards. Це забезпечує гнучкість у виборі методології управління та підходів до виконання проєкту.

Завдяки WBS, створення документації та звітів стає більш структурованим та менш трудомістким. Кожен елемент у WBS може мати власні вимоги до документації, що спрощує збір та організацію проєктних даних. WBS дозволяє легко адаптуватися до змін, забезпечуючи при цьому, що всі зміни залишаються в рамках визначеного обсягу робіт. Зміни в одному сегменті WBS можуть бути оцінені на предмет їх впливу на інші сегменти, що допомагає уникнути небажаних відхилень від плану.

Розробка та застосування WBS є критично важливим аспектом успішного управління проєктом, який допомагає ефективно розподіляти ресурси, відстежувати прогрес та досягати проєктних цілей в умовах чіткої структури. WBS діаграма проєкту зображена на рисунку А.1.


Рисунок А.1 – WBS структура проєкту

*Джерело*: побудовано автором (знімок з екрану)

**Організаційна структура проєкту (OBS).** Організаційна структура проєкту (OBS) служить каркасом для забезпечення зв'язку між організаційними одиницями і проєктними завданнями, які вони мають виконати. Це критичний інструмент для ідентифікації і визначення ролей, відповідальностей та структурного розподілу команди проєкту.

Конструкція OBS вимагає від менеджерів проєктів чіткого розуміння всіх аспектів проєкту та його цілей, а також знання ресурсів і вміння аналізувати внутрішні процеси компанії.

Використання OBS у проєктному менеджменті є фундаментальним для успішної реалізації комплексних проєктів, особливо у великих і розгалужених організаціях. Забезпечуючи чітку організаційну структуру та інтеграцію з завданнями проєкту, OBS сприяє кращому плануванню, виконанню та контролю проєктів. OBS діаграма проєкту зображена на рисунку А.2.

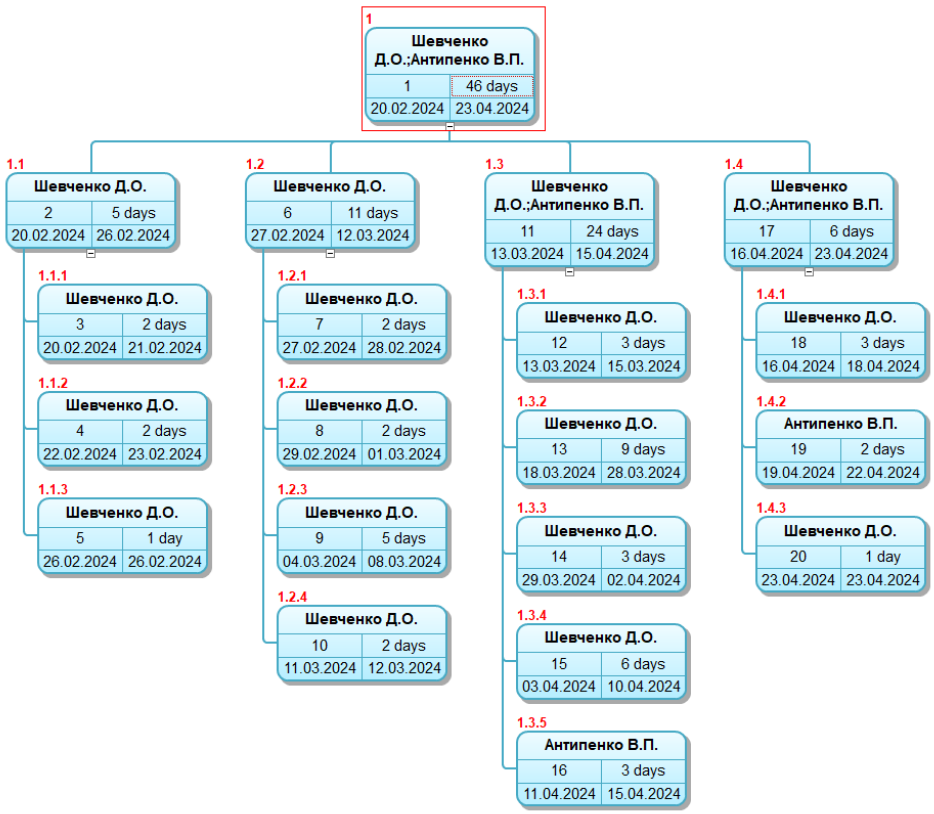



Рисунок А.2 – OBS структура проєкту

*Джерело*: побудовано автором (знімок з екрану)

**Побудова календарного графіка виконання ІТ-проєкту**. Для точного планування та відстеження ходу виконання проєкту важливо розробити календарний графік, який б враховував всі тимчасові обмеження та ресурсні можливості. Календарний графік робіт детально визначає періоди виконання кожного з завдань проєкту, узгоджуючи їх із загальнодоступними ресурсами та необхідністю дотримання встановлених термінів. За допомогою програми Microsoft Project побудовано календарний план (рис. А.3).

| Task Mode | Task Name | Duration | Start | Finish | Predecessors | Resource Names |
|---|---|---|---|---|---|---|
| | **Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів** | **46 days** | **Tue 20.02.24** | **Tue 23.04.24** | | |
| | **Аналіз предметної області** | **5 days** | **Tue 20.02.24** | **Mon 26.02.24** | | |
| | Огляд останніх досліджень | 2 days | Tue 20.02.24 | Wed 21.02.24 | | Шевченко Д.О. |
| | Огляд сучасного стану речей | 2 days | Thu 22.02.24 | Fri 23.02.24 | 3 | Шевченко Д.О. |
| | Визначення пробем інтеграції та взаємодії | 1 day | Mon 26.02.24 | Mon 26.02.24 | 4 | Шевченко Д.О. |
| | **Планування** | **11 days** | **Tue 27.02.24** | **Tue 12.03.24** | **2** | |
| | Деталізація завдань дослідження | 2 days | Tue 27.02.24 | Wed 28.02.24 | | Шевченко Д.О. |
| | Технічні вимоги до продукту | 2 days | Thu 29.02.24 | Fri 01.03.24 | 7 | Шевченко Д.О. |
| | Визначення технологій автоматизації | 5 days | Mon 04.03.24 | Fri 08.03.24 | 8 | Шевченко Д.О. |
| | Визначення технологій веб-платформи | 2 days | Mon 11.03.24 | Tue 12.03.24 | 9 | Шевченко Д.О. |
| | **Реалізація** | **24 days** | **Wed 13.03.24** | **Mon 15.04.24** | **6** | |
| | Побудова тестового кластеру мережі | 3 days | Wed 13.03.24 | Fri 15.03.24 | | Шевченко Д.О. |
| | Розробка та налаштування алгоритмів автоматизації | 9 days | Mon 18.03.24 | Thu 28.03.24 | 12 | Шевченко Д.О. |
| | Розробка інтерфейсу користувача | 3 days | Fri 29.03.24 | Tue 02.04.24 | 13 | Шевченко Д.О. |
| | Розробка бекенду | 6 days | Wed 03.04.24 | Wed 10.04.24 | 14 | Шевченко Д.О. |
| | Тестування | 3 days | Thu 11.04.24 | Mon 15.04.24 | 15 | Антипенко В.П. |
| | **Завершення** | **6 days** | **Tue 16.04.24** | **Tue 23.04.24** | **11** | |
| | Написання документації | 3 days | Tue 16.04.24 | Thu 18.04.24 | | Шевченко Д.О. |
| | Перевірка працездатності | 2 days | Fri 19.04.24 | Mon 22.04.24 | 18 | Антипенко В.П. |
| | Реліз | 1 day | Tue 23.04.24 | Tue 23.04.24 | 19 | Шевченко Д.О. |

Рисунок А.3 – Календарний план проєкту

*Джерело*: побудовано автором (знімок з екрану)

Діаграма Ганта служить ключовим інструментом у цьому процесі, оскільки вона забезпечує наглядність та зрозумілість плану проєкту. Вона відображає кожне завдання як горизонтальний бар на сітці календарних дат, де довжина кожного бару відображає тривалість завдання, а положення на осі часу вказує на

заплановані дати початку та завершення. На діаграмі Ганта можна також відзначити залежності між завданнями, позначивши стрілками або лініями, які показують, які завдання необхідно завершити перед початком наступних. Це допомагає команді проєкту візуально оцінити критичний шлях проєкту та розробити ефективну стратегію виконання робіт. Завдяки використанню програмного забезпечення для управління проєктами, такого як Microsoft Project, можливо автоматизувати багато аспектів планування та відстеження проєкту. Це включає автоматичне оновлення діаграми при внесенні змін у графік завдань, що дозволяє проєктному менеджеру та зацікавленим сторонам миттєво бачити вплив цих змін на загальний план проєкту. За допомогою програми Microsoft Project побудовано діаграму Ганта (рис. А.4).

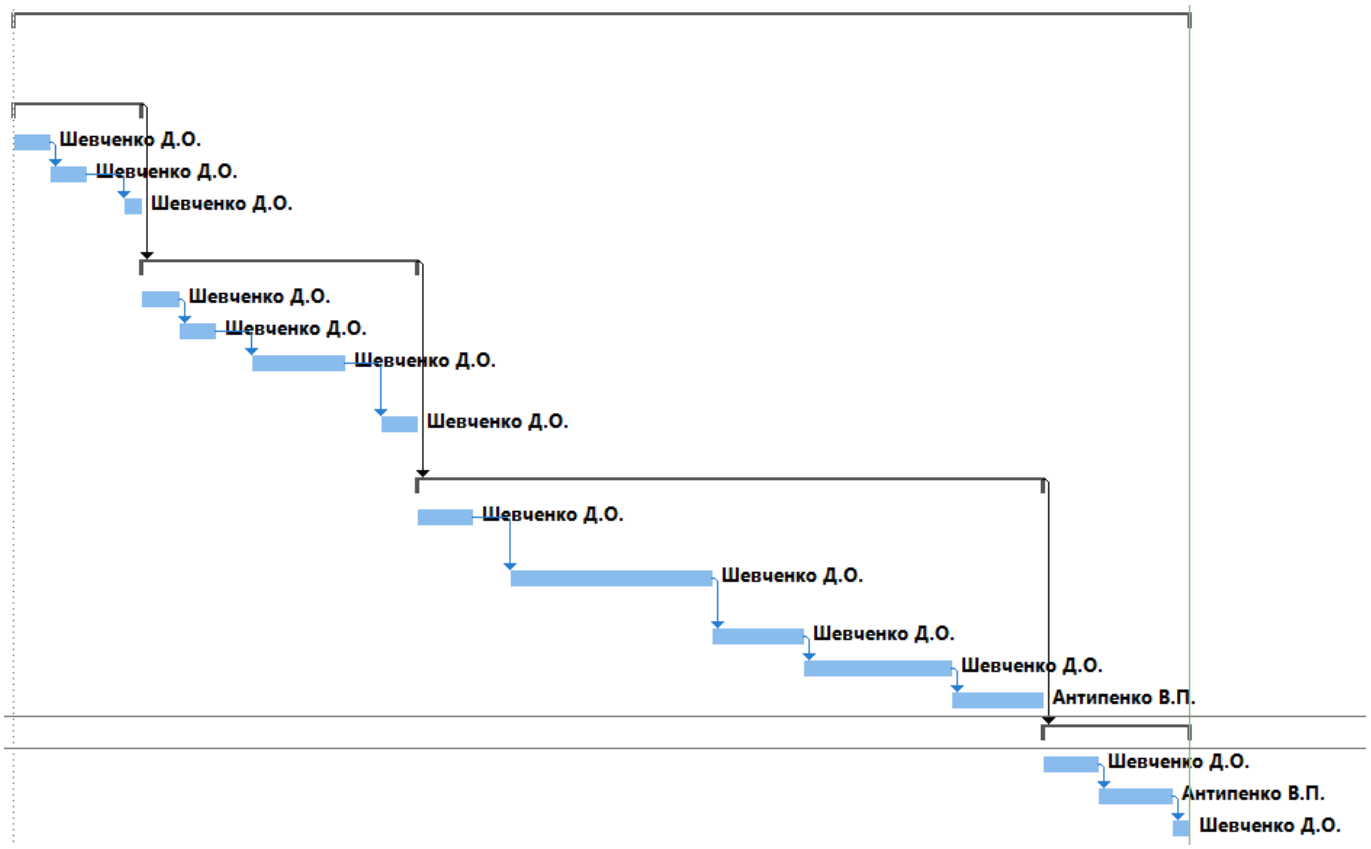



Рисунок А.4 – Діаграма Ганта

*Джерело*: побудовано автором (знімок з екрану)

Таким чином, календарний графік та діаграма Ганта є незамінними інструментами у процесі планування, виконання та контролю проєкту, надаючи необхідну структуру та ясність, які забезпечують успішне досягнення проєктних цілей.

**Управління ризиками.** Управління ризиками в даному проєкті полягає в ідентифікації, аналізі та вжитті заходів щодо потенційних ризиків, які можуть виникнути під час розробки та впровадження системи автоматизації конфігурацій мережевих пристроїв. Завдання цього процесу – мінімізувати негативний вплив на проєкт і забезпечити його успішне завершення в рамках встановлених термінів і бюджету. У таблиці А.2 представлено шкалу для класифікації ризиків за величиною впливу на проєкт та ймовірністю виникнення.

Таблиця А.2 – Визначення ймовірності, впливу та рангу ризиків проєкту

| № | Назва ризику | Ймовірність (0,1 – 0,9) | Вплив (0,05-0,8) | Ранг |
|---|---|---|---|---|
| 1 | Зміна вимог технічного завдання | 0,5 | 0,2 | 0,1 |
| 2 | Проблеми з підключенням до Інтернету | 0,3 | 0,4 | 0,12 |
| 3 | Збій електромережі | 0,3 | 0,4 | 0,12 |
| 4 | Хвороба працівника | 0,5 | 0,2 | 0,1 |
| 5 | Збій у роботі мережі | 0,3 | 0,4 | 0,12 |
| 6 | Збій у роботі техніки розробника | 0,3 | 0,2 | 0,06 |
| 7 | Нечітко виділені вимоги | 0,1 | 0,4 | 0,04 |
| 8 | Неправильний розподіл часу | 0,1 | 0,2 | 0,02 |
| 9 | Непрацездатність хмарного сервісу | 0,1 | 0,4 | 0,04 |
| 10 | Виявлення помилок під час тестування | 0,5 | 0,1 | 0,05 |
| 11 | Збій програмного забезпечення | 0,3 | 0,4 | 0,12 |

*Джерело*: побудовано автором

Для мінімізації негативного впливу ризиків на проєкт необхідно здійснити планування реагування на них. Це включає оцінку ефективності розробки та аналіз потенційних наслідків для проєкту. Оцінки базуються на критеріях, представлених у таблиці А.2. В результаті була сформована матриця ймовірностей виникнення ризиків та їхнього впливу, показана у таблиці А.3. На матриці зелений колір вказує на прийнятні ризики, жовтий – на виправдані, а червоний – на недопустимі.

Таблиця А.3 – Матриця ймовірності та впливу згідно проєкту

| Ймовірність | Вплив загрози(ризику) | | | | |
|---|---|---|---|---|---|
| | Дуже малий 0,05 | Малий 0,1 | Середній 0,2 | Великий 0,4 | Дуже великий 0,8 |
| 0,9 | | | | | |
| 0,7 | | | | | |
| 0,5 | | R10(0.05) | R1(0.1)<br>R4(0.1) | | |
| 0,3 | | | R6(0.06) | R2(0.12)<br>R3(0.12)<br>R5(0.12)<br>R11(0.12) | |
| 0,1 | | | R8(0.02) | R7(0.04)<br>R9(0.04) | |

*Джерело*: побудовано автором

Ризики проєкту класифікуються за рівнем відповідно до значення індексу, який представлений у таблиці А.4. Деталізація ризиків та стратегій реагування на кожен з них викладена у таблиці А.5.

Таблиця А.4 – Шкала оцінювання за рівнем ризику.

| № | Назва | Межі | Ризик, які входять (номера) |
|---|---|---|---|
| 1 | Прийнятні | 0,005≤ R ≤0,05 | 7, 8, 9, 10 |
| 2 | Виправдані | 0,05< R ≤0,14 | 1, 2, 3, 4, 5, 6, 11 |
| 3 | Недопустимі | 0,14< R ≤0,72 | |

*Джерело*: побудовано автором

Таблиця А.5 – Ризики та стратегії реагування

| ID | Статус ризику | Опис | Ймовірність | Вплив | Ранг ризику | План А | Тип стратегії реагування | План Б |
|---|---|---|---|---|---|---|---|---|
| 1 | Новий | Зміна вимог технічного завдання | Середній | Середній | 0,1 | Обговорити питання на початку проєкту | Ухилення | - |
| 2 | Новий | Проблеми з підключенням до Інтернету | Низький | Високий | 0.12 | Підключити два провайдери інтернету | Ухилення | Залучити спеціаліста для усунення проблем |
| 3 | Новий | Збій електромережі | Низький | Високий | 0.12 | Залучити спеціаліста для усунення збоїв | Зменшення | - |
| 4 | Новий | Хвороба працівника | Середній | Середній | 0.4 | Проводити профілактику захворювань | Зменшення | - |

Продовження табл. Б.5.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | Новий | Збій у роботі мережі | Низький | Високий | 0.12 | Залучити спеціаліста для збою. | Зменшення | Замінити проблемний елемент мережі. |
| 6 | Новий | Збій у роботі техніки розробника | Середній | Середній | 0.06 | Залучити спеціаліста для усунення проблем | Зменшення | Змінити обладнання |
| 7 | Новий | Нечітко виділені вимоги | Низький | Високий | 0.04 | Обговорити не зрозумілі пункти завдання | Зменшення | - |
| 8 | Новий | Неправильний розподіл часу | Низький | Середній | 0.02 | Перевизначити терміни виконання завдання | Зменшення | Працювати понаднормово |
| 9 | Новий | Непрацездатність хмарного сервісу | Низький | Високий | 0.04 | - | - | - |

Продовження табл. Б.5

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 10 | Новий | Виявлення помилок під час тестування | Середній | Середній | 0.05 | Виправити помилки | Зменшення | Проігнорувати помилки якщо вони не значні |
| 11 | Новий | Збій програмного забезпечення | Низький | Високий | 0.12 | Виправити помилку, що призводить до збою | Зменшення | Перевстановити програмне забезпечення |

*Джерело*: побудовано автором

# ДОДАТОК Б

## Акт впровадження

АКТ

Впровадження результатів студентської магістерської роботи «Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів» у діяльності компанії Infopulse у 2024 р.

Цей акт підтверджує, що результати студентської магістерської дипломної роботи Шевченка Д.О. на тему «Інформаційна технологія автоматизації конфігурації мережевих пристроїв та сервісів» впроваджено у діяльність компанії.

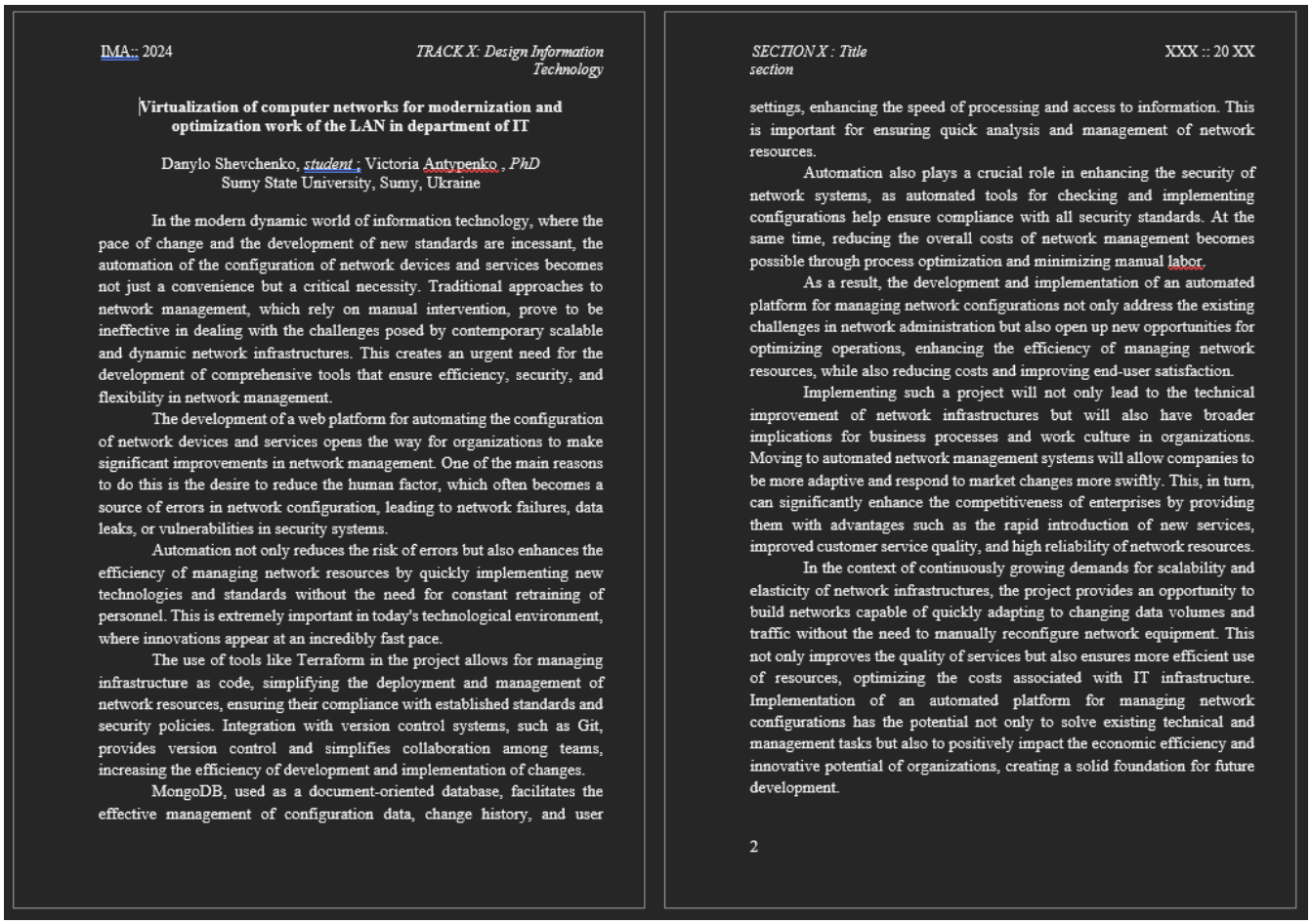
Було впроваджено у використання систему для автоматизації конфігурації мережевих пристроїв та сервісів, для покращенням ефективності та зниження часу на налаштування мережевих пристроїв.

Впровадження результатів студентської роботи Шевченка Д.О. дозволяє робітникам компанії покращити процеси налаштування та управління конфігураціями мережі, що сприяє значному підвищенню її оперативної стабільності та надійності.

Анна Кащук

# ДОДАТОК В

## Апробація та впровадження результатів дослідження





**Virtualization of computer networks for modernization and optimization work of the LAN in department of IT**

Danylo Shevchenko, *student* ; Victoria Antypenko , *PhD*
Sumy State University, Sumy, Ukraine

In the modern dynamic world of information technology, where the pace of change and the development of new standards are incessant, the automation of the configuration of network devices and services becomes not just a convenience but a critical necessity. Traditional approaches to network management, which rely on manual intervention, prove to be ineffective in dealing with the challenges posed by contemporary scalable and dynamic network infrastructures. This creates an urgent need for the development of comprehensive tools that ensure efficiency, security, and flexibility in network management.

The development of a web platform for automating the configuration of network devices and services opens the way for organizations to make significant improvements in network management. One of the main reasons to do this is the desire to reduce the human factor, which often becomes a source of errors in network configuration, leading to network failures, data leaks, or vulnerabilities in security systems.

Automation not only reduces the risk of errors but also enhances the efficiency of managing network resources by quickly implementing new technologies and standards without the need for constant retraining of personnel. This is extremely important in today's technological environment, where innovations appear at an incredibly fast pace.

The use of tools like Terraform in the project allows for managing infrastructure as code, simplifying the deployment and management of network resources, ensuring their compliance with established standards and security policies. Integration with version control systems, such as Git, provides version control and simplifies collaboration among teams, increasing the efficiency of development and implementation of changes.

MongoDB, used as a document-oriented database, facilitates the effective management of configuration data, change history, and user



settings, enhancing the speed of processing and access to information. This is important for ensuring quick analysis and management of network resources.

Automation also plays a crucial role in enhancing the security of network systems, as automated tools for checking and implementing configurations help ensure compliance with all security standards. At the same time, reducing the overall costs of network management becomes possible through process optimization and minimizing manual labor.

As a result, the development and implementation of an automated platform for managing network configurations not only address the existing challenges in network administration but also open up new opportunities for optimizing operations, enhancing the efficiency of managing network resources, while also reducing costs and improving end-user satisfaction.

Implementing such a project will not only lead to the technical improvement of network infrastructures but will also have broader implications for business processes and work culture in organizations. Moving to automated network management systems will allow companies to be more adaptive and respond to market changes more swiftly. This, in turn, can significantly enhance the competitiveness of enterprises by providing them with advantages such as the rapid introduction of new services, improved customer service quality, and high reliability of network resources.

In the context of continuously growing demands for scalability and elasticity of network infrastructures, the project provides an opportunity to build networks capable of quickly adapting to changing data volumes and traffic without the need to manually reconfigure network equipment. This not only improves the quality of services but also ensures more efficient use of resources, optimizing the costs associated with IT infrastructure. Implementation of an automated platform for managing network configurations has the potential not only to solve existing technical and management tasks but also to positively impact the economic efficiency and innovative potential of organizations, creating a solid foundation for future development.

# ДОДАТОК Г

## Лістинг коду модулів вебплатформи

### Код файлу index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset="UTF-8">
   <title>Dashboard</title>
   <link rel="stylesheet" href="styles.css">
  </head>
  <body>
   <header>
     <h1>Automation Dashboard</h1>
   </header>
   <div class="container">
     <nav id="side-menu">
      <ul>
        <li class="selected"><a href="#" id="link-Function1">Check Point</a></li>
        <li><a href="#" id="link-Function2">AWS Instances</a></li>
        <li><a href="#" id="link-Function3">Cisco Routers</a></li>
      </ul>
     </nav>
     <section id="main-content">
      <div style="display:block" id="Function1">
        <h2>Check Point Rules</h2>
        <div class="form-container">
         <form id="ruleForm">
           <div class="input-group">
            <label for="src-ip">Source IP:</label>
            <input type="text" id="src-ip" name="src-ip" required>
            <div class="radio-group">
              <input  type="radio"  id="src-host"  name="src-type"  value="host"
checked>
              <label for="src-host">Host</label>
              <input      type="radio"      id="src-network"      name="src-type"
value="network">
```

```
            <label for="src-network">Network</label>
            <input type="radio" id="src-group" name="src-type" value="group">
            <label for="src-group">Group</label>
          </div>
        </div>
        <div class="input-group">
          <label for="dst-ip">Destination IP:</label>
          <input type="text" id="dst-ip" name="dst-ip" required>
          <div class="radio-group">
            <input  type="radio"  id="dst-host"  name="dst-type"  value="host"
checked>
            <label for="dst-host">Host</label>
            <input      type="radio"      id="dst-network"      name="dst-type"
value="network">
            <label for="dst-network">Network</label>
            <input type="radio" id="dst-group" name="dst-type" value="group">
            <label for="dst-group">Group</label>
          </div>
        </div>
        <div class="input-group">
          <label for="port">Port:</label>
          <input type="text" id="port" name="port">
          <div class="radio-group">
            <input type="radio" id="protocol-tcp" name="protocol" value="tcp"
checked>
            <label for="protocol-tcp">TCP</label>
            <input type="radio" id="protocol-udp" name="protocol" value="udp">
            <label for="protocol-udp">UDP</label>
            <input     type="radio"     id="protocol-icmp"     name="protocol"
value="icmp">
            <label for="protocol-icmp">ICMP</label>
          </div>
        </div>
        <button type="submit">Check</button>
        <button type="button" id="runPlaybook" disabled>Create Rule</button>
<!-- New button for running playbook -->
      </form>
    </div>
    <div class="form-container">
      <!-- New fields for object creation -->
      <!-- New fields for object creation -->
```

```
        <form id="objectForm">
          <div class="input-group">
           <label for="object-name">Object Name:</label>
           <input type="text" id="object-name" name="object-name" required>
          </div>
          <div class="input-group">
           <label for="object-ip">IP Address:</label>
           <input type="text" id="object-ip" name="object-ip" required>
          </div>
          <div   class="input-group"   id="network-mask-group"   style="display:
none;">
           <label for="object-mask">Subnet Mask:</label>
           <input type="text" id="object-mask" name="object-mask">
          </div>
          <div class="radio-group">
           <input  type="radio"  id="type-host"  name="object-type"  value="host"
checked>
           <label for="type-host">Host</label>
           <input     type="radio"     id="type-network"     name="object-type"
value="network">
           <label for="type-network">Network</label>
          </div>
          <button type="button" id="checkObject">Check</button>
          <button type="button" id="createObject" disabled>Create</button>
        </form>
       </div>
       <div class="form-container">
        <form id="objectForm">
          <div class="input-group">
           <label for="service-name">Service Name:</label>
           <input type="text" id="service-name" name="service-name" required>
          </div>
          <div class="input-group">
           <label for="service-port">Port:</label>
           <input type="text" id="service-port" name="service-port" required>
          </div>
          <div class="radio-group">
           <input   type="radio"   id="service-type-tcp"   name="service-type"
value="tcp" checked>
           <label for="service-type-tcp">TCP</label>
```

```
            <input   type="radio"   id="service-type-udp"   name="service-type"
value="udp">
            <label for="service-type-udp">UDP</label>
          </div>
          <button type="button" id="checkService">Check</button>
          <button type="button" id="createService" disabled>Create</button>
        </form>
      </div>
    </div>
    <div style="display:none" id="Function2">
      <h2>Create AWS Instance</h2>
      <form id="createInstanceForm" class="form-container">
       <div class="input-group">
         <label for="instance-name">Instance Name:</label>
         <input type="text" id="instance-name" name="instance-name" required>
       </div>
       <div class="input-group">
         <label for="instance-type">Instance Type:</label>
         <select id="instance-type" name="instance-type">
          <option value="t2.micro">t2.micro</option>
          <option value="t2.small">t2.small</option>
          <option value="t2.large">t2.large</option>
         </select>
       </div>
       <div class="input-group">
         <label for="os-type">Operating System:</label>
         <select id="os-type" name="os-type">
          <option value="ubuntu">Ubuntu</option>
          <option value="amazon_linux">Amazon Linux</option>
          <option value="windows">Windows</option>
          <option value="redhat">Redhat</option>
         </select>
       </div>
       <button type="submit">Create Instance</button>
      </form>
    </div>
    <div style="display:none" id="Function3">
      <h2>Create VRF Configuration</h2>
      <form id="createVrfForm" class="form-container">
       <div class="input-group">
         <label for="vrf-name">VRF Name:</label>
```

```
          <input type="text" id="vrf-name" name="vrf-name" required>
        </div>
        <div class="input-group">
          <label for="vrf-description">Description:</label>
          <input   type="text"   id="vrf-description"   name="vrf-description"
required>
        </div>
        <div class="input-group">
          <label for="interface-name">Interface Name:</label>
          <input type="text" id="interface-name" name="interface-name" required>
        </div>
        <div class="input-group">
          <label for="subint-id">Subinterface ID:</label>
          <input type="text" id="subint-id" name="subint-id" required>
        </div>
        <div class="input-group">
          <label for="vlan-id">VLAN ID:</label>
          <input type="text" id="vlan-id" name="vlan-id" required>
        </div>
        <div class="input-group">
          <label for="ip-address">IP Address:</label>
          <input type="text" id="ip-address" name="ip-address" required>
        </div>
        <div class="input-group">
          <label for="subnet-mask">Subnet Mask:</label>
          <input type="text" id="subnet-mask" name="subnet-mask" required>
        </div>
        <button type="button" onclick="submitVrfForm()">Create VRF</button>
        <button        type="button"        onclick="generateConfig()">Generate
Config</button>
       </form>
      </div>
     </section>
   </div>
   <footer>
     <p>Copyright © 2024</p>
   </footer>
   <script src="script.js"></script>
  </body>
</html>
```

Код файлу styles.css

```
body {
  font-family: Arial, sans-serif;
  background-color: #f0f0f0;
  margin: 0;
  padding: 0;
}

header {
  background-color: #004D40;
  color: white;
  padding: 10px 20px;
  text-align: center;
}

button:disabled {
  background-color: #cccccc; /* Grey background for disabled buttons */
  color: #666666; /* Darker text color for better contrast */
  cursor: not-allowed; /* Show a 'not-allowed' cursor on hover */
}



.container {
  display: flex;
  flex-direction: row;
  margin-top: 20px;
}

#Function1 .form-container {
  margin-bottom: 20px; /* Increase the bottom margin */
}



#side-menu {
  width: 200px;
  background: #f4f4f4;
  height: calc(100vh - 40px); /* Assuming header and footer are 20px each */
  padding: 20px;
}

#side-menu ul {
```

```
  list-style-type: none;
  padding: 0;
  margin: 0;
}

#side-menu li {
  padding: 10px;
  background-color: #f4f4f4;
  margin-bottom: 2px; /* Add spacing between menu items */
  transition: background-color 0.3s ease; /* Smooth transition for hover effect
*/
}

#side-menu li.selected, #side-menu li:hover {
  background-color: #ccc;
}

#main-content {
  flex-grow: 1;
  padding: 20px;
}

.form-container {
  background-color: #fff;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0,0,0,0.1);
  width: auto;
}


.input-group {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}

.input-group label {
  margin-right: 10px; /* Space between label and input */
  width: 20%; /* Labels take up 20% of the group width */
```

```
}

.input-group input[type="text"],
.input-group input[type="radio"] + label {
  flex-grow: 1;
}




.input-group select, .input-group input[type="text"] {
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
  width: 100%; /* Full width of the grid column */
}

button.btn-primary {
  background-color: #007BFF;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  width: 100%; /* Full-width button */
  font-size: 16px; /* Larger font size for better readability */
  transition: background-color 0.2s ease-in-out;
}



.radio-group {
  display: flex;
  justify-content: flex-start;
}

.radio-group input[type="radio"] {
  margin-right: 5px;
}

.radio-group label {
```

```
  margin-right: 20px;
}

button {
  display: inline-block;
  padding: 10px 15px;
  background-color: #007BFF;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  margin-top: 10px;
}

button:hover {
  background-color: #0056b3;
}

footer {
  background-color: #004D40;
  color: white;
  text-align: center;
  padding: 10px;
}

@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }

  #side-menu {
    width: 100%;
    height: auto;
    padding: 0; /* Adjust padding for mobile */
    margin-bottom: 20px; /* Space between menu and content */
  }

  #main-content {
    order: -1; /* Main content comes first on small screens */
  }
}
```

```
#side-menu ul li a {
  text-decoration: none;
  color: #333;
  display: block;
  padding: 10px;
  transition: background-color 0.3s, color 0.3s;
}

#side-menu ul li a:hover {
  background-color: #C8E6C9;
  color: #005F40;


.user-icon {
  padding-right: 10px;
}



#Function2 {
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
}

@media (max-width: 768px) {
  .input-group {
    grid-template-columns: 1fr; /* Stack label and input vertically on small
screens */
  }

  .input-group label {
    text-align: left; /* Align labels to left on small screens */
    padding-bottom: 5px;
  }
}


#Function3 {
```

```
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
  width: 100%; /* Ensures that the div takes full width */
}

#Function3 .form-container {
  background-color: #fff;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0,0,0,0.1);
  margin: auto;
  width: 80%;
  max-width: 600px;
}

#Function3 .input-group {
  margin-bottom: 20px;
  display: grid;
  grid-template-columns: 1fr 2fr;
}

#Function3 .input-group label {
  padding-right: 10px;
  text-align: right;
  display: flex;
  align-items: center;
  justify-content: flex-end;
}

#Function3 .input-group input[type="text"],
#Function3 .input-group select {
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
  width: 100%;
}

#Function3 button.btn-primary {
  background-color: #007BFF;
```

```
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  width: 100%;
  font-size: 16px;
  transition: background-color 0.2s ease-in-out;
}

#Function3 button.btn-primary:hover {
  background-color: #0056b3;
}

@media (max-width: 768px) {
  #Function3 .input-group {
    grid-template-columns: 1fr; /* Stack label and input vertically on small
screens */
  }

  #Function3 .input-group label {
    text-align: left;
    padding-bottom: 5px;
  }
}
```

## Код файлу script.js

```
const sections = document.querySelectorAll('#main-content > div');
const links = document.querySelectorAll('#side-menu a');


function hideAllSections() {
  sections.forEach(section => {
    section.style.display = 'none';
  });
}

function clearSelected() {
  links.forEach(link => {
```

```
    link.parentElement.classList.remove('selected');
  });
}

links.forEach(link => {
  link.addEventListener('click', function(event) {
    hideAllSections();
    clearSelected();

    const functionNumber = this.id.split('-')[1];
    document.getElementById(functionNumber).style.display = 'block';
    this.parentElement.classList.add('selected');
  });
});

// Initialize the default visibility
hideAllSections();
document.getElementById('Function1').style.display = 'block';
document.getElementById('link-
Function1').parentElement.classList.add('selected');
document.querySelectorAll('.button').forEach(button => {
  button.addEventListener('click', function() {
    button.innerHTML = 'Loading...';
    button.disabled = true;
      if (this.disabled) {
    //  event.preventDefault(); // Prevent any button action
      alert("First pass the entity check"); // Show a message
    }

  });
});

document.getElementById('protocol-icmp').addEventListener('change', function() {
  document.getElementById('port').disabled = this.checked;
});

document.getElementById('protocol-tcp').addEventListener('change', function() {
  document.getElementById('port').disabled = false;
});

document.getElementById('protocol-udp').addEventListener('change', function() {
```

```
  document.getElementById('port').disabled = false;
});

// To handle form submission
document.getElementById('ruleForm').addEventListener('submit', function(event) {
  event.preventDefault();
  const srcIp = document.getElementById('src-ip').value;
  const dstIp = document.getElementById('dst-ip').value;
  const port = document.getElementById('port').value;
  const protocol =
document.querySelector('input[name="protocol"]:checked').value;

  fetch('http://127.0.0.1:5000/api/run_playbook', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ src_ip: srcIp, dst_ip: dstIp, port: port, protocol:
protocol })
  })
      .then(response => response.json())
  .then(data => {
    if (data.status === 'success') {
      alert(data.message);
      localStorage.setItem('lastRecordId', data.db_record._id);
            document.getElementById('runPlaybook').disabled = false;
    } else {
      alert(data.message);
            document.getElementById('runPlaybook').disabled = true;
    }
  })
  .catch(error => {
    console.error('Error:', error);
        document.getElementById('runPlaybook').disabled = true;
  });
});
document.getElementById('runPlaybook').addEventListener('click', function() {
  const recordId = localStorage.getItem('lastRecordId'); // Retrieve the record
ID from storage
  if (!recordId) {
    alert('No record selected or check not performed.');
```

```
    return;
  }
  fetch('http://127.0.0.1:5000/api/runPlaybook', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ record_id: recordId })
  })
  .then(response => response.json())
  .then(data => {
    if (data.status === "success") {
      alert('Rule created successfuly!');
    } else {
      alert('Failed to create rule: ' + data.message);
    }
  })
  .catch(error => {
    console.error('Error while creating rule:', error);
    alert('Failed to create rule.');
  });
});
document.getElementById('checkService').addEventListener('click', function() {
  const serviceName = document.getElementById('service-name').value;
  const servicePort = document.getElementById('service-port').value;
  const serviceType = document.querySelector('input[name="service-
type"]:checked').value;

  // Prepend protocol type to the service port if the type is TCP
  const formattedPort = serviceType === 'tcp' ? `tcp_${servicePort}` :
servicePort;

  fetch('http://127.0.0.1:5000/api/checkService', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
  body: JSON.stringify({ serviceName, servicePort: formattedPort, serviceType })
  })
  .then(response => response.json())
  .then(data => {
```

```
    alert(data.message);
    if (data.message.includes("Service found")) {
      document.getElementById('createService').disabled = true; // Enable the
create button if the service is found
    } else {
      document.getElementById('createService').disabled = false; // Keep the
button disabled if the service is not found
    }
  })
  .catch(error => {
    console.error('Error:', error);
    document.getElementById('createService').disabled = true; // Disable the
button if there's an error
  });
});

document.getElementById('createService').addEventListener('click', function() {
  const serviceName = document.getElementById('service-name').value;
  const servicePort = document.getElementById('service-port').value;
  const      serviceType      =      document.querySelector('input[name="service-
type"]:checked').value;

  fetch('http://127.0.0.1:5000/api/createService', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ serviceName, servicePort, serviceType })
  })
  .then(response => response.json())
  .then(data => alert(data.message))
  .catch(error => console.error('Error:', error));
});

document.addEventListener('DOMContentLoaded', function() {
  const checkObjectBtn = document.getElementById('checkObject');
  const createObjectBtn = document.getElementById('createObject');

  checkObjectBtn.addEventListener('click', function() {
    const objectIp = document.getElementById('object-ip').value;
```

```
    const      objectType      =      document.querySelector('input[name="object-
type"]:checked').value;

    fetch('http://127.0.0.1:5000/api/checkObject', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        objectIp: objectIp,
        objectType: objectType
      })
    })
    .then(response => response.json())
    .then(data => {
      if (data.status === "success") {
        alert('Object found!');
            document.getElementById('createObject').disabled = true; // Enable
the create button if the service is found
      } else {
        alert('Failed to find object: ' + data.message);
            document.getElementById('createObject').disabled = false; // Enable
the create button if the service is found
      }
    })
    .catch(error => {
      console.error('Error checking object:', error);
      alert('Failed to check object. See console for details.');
        document.getElementById('createObject').disabled = true; // Enable the
create button if the service is found
    });
  });

  createObjectBtn.addEventListener('click', function() {
    const objectName = document.getElementById('object-name').value;
    const objectIp = document.getElementById('object-ip').value;
    const      objectType      =      document.querySelector('input[name="object-
type"]:checked').value;
    let extraVars = {
      objectName: objectName,
      objectIp: objectIp,
```

```
      objectType: objectType
    };

    if (objectType === 'network') {
      const objectMask = document.getElementById('object-mask').value;
      extraVars.objectMask = objectMask;
    }

    fetch('http://127.0.0.1:5000/api/createObject', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(extraVars)
    })
    .then(response => response.json())
    .then(data => {
      if (data.status === "success") {
        alert('Object created successfully.');
      } else {
        alert('Failed to create object: ' + data.message);
      }
    })
    .catch(error => {
      console.error('Error creating object:', error);
      alert('Failed to create object. See console for details.');
    });
  });

  // Optionally toggle visibility of subnet mask input based on object type
  document.querySelectorAll('input[name="object-type"]').forEach(radio => {
    radio.addEventListener('change', function() {
      const networkMaskGroup = document.getElementById('network-mask-group');
      if (this.value === 'network') {
        networkMaskGroup.style.display = 'block';
      } else {
        networkMaskGroup.style.display = 'none';
      }
    });
  });
});
```

```
document.addEventListener('DOMContentLoaded', function() {
  document.getElementById('createInstanceForm').addEventListener('submit',
function(event) {
    event.preventDefault();
    const instanceName = document.getElementById('instance-name').value;
    const instanceType = document.getElementById('instance-type').value;
    const osType = document.getElementById('os-type').value;

    fetch('http://127.0.0.1:5000/api/create_aws_instance', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ instanceName, instanceType, osType })
    })
    .then(response => response.json())
    .then(data => alert(data.message))
    .catch(error => console.error('Error:', error));
  });
});
      function submitVrfForm() {
  const vrfData = {
    vrfName: document.getElementById('vrf-name').value,
    vrfDescription: document.getElementById('vrf-description').value,
    interfaceName: document.getElementById('interface-name').value,
    subintId: document.getElementById('subint-id').value,
    vlanId: document.getElementById('vlan-id').value,
    ipAddress: document.getElementById('ip-address').value,
    subnetMask: document.getElementById('subnet-mask').value
  };
  // AJAX request to send data to the server
  fetch('http://127.0.0.1:5000/api/create-vrf', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json'
      },
    body: JSON.stringify(vrfData)
  })
  .then(response => response.json())
```

```
  .then(data => {
    if (data.status === 'success') {
      alert('VRF creation successful!');
    } else {
      alert('Error: ' + data.message);
    }
  })
  .catch(error => {
    console.error('Error creating VRF:', error);
  });
}
function generateConfig() {
  const vrfName = document.getElementById('vrf-name').value;
  const vrfDescription = document.getElementById('vrf-description').value;
  const interfaceName = document.getElementById('interface-name').value;
  const subintId = document.getElementById('subint-id').value;
  const vlanId = document.getElementById('vlan-id').value;
  const ipAddress = document.getElementById('ip-address').value;
  const subnetMask = document.getElementById('subnet-mask').value;

  fetch('/vrf_config_template.txt')
    .then(response => response.text())
    .then(template => {
      const configText = template
        .replace('{{ vrf_name }}', vrfName)
        .replace('{{ vrf_description }}', vrfDescription)
        .replace('{{ interface_name }}', interfaceName)
        .replace('{{ subint_id }}', subintId)
        .replace('{{ vlan_id }}', vlanId)
        .replace('{{ ip_address }}', ipAddress)
        .replace('{{ subnet_mask }}', subnetMask);

      // Here, add code to display or download configText as needed
    alert(configText); // Example action
  const configBlob = new Blob([configText], {type: 'text/plain'});
  // Create a URL for the Blob
  const configUrl = window.URL.createObjectURL(configBlob);
  // Create a temporary anchor element and trigger the download
  const downloadLink = document.createElement('a');
  downloadLink.href = configUrl;
```

```
  downloadLink.download = `${vrfName}_config.txt`; // Set the file name for the
download
  document.body.appendChild(downloadLink);
  downloadLink.click();
  document.body.removeChild(downloadLink);

});
}
```

## Код файлу vrf_config_template

```
vrf definition {{ vrf_name }}
 description {{ vrf_description }}
!
interface {{ interface_name }}.{{ subint_id }}
 encapsulation dot1Q {{ vlan_id }}
 ip vrf forwarding {{ vrf_name }}
 ip address {{ ip_address }} {{ subnet_mask }}
!
```

## Код Flask бекенду

```
from flask import Flask, request, jsonify, redirect, url_for, render_template,
session
import subprocess
import re
from flask_cors import CORS
from pymongo import MongoClient
from datetime import datetime
import uuid
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required, LoginManager
from bson import ObjectId
import bson.errors
from flask_login import LoginManager, UserMixin, login_user, logout_user,
login_required
from flask.sessions import SecureCookieSessionInterface
from functools import wraps
from flask import redirect, url_for
from datetime import timedelta
from flask_session import Session
import gitlab
```

```
import os
import json

app = Flask(__name__)
CORS(app, supports_credentials=True)

# Setup MongoDB connection
client = MongoClient('mongodb://localhost:27017/') # Adjust connection string if
needed
db = client['mydatabase'] # Use or create a database
records = db.records # Use or create a collection
users = db.users

def run_ansible_playbook(playbook, extra_vars):
  """Run an Ansible playbook with extra variables and return stdout."""
  try:
    command = ['ansible-playbook', playbook, '-e', extra_vars]
    result = subprocess.run(command, capture_output=True, text=True, check=True)
    return result.stdout # Ensure only stdout is returned, which should be a
string
  except subprocess.CalledProcessError as e:
    # Return stderr for error analysis; you might adjust this based on your needs
    return e.stderr
  except Exception as e:
    # Catch-all for any other exception, return the exception message
    return str(e)




def run_ansible_playbook_create_service(playbook, extra_vars):
  """Run an Ansible playbook with the given extra variables."""
  try:
    command = ['ansible-playbook', playbook, '-e', extra_vars]
    result = subprocess.run(command, capture_output=True, text=True)
    return result.returncode, result.stdout # Return both return code and stdout
  except subprocess.CalledProcessError as e:
    # Handle cases where the playbook fails
    return e.returncode, e.stderr # Return the error code and stderr
  except Exception as e:
    # Handle other exceptions
```

```
    return 1, str(e) # Return a non-zero error code and error message


def parse_ansible_output(output):
  """Parse the output from Ansible to extract the UID."""
  pattern                                                                    =
r'\{\s*"objects_result\.ansible_facts\.objects\.objects":\s*\[\s*"([\w-
]+)"\s*\]\s*\}'
  match = re.search(pattern, output)
  return match.group(1) if match else None

@app.route('/api/checkObject', methods=['POST'])
#@cross_origin(supports_credentials=True)
def check_object():
  data = request.get_json()
  object_filter = data['objectIp']
  object_type = data['objectType']

  # Build command to run the playbook with dynamic inventory and extra vars
  command = [
    'ansible-playbook',
    'FindObject_Playbook.yml',
    '-e', f"filter_ip={object_filter} type={object_type}"
  ]

  try:
    result = subprocess.run(command, capture_output=True, text=True, check=True)
    # Parse output to find object details or determine if no object found
    object_details = parse_ansible_output(result.stdout)
    if object_details:
      return jsonify({"status": "success", "message": "Object found.", "details":
object_details})
    else:
      return jsonify({"status": "error", "message": "Object not found."})
  except subprocess.CalledProcessError as e:
    return jsonify({"status": "error", "message": "Failed to check object",
"error": str(e)}), 500

@app.route('/api/createObject', methods=['POST'])
def create_object():
  data = request.get_json()
```

```
  object_name = data['objectName']
  object_ip = data['objectIp']
  object_type = data['objectType']
  playbook_path = "CreateHost_Playbook.yml" if object_type == 'host' else
"CreateNetwork_Playbook.yml"

  # For network objects, also pass subnet mask
  extra_vars = f"filter_name={object_name} filter_ip={object_ip}"
  if object_type == 'network':
    object_mask = data['objectMask']
    extra_vars += f" filter_mask={object_mask}"

  command = [
    'ansible-playbook',
    playbook_path,
    '-e', extra_vars
  ]

  try:
    result = subprocess.run(command, capture_output=True, text=True, check=True)
    if result.returncode == 0:
      return jsonify({"status": "success", "message": "Object created
successfully."})
    else:
      return jsonify({"status": "error", "message": "Failed to create object",
"error": result.stderr}), 400
  except subprocess.CalledProcessError as e:
    return jsonify({"status": "error", "message": str(e)}), 500

@app.route('/api/runPlaybook', methods=['POST'])
def run_firewall_playbook():
  data = request.get_json()
  record_id = data.get('record_id')

  if not record_id:
    return jsonify({"status": "error", "message": "Record ID is required"}), 400

  record = records.find_one({'_id': record_id}) # Use the UUID directly without
conversion

  if not record:
```

```
    return jsonify({"status": "error", "message": "Record not found"}), 404

  playbook_vars = f"filter_src={record['src_uid']} filter_dst={record['dst_uid']}
filter_service={record['service_uid']} filter_name={record['_id']}"

  try:
    result = subprocess.run(
      ['ansible-playbook', 'InstallRule_Playbook.yml', '-e', playbook_vars],
      capture_output=True, text=True,
      check=True
    )
    return   jsonify({"status":   "success",   "message":   "Playbook   executed
successfully", "output": result.stdout})
  except subprocess.CalledProcessError as e:
    return jsonify({"status": "error", "message": "Playbook execution failed",
"error": e.stderr}), 500
  except Exception as e:
    return jsonify({"status": "error", "message": str(e)}), 500



@app.route('/api/run_playbook', methods=['POST'])
def run_playbook():
  data = request.json
  src_ip = data['src_ip']
  dst_ip = data['dst_ip']
  protocol_type = data['protocol'] # Assuming the protocol (TCP/UDP) is also sent
  port_name = f"{protocol_type}_{data['port']}"
  #port_name = data['port']
  port_type = f"service-{data['protocol']}" if data['protocol'] in ['tcp', 'udp']
else data['protocol']

  # Running playbooks based on IP and port
  source_uid                                                                    =
parse_ansible_output(run_ansible_playbook('FindObject_Playbook.yml',
f"filter_ip={src_ip}"))
  destination_uid                                                               =
parse_ansible_output(run_ansible_playbook('FindObject_Playbook.yml',
f"filter_ip={dst_ip}"))
```

```
  service_uid                                                                    =
parse_ansible_output(run_ansible_playbook('FindService_Playbook.yml',
f"filter_port={port_name} filter_type={port_type}"))

    # Check if the UIDs exist (dummy function for example)
  source_exists = "Exist" if source_uid else "Not exist"
  destination_exists = "Exist" if destination_uid else "Not exist"
  service_exists = "Exist" if service_uid else "Not exist"

  all_exist = source_uid and destination_uid and service_uid # Only true if all
are found

  # Record creation with a unique identifier and timestamp
  record = {
    #'_id': str(uuid.uuid4()), # Generate a unique identifier
    '_id': str(uuid.uuid4()).replace('-', ''),
    'src_ip': src_ip,
    'src_uid': source_uid,
    'src_type': data.get('src_type', 'host'), # Default to 'host' if not specified
    'dst_ip': dst_ip,
    'dst_uid': destination_uid,
    'dst_type': data.get('dst_type', 'host'), # Default to 'host'
    'port': f"{port_type}_{port_name}",
    'service_uid': service_uid,
    'created_at': datetime.now() # Timestamp of creation
  }

  # Insert the record into MongoDB
#  records.insert_one(record)

  # Return the UIDs found along with a success message or the record itself
  if all_exist:
    records.insert_one(record) # Assume records is your MongoDB collection
    return jsonify({
      'status': 'success',
      'message': 'All entities exist.',
      'db_record': record
    })
  else:
    return jsonify({
      'status': 'error',
```

```
      'message': 'One or more entities do not exist.',
      'source': source_exists,
      'destination': destination_exists,
      'service': service_exists
    })

@app.route('/api/createService', methods=['POST'])
def create_service():
  data = request.get_json()
  filter_name = data['serviceName']
  filter_port = data['servicePort']
  filter_type = data['serviceType']
  playbook = 'CreateServiceTCP_Playbook.yml' if filter_type == 'tcp' else
'CreateServiceUDP_Playbook.yml'

  # run_ansible_playbook now returns both return code and output
  return_code, output = run_ansible_playbook_create_service(playbook,
f"filter_name={filter_name} filter_port={filter_port}")

  if return_code == 0:
    message = "Service created successfully"
  else:
    message = f"Failed to create service: {output}"

  return jsonify({'message': message})


@app.route('/api/checkService', methods=['POST'])
def check_service():
  data = request.get_json()
  filter_port = data['servicePort']
  filter_type = f"service-{data['serviceType']}"

  # This function now correctly returns a string
  output = run_ansible_playbook('FindService_Playbook.yml',
f"filter_port={filter_port} filter_type={filter_type}")

  object_uid = parse_ansible_output(output)
  message = "Service found!" if object_uid else "Service not found"

  return jsonify({'message': message})
```

```
# Setup GitLab client
gl                    =                    gitlab.Gitlab('https://gitlab.com/',
private_token=os.environ['GITLAB_TOKEN'])

@app.route('/api/create_aws_instance', methods=['POST'])
def create_aws_instance():
  data = request.get_json()
  instance_name = data['instanceName']
  instance_type = data['instanceType']
  os_type = data['osType']

  # Map OS type to AMI
  ami_map = {
    "ubuntu": "ami-023adaba598e661ac",
    "amazon_linux": "ami-0f673487d7e5f89ca",
    "windows": "ami-0847a7983fdc60e79",
    "redhat": "ami-0134dde2b68fe1b07"
  }
  ami = ami_map.get(os_type.lower())

  # Generate the Terraform configuration
  terraform_config = f"""
resource "aws_instance" "{instance_name}" {{
 ami      = "{ami}"
 instance_type = "t2.micro"
}}
"""
  print (terraform_config)
  # File path and branch name generation
  branch_name = f'create-instance-{uuid.uuid4()}'
  file_path = f'terraform/Cloud/{instance_name}.tf'


  try:
    project = gl.projects.get('shedovich/automation')
    print (project)
    # Create a new branch
    project.branches.create({'branch': branch_name, 'ref': 'main'})

    print (project)
```

```
    # Commit the changes
    project.files.create({
      'file_path': file_path,
      'branch': branch_name,
      'content': terraform_config,
      'author_email': 'you@example.com',
      'author_name': 'Name',
      'commit_message': f'Add new instance {instance_name}'
    })

    # Create a merge request
    mr = project.mergerequests.create({
      'source_branch': branch_name,
      'target_branch': 'main',
      'title': f'Create new instance {instance_name}',
    })


    return jsonify({'message': 'Merge request created successfully!', 'mr_url':
mr.web_url})
  except Exception as e:
    return jsonify({'message': 'Failed to create instance', 'error': str(e)}),
500


@app.route('/api/create-vrf', methods=['POST'])
def create_vrf():
  data = request.get_json()
  try:
    # Extracting VRF and subinterface details from the request
    vrf_name = data['vrfName']
#    vrf_rd = '100:1'
    vrf_description = data['vrfDescription']
    interface_name = data['interfaceName']
    subint_id = data['subintId']
    vlan_id = data['vlanId']
    ip_address = data['ipAddress']
    subnet_mask = data['subnetMask']

    # Constructing extra vars for Ansible
```

```
    extra_vars = {
      "vrf_name": vrf_name,
      "vrf_description": vrf_description,
      "interface_configs": [
        {
          "interface": interface_name,
          "subint_id": subint_id,
          "vlan_id": vlan_id,
          "ip_address": ip_address,
          "subnet_mask": subnet_mask
        }
      ]
    }

    # Running the Ansible playbook with the provided details
    playbook_output = run_ansible_playbook('CreateVRF_Playbook.yml',
json.dumps(extra_vars))
    return jsonify({'status': 'success', 'message': 'VRF created successfully',
'ansible_output': playbook_output})
  except Exception as e:
    return jsonify({'status': 'error', 'message': str(e)}), 500


if __name__ == '__main__':
  app.run(debug=True, host='0.0.0.0')
```

## Код файлу ansible.cfg

```
[defaults]

inventory=hosts

host_key_checking=false
retry_files_enabled=false
interpreter_python=/usr/bin/python3
```

## Код файлу hosts

```
[smc]
192.168.1.200
```

```
[routers]
192.168.1.10

[routers:vars]
ansible_user=admin
ansible_password=admin123
adnsible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.ios.ios


[smc:vars]
ansible_httpapi_use_ssl=true
ansible_httpapi_validate_certs=false
ansible_user=ansible
ansible_password=ansible
ansible_network_os=check_point.mgmt.checkpoint


[routers:vars]
ansible_user=admin
ansible_password=admin123
adnsible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.ios.ios
```

## Код файлу FindObject_Playbook.yml

```
---
- name: "Find object"
 hosts: smc
 gather_facts: no
 connection: httpapi
 vars:
  object_filter: "{{ filter_ip }}"
 tasks:
  - name: "find object"
   check_point.mgmt.cp_mgmt_objects_facts:
    filter: "{{ object_filter }}"
    ip_only: true
    type: host
    details_level: uid
   register: objects_result
```

```
  - name: "Print objects"
    debug:
      var: objects_result.ansible_facts.objects.objects
```

Код файлу CreateHost_Playbook.yml

```
---
- name: "Add host object"
 hosts: smc
 gather_facts: no
 connection: httpapi
 vars:
  object_name: "{{ filter_name }}"
  object_ip: "{{ filter_ip }}"

 tasks:

  - name: "add host object"
    check_point.mgmt.cp_mgmt_host:
      name: "{{ object_name }}"
      color: pink
      ipv4_address: "{{ object_ip }}"
      auto_publish_session: true
```

Код файлу CreateNetwork_Playbook.yml

```
---
- name: "Add network object"
 hosts: smc
 gather_facts: no
 connection: httpapi
 vars:
  object_name: "{{ filter_name }}"
  object_ip: "{{ filter_ip }}"
  object_mask: "{{ filter_mask }}"

 tasks:

  - name: "add network object"
```

```
   check_point.mgmt.cp_mgmt_network:
    name: "{{ object_name }}"
    color: pink
    subnet4: "{{ object_ip }}"
    subnet_mask: "{{ object_mask }}"
    auto_publish_session: true
```

Код файлу InstallRule_Playbook.yml

```
---
- name: "Add acces rule"
 hosts: smc
 gather_facts: no
 connection: httpapi
 vars:
  object_src: "{{ filter_src }}"
  object_dst: "{{ filter_dst }}"
  object_service: "{{ filter_service }}"
  object_name: "{{ filter_name }}"
 tasks:

  - name: "add rule"
   check_point.mgmt.cp_mgmt_access_rule:
    layer: network
    enabled: true
    name: "{{ object_name }}"
    relative_position:
     top: "Automation"
    source: "{{ object_src }}"
    destination: "{{ object_dst }}"
    service: "{{ object_service }}"
    action: accept
    auto_publish_session: true
    install_on: "CP_Cluster_1"
```

Код файлу FindObject_Playbook.yml

```
---
- name: "Find object"
 hosts: smc
 gather_facts: no
 connection: httpapi
```

```
 vars:
  object_filter: "{{ filter_ip }}"
 tasks:
  - name: "find object"
   check_point.mgmt.cp_mgmt_objects_facts:
    filter: "{{ object_filter }}"
    ip_only: true
    type: host
    details_level: uid
   register: objects_result

  - name: "Print objects"
   debug:
    var: objects_result.ansible_facts.objects.objects
```

Код файлу FindService_Playbook.yml

```
---
- name: "Find service"
 hosts: smc
 gather_facts: no
 connection: httpapi
 vars:
  object_filter: "{{ filter_port }}"
  object_type: "{{ filter_type }}"
 tasks:
  - name: "find object"
   check_point.mgmt.cp_mgmt_objects_facts:
    filter: "{{ object_filter }}"
    type: "{{ object_type }}"
    details_level: uid
   register: objects_result

  - name: "Print objects"
   debug:
    var: objects_result.ansible_facts.objects.objects
```

Код файлу CreateServiceTCP_Playbook.yml

```
---
- name: "Add service TCP"
 hosts: smc
```

```
 gather_facts: no
 connection: httpapi
 vars:
  object_name: "{{ filter_name }}"
  object_port: "{{ filter_port }}"

 tasks:

  - name: "add service tcp"
    check_point.mgmt.cp_mgmt_service_tcp:
     name: "{{ object_name }}"
     color: pink
     port: "{{ object_port }}"
     auto_publish_session: true
```

Код файлу CreateServiceUDP_Playbook.yml

```
---
- name: "Add service UDP"
 hosts: smc
 gather_facts: no
 connection: httpapi
 vars:
  object_name: "{{ filter_name }}"
  object_port: "{{ filter_port }}"

 tasks:

  - name: "add service udp"
    check_point.mgmt.cp_mgmt_service_udp:
     name: "{{ object_name }}"
     color: pink
     port: "{{ object_port }}"
     auto_publish_session: true
```

Код файлу CreateVRF_Playbook.yml

```
---
- name: Configure VRF and subinterfaces on Cisco IOS devices
 hosts: routers
 gather_facts: no
 connection: network_cli
```

```
 tasks:
  - name: Create VRF on the device
    cisco.ios.ios_vrf:
     name: "{{ vrf_name }}"
     rd: "100:1"
     description: "{{ vrf_description }}"
     state: present
    become: yes
    become_method: enable

  - name: Configure subinterfaces within the VRF
    cisco.ios.ios_config:
     lines:
      - "encapsulation dot1Q {{ item.vlan_id }}"
      - "ip vrf forwarding {{ vrf_name }}"
      - "ip address {{ item.ip_address }} {{ item.subnet_mask }}"
     parents: "interface {{ item.interface }}.{{ item.subint_id }}"
    loop: "{{ interface_configs }}"
    become: yes
    become_method: enable
```

## Код файлу .gitlab-ci.yml

```
---
workflow:
  rules:
   - if: $CI_COMMIT_BRANCH == "main" # This will allow the pipeline to run on main, but jobs within need to be triggered manually.
   - if: $CI_PIPELINE_SOURCE == "merge_request_event" # Allows automatic runs for merge requests for validation and planning.
   - when: never # Prevents automatic execution in other scenarios.
variables:
 TF_DIR: ${CI_PROJECT_DIR}/terraform
 STATE_NAME: "shedovich-tf"
 ADDRESS: "https://gitlab.com/api/v4/projects/${CI_PROJECT_ID}/terraform/state/${STATE_NAME}"
stages:
 - validate
 - plan
 - apply
```

```
 - destroy
image:
 name: hashicorp/terraform:light
 entrypoint: [""]
before_script:
 - terraform --version
 - export GITLAB_ACCESS_TOKEN=$TOKEN
 - cd ${TF_DIR}
 - terraform init -reconfigure -backend-config="address=${ADDRESS}" -backend-
config="lock_address=${ADDRESS}/lock" -backend-
config="unlock_address=${ADDRESS}/lock" -backend-config="username=<UseName>" -
backend-config="password=$GITLAB_ACCESS_TOKEN" -backend-
config="lock_method=POST" -backend-config="unlock_method=DELETE" -backend-
config="retry_wait_min=5"
validate:
 stage: validate
 script:
  - terraform validate
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull-push
plan:
 stage: plan
 script:
  - terraform plan
 dependencies:
  - validate
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull
apply:
 stage: apply
 script:
  - terraform apply -auto-approve
 rules:
  - if: '$CI_COMMIT_BRANCH == "main"'
   when: manual
```

```
 dependencies:
  - plan
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull
 when: manual
destroy:
 stage: destroy
 script:

  - terraform destroy -auto-approve
 rules:
  - if: '$CI_COMMIT_BRANCH == "main"'
   when: manual
 dependencies:
  - plan
  - apply
 cache:
  key: ${CI_COMMIT_REF_NAME}
  paths:
  - ${TF_DIR}/.terraform
  policy: pull
 when: manual
```

## Код файлу main.tf

```
module "cloud_resources" {  source = "./Cloud"  # Include any necessary variables here }
```

## Код файлу provider.tf

```
terraform {
required_providers {
aws = {
source = "hashicorp/aws"
version = "~> 3.27"
}}
backend "http" {} }
# Configure and downloading plugins for aws
```

```
provider "aws" {
region   = "${var.aws_region}"
}
```

## Код файлу variables.tf

```
# Defining Region
variable "aws_region" {
default = "eu-central-1" }
# Defining CIDR Block for VPC
variable "vpc_cidr" {
default = "10.0.0.0/16" }
```

## Код файлу Ubuntu_test_instance.tf

```
resource "aws_instance" "Ubuntu_test_instance" {
ami      = "ami-023adaba598e661ac"
instance_type = "t2.micro" }
```