ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка веб-додатку для налаштування та управління мережевих комутаторів мовою Python»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Дмитро ТИЩЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПП3-51

Дмитро ТИЩЕНКО

Керівник: Віктор ГРЕБЕНЮК
доктор-філософії (PhD)

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Тищенку Дмитру Станіславовичу

1. Тема кваліфікаційної роботи: «Розробка веб-додатку для налаштування та управління мережевих комутаторів мовою Python»
керівник кваліфікаційної роботи доктор філософії (PhD), доцент кафедри ІПЗ Віктор ГРЕБЕНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні та практичні навички налаштування мережевих комутаторів, розробки веб-додатків з використанням технічної документації Python та Django, розуміння принципів клієнт-серверної архітектури та досвід роботи з протоколом Telnet.

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

1. Аналіз галузі мережевого адміністрування та програмного забезпечення для автоматизації процесів управління
2. Огляд засобів реалізації веб-додатку для налаштування та управління мережевими комутаторами

3. Опис розробки веб-додатка для налаштування та управління мережевими комутаторами
4. Тестування веб-додатка.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до веб-додатка.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма класів
6. Схема бази даних
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих рішень для автоматизації та протоколів керування комутаторами, засобів та бібліотек для мережевої автоматизації	14.03-20.03.2024	
4	Проєктування веб-додатка для налаштування та управління комутаторами	21.03-01.04.2024	
5	Програмна реалізація веб-додатка	02.04-25.04.2024	
6	Тестування веб-додатка	26.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Дмитро ТИЩЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ГРЕБЕНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 54 стор., 2 табл., 32 рис., 18 джерел.

Мета роботи – спростити процес налаштування та керування мережевими комутаторами шляхом використання веб-додатку.

Об'єкт дослідження – процес налаштування та адміністрування мережевих комутаторів.

Предмет дослідження – програмне забезпечення та засоби для керування та налаштування комутаторів.

Короткий зміст роботи: В роботі проведено дослідження галузі мережевого адміністрування, розглянуті основні протоколи для взаємодії з комутаторами. Проаналізовано програмні засоби для автоматизації процесу управління комутаторами: SolarWinds Network Automation Manager, Cisco DNA, ManageEngine NCM, Red Hat Ansible Automation Platform. Розроблено архітектуру веб-додатку та програмно реалізовані ключові функціональні можливості, зокрема: додавання, видалення та редагування комутаторів, виконання сценаріїв налаштування та керування ними, отримання інформації про стан комутаторів, налаштування VLAN та портів, а також реалізовано журнал подій. Проведено функціональне тестування веб-додатку. В роботі використано бібліотека Telnetlib для взаємодії з мережевими комутаторами, фреймворк Django для розробки веб-додатку, HTML та CSS для побудови користувацького інтерфейсу, JavaScript для динамічного відображення.

Сферою використання застосунку є спрощення та автоматизація процесу налаштування та адміністрування мережевими комутаторами.

КЛЮЧОВІ СЛОВА: МЕРЕЖЕВІ КОМУТАТОРИ, УПРАВЛІННЯ МЕРЕЖЕЮ, АВТОМАТИЗАЦІЯ, PYTHON, DJANGO, TELNET, ВЕБ-ДОДАТОК.

ЗМІСТ

ВСТУП

У сучасному світі, де мережеві технології відіграють ключову роль у забезпеченні зв'язку та передачі даних, мережеві комутатори є невід'ємною частиною будь-якої інфраструктури. Зі зростанням складності та масштабів мереж, управління ними стає все більш трудомістким та вимагає значних зусиль від мережевих адміністраторів. Ручне налаштування кожного комутатора окремо є не тільки затратним за часом, але й може привести до простоїв в результаті помилок інженерів. Тому автоматизація налаштування та управління мережевими комутаторами є актуальним питанням сьогодення.

Мережеві комутатори другого рівня (L2), що працюють на канальному рівні моделі OSI, є основою багатьох мереж. Вони забезпечують передачу даних між пристроями в локальній мережі на основі MAC-адрес. Автоматизація процесів налаштування та управління L2 комутаторами дозволить значно спростити та прискорити виконання рутинних операцій, зменшити ризик виникнення помилок, пов'язаних з людським фактором, та підвищити ефективність роботи мережевої інфраструктури.

Мета роботи: спростити процес налаштування та керування мережевими комутаторами шляхом розробки веб-додатку.

Об’єкт дослідження: процес налаштування та адміністрування мережевих комутаторів.

Предмет дослідження: програмне забезпечення та засоби для керування та налаштування комутаторів.

Актуальність теми дослідження полягає у необхідності розробки інструменту, який можна легко почати використовувати, що дозволить автоматизувати процес налаштування та управління комутаторами, зменшуючи навантаження на мережевих адміністраторів та підвищуючи надійність мережі.

Очікується, що в результаті цієї роботи буде створено функціональний та зручний веб-додаток, який дозволить підвищити ефективність та надійність управління мережевими комутаторами в різних мережевих інфраструктурах.

1 АНАЛІЗ ГАЛУЗІ МЕРЕЖЕВОГО АДМІНІСТРУВАННЯ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ УПРАВЛІННЯ КОМУТАТОРАМИ

1.1 Проблеми з якими зустрічаються мережеві адміністратори

Традиційно налаштування та зміна конфігурацій на комутаторах виконується вручну за допомогою CLI (command-line interface - інтерфейс командного рядка), та згідно з дослідженнями на сьогодні все ще більша частина подібних завдань виконується саме традиційним методом.

Мережеві інженери, адміністратори та інші працівники галузі на яких покладена відповідальність керування мережевою інфраструктурою, кожен день зустрічаються з різноманітними завданнями різної складності які потребують взаємодії з мережевими пристроями [3]. Розглянемо декілька викликів з якими регулярно зустрічаються мережеві адміністратори:

- Складність в управлінні великими мережами: іноді мережі великих бізнесів або інтернет сервіс провайдерів можуть налічувати сотні, а то і тисячі комутаторів, кожен з яких має унікальні конфігурації. Керування такою кількість пристроїв вручну не тільки часозатратно, але й може призвести до збільшення кількості помилок, несправностей, та простою сервісу.
- Потреба у швидкому реагуванні на зміни: періодично може змінюватись мережева інфраструктура та бізнес потреби, або інфраструктура може зростати та з’являтись нові сегменти мережі. Адміністратори мають бути готовими швидко впровадити відповіді зміни до конфігурацій на всіх пристроях.
- Забезпечення високої надійності та стабільності: перебої в роботі мережі можуть призвести до простоїв сервісу, а відповідно і значних збитків для компаній або її клієнтів. Мережеві адміністратори мають забезпечувати

стабільну роботу мережі, моніторинг стану комутаторів та виявлення й усунення несправностей.

- Забезпечення безпеки мережі: зі зростанням різноманітних кіберзагроз, безпека є пріоритетом для всіх організації. Тому мережевим адміністраторам слід підтримувати належні конфігурації на обладнанні для запобігання потенційних атак та несанкціонованого доступу [4].

1.2 Переваги автоматизації процесу керування та налаштування комутаторами

Впровадження автоматизації управління мережевими комутаторами надає значні переваги для мережевих адміністраторів, покращує ефективність та швидкість виконання задач. Автоматизація може бути реалізована за допомогою скриптів та сценаріїв на Bash або Python, але ці методи потребують від інженерів певних навичок програмування, якими більшість не володіють. Нижче будуть наведені основні переваги автоматизації [5]:

- Зменшення помилок інженерів: однією з головних переваг автоматизації є зменшення імовірність помилок які можуть виникнути під час ручного налаштування обладнання. Автоматизація дозволяє стандартизувати процеси управління та налаштування комутаторів, що забезпечить стабільну та надійну роботу мережі.
- Покращення продуктивності: завдяки автоматизації, мережеві адміністратори замість виконання одноманітних рутинних задач можуть сконцентруватись на більш важливих завданнях в управлінні мережею та плануванні. Вона забезпечує краще використання часу та підвищує загальну продуктивність.
- Швидке реагування на зміни: в динамічній інфраструктурі автоматизація дозволяє швидко реагувати на потреби інфраструктури, та легко і швидко впроваджувати зміни в конфігураціях мережевих комутаторів.
- Масштабованість: автоматизація забезпечує легке масштабування

операцій, це особливо важливо для великих мереж. Наприклад однакові налаштування можуть бути легко застосовані на велику групу комутаторів без суттєвих зусиль.

1.3 Перспективи зростання ринку мережевої автоматизації

Ринок мережевої автоматизації зростає та у 2024 році досяг відмітки в 25 мільярдів доларів США, очікується що в період з 2024 по 2029 роки продовжить зростання на 19.22% в рік [6]. Це пов’язано з високим попитом та можливостями які надає автоматизація мереж для оптимізації мережевих операцій та зменшення потреб в людських ресурсах та часі. Навички автоматизації мереж стають необхідними для мережевих інженерів та адміністраторів, чим надають їм перевагу та можливості бути лідерами в мережевій індустрії.

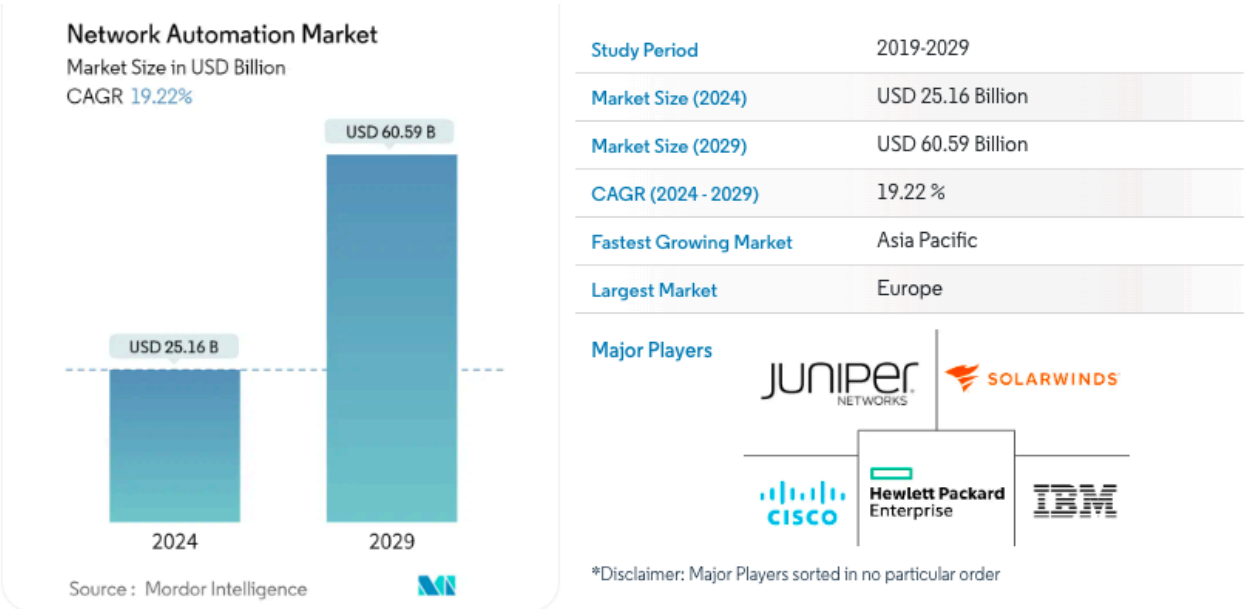


Рис. 1.1 Ринок мережевої автоматизації

1.4 Аналіз програмного забезпечення для автоматизованого керування та налаштування мережевих комутаторів

З розвитком популярності мережевої автоматизації на ринку існує багато додатків та рішень для автоматизації та управління мережевими пристроями. Ці рішення пропонують багато можливостей, надають простір для централізованого керування інфраструктурою та спрощують процеси автоматизації. Далі буде розглянуто декілька найпопулярніших застосунків.

1.4.1 SolarWinds Network Automation Manager

SolarWinds Network Automation Manager - це одне з найвідоміших рішень для автоматизацій мереж, його зазвичай використовують для великих та корпоративних мереж. Цей застосунок забезпечує мережеву автоматизацію, щоб оптимізувати керування змінами в конфігураціях дротових та бездротових пристроїв. Також в нього є функції моніторингу, аналізу трафіку, управління просторами Ip-адрес [7] [8].

З переваг застосунку можна виділити :

- Можливість моніторингу продуктивності мережі;
- Автоматизовує процес керування конфігураціями та внесення змін на комутаторах;
- Безпомилкове та швидке застосування змін;
- Резервне копіювання конфігурацій які легко відновити або скопіювати;
- Підійде для використання в масштабних середовищах з десятками тисяч вузлів.

Недоліки також є:

- Складне впровадження, для розгортання системи необхідні значні технічні навички, часто до цього залучають спеціалізовані компанії;
- Висока вартість, що може буди недоступним для середніх та малих організацій;
- Обмежена можливість інтеграцій сторонніх рішень та технологій, що

викликає залежність від інших продуктів компанії SolarWind.

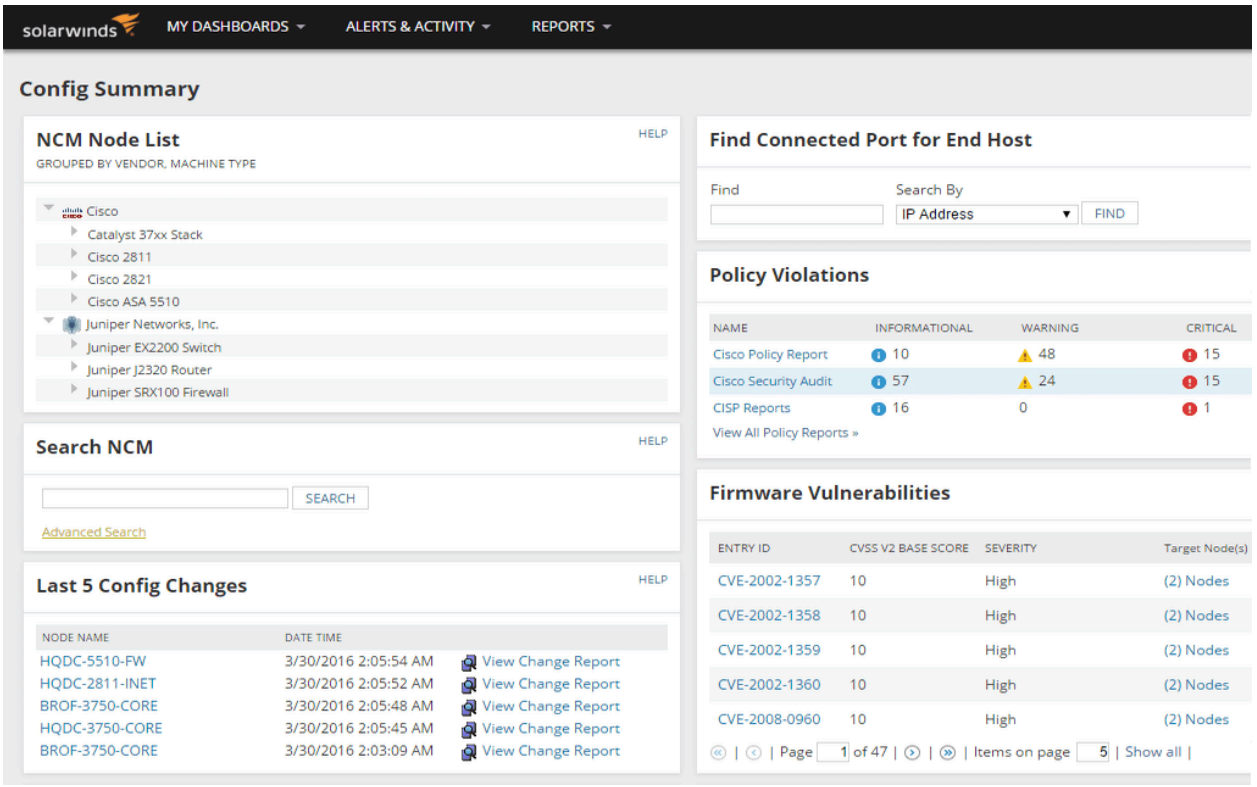


Рис. 1.2 Інтерфейсу користувача додатку SolarWinds Network Automation Manager

1.4.2 Cisco DNA (Digital Network Architecture)

Cisco DNA (Digital Network Architecture) - це рішення для побудови сучасної легко контрольованої мережевої архітектури, яке створює інтелектуальну мережу готову до змін та застосування нових технологій в міру їх зростання. Воно базується на принципах автоматизації, забезпечення безпеки, моніторингу та аналізу даних [9].

Перевагами цього рішення є:

- Автоматизація на основі заздалегідь підготовлених політик;
- Використовують AI/ML для покращення робочих процесів;
- Пропонує готові практичні ідеї;
- Широка інтеграція з іншими продуктами компанії;
- Підходить для великих мережевих середовищ;

З недоліків можна визначити:

- Вендорна залежність, підтримує тільки обладнання компанії Cisco;
- Складне впровадження, необхідно спеціалізуватись на продуктах компанії та мати значні технічні навички;
- Висока вартість, що робить його недоступним для малих та середніх бізнесів.

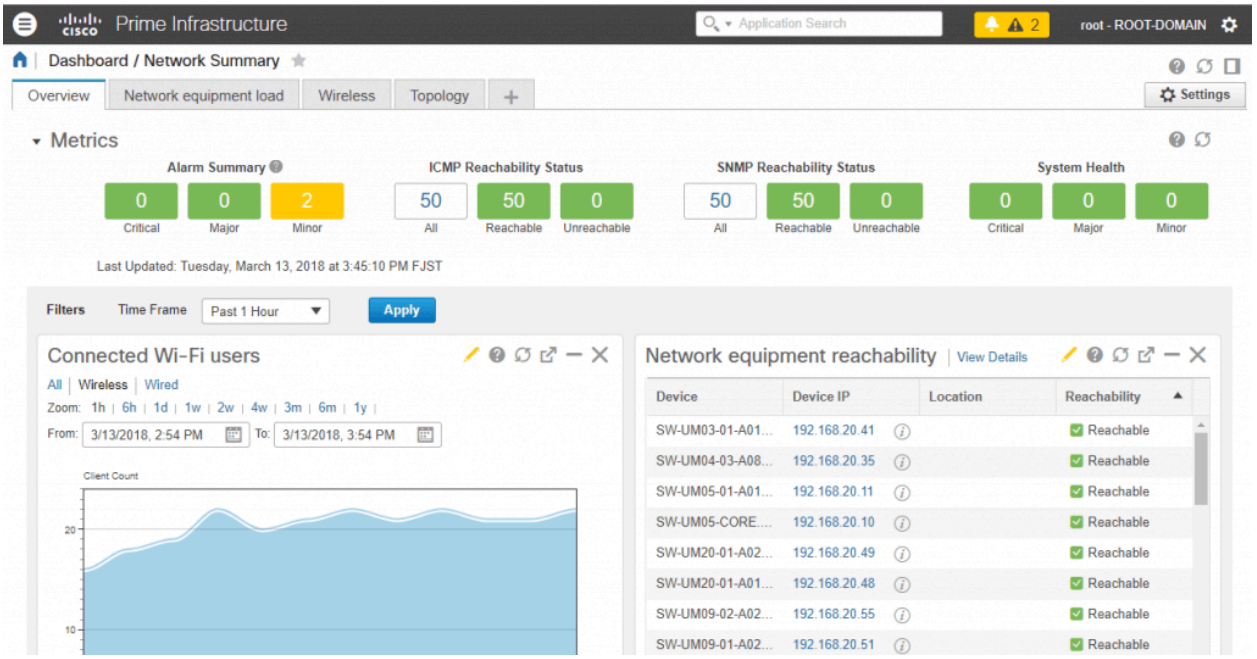


Рис. 1.3 Інтерфейсу користувача додатку Cisco DNA

1.4.3 ManageEngine NCM

ManageEngine Network Configuration Management (NCM) - це рішення для автоматизованого управління змінами в конфігураціях мережах. Надає змогу контролювати повний цикл управління конфігураціями. Спрямований на заощадження часу за допомогою автоматизації задач які постійно повторюється та забезпечити централізоване керування [10].

Перевагами застосунку є [11]:

- Можливість автоматизувати завдання різної складності;
- Вбудовані конфігурації для пристроїв різноманітних вендорів;
- Автоматичне створення, шифрування та зберігання конфігураційних

файлів;

- Відстежування та запис змін в конфігураційних файлах;
- Миттєве відновлення мережевих збоїв та механізми відновлення конфігурацій;
- Створення детальних звітів про зміни, відповідність конфігурацій, інвентаризаційні дані;

До недоліків відносяться:

- Складність в розгортанні та налаштуванні системи;
- Відсутність повноцінних можливостей моніторингу стану мережі;
- Обмежена підтримка специфічного або застарілого обладнання;
- Іноді користувачі можуть зустрічатись з недостатньою гнучкістю та обмеженнями щодо скриптів автоматизації для складних та нестандартних завдань.

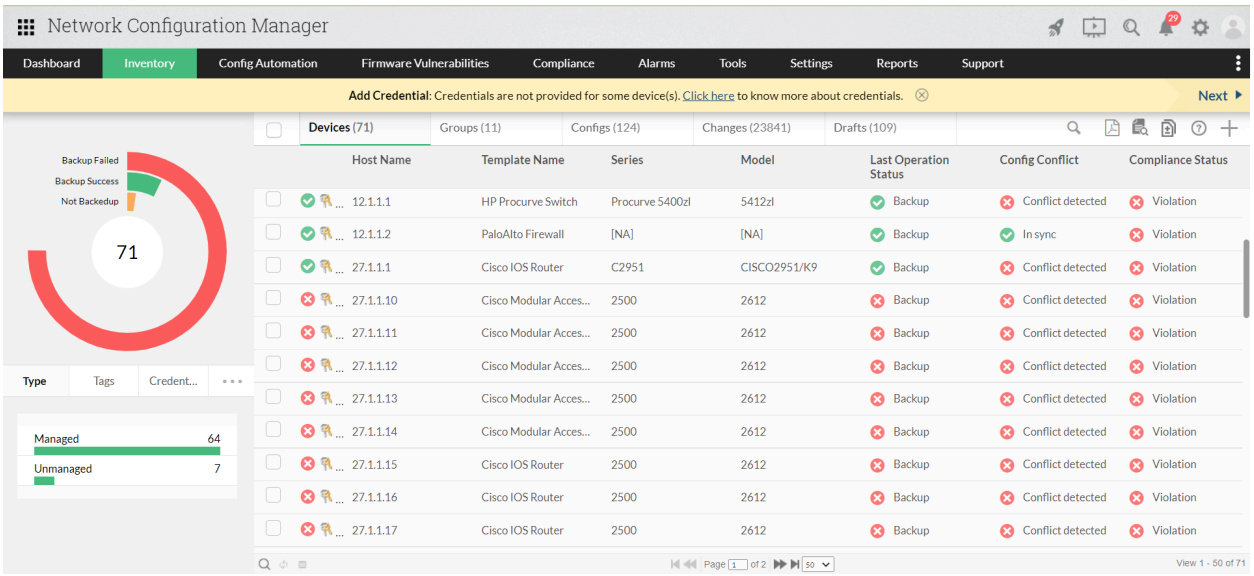


Рис. 1.4 Інтерфейсу користувача додатку ManageEngine NCM

1.4.4 Red Hat Ansible Automation Platform

Red Hat Ansible Automation Platform - це платформа для автоматизації, яка може застосовуватись в різноманітних галузях, вона надає можливість управління мережевими компонентами, серверами та багатьма іншими апаратними та програмними IT-засобами. Платформа відома своєю гнучкість, підтримкою різних модулів та можливістю інтеграції з існуючими IT-інфрструктурами. Платформа надає можливість централізовано створення та управління автоматизацією задач [12] [13].

Для мережевої автоматизації можна виділити наступні переваги платформи:

- Використовує модульний підхід що надає можливість легко налаштовувати автоматизації;
- Може виконувати керування на основі події;
- Є змога інтеграції з різними сторонніми системами та технологіями;
- Має відкритий код та велику спільному яка приймає участь в розробці продукту;

До недоліків можна віднести:

- Залежний до правильного написання плейбуків (сценаріїв), помилки можуть призвести до непередбачуваних наслідків;
- Складність використання в великих середовищах, через велику кількість плейбуків, які потребують добре організовану структуру;
- Не вміє контролювати стан, якщо зміни додаються вручну або іншими застосунками, це може призвести до неконсистентності даних;
- Деяке мережеве обладнання може мати обмежену підтримку або потребувати спеціальних модулів для управління.

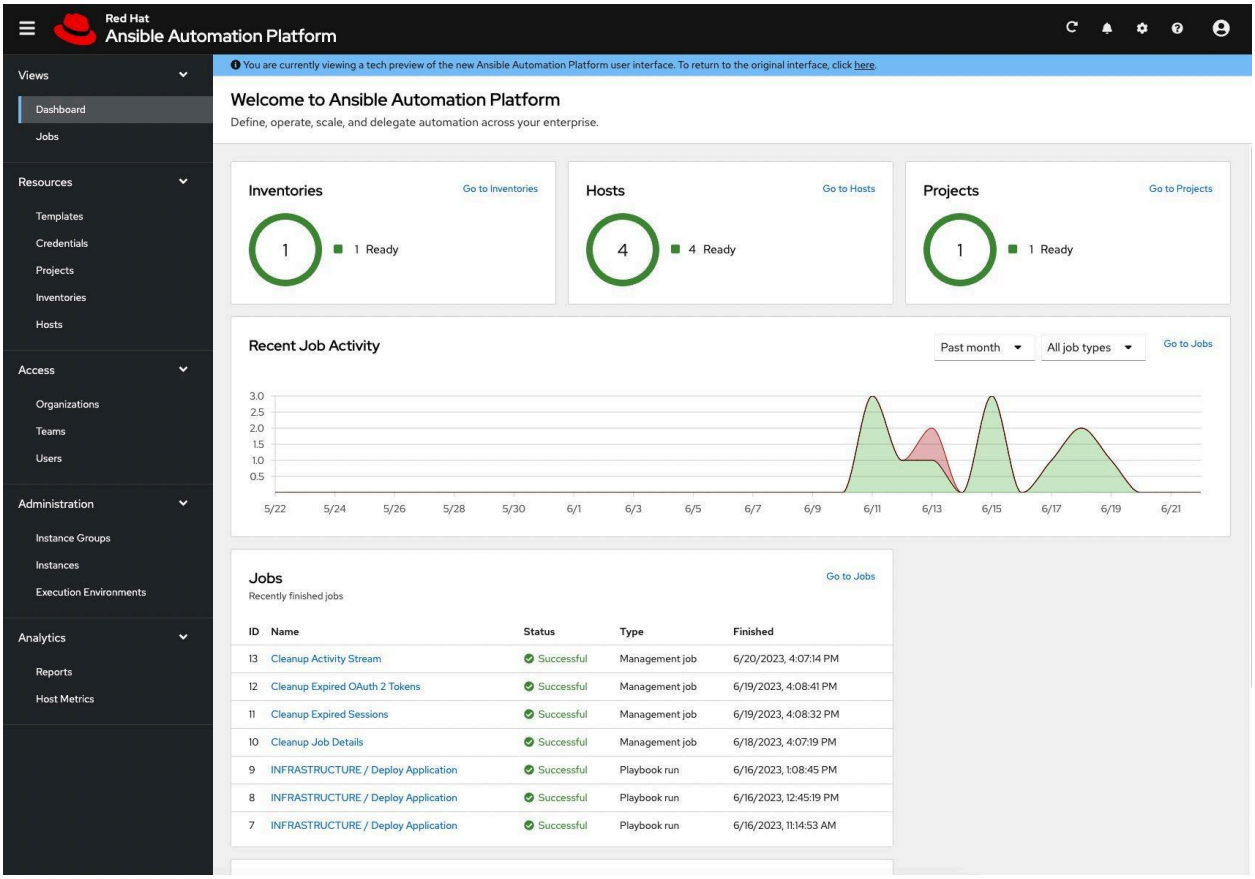


Рис. 1.5 Інтерфейсу користувача додатку Ansible Automation Platform

Таблиця 1.1

Зведені результати аналізу характеристик додатків для автоматизації та керування мережевими комутаторами

Застосунок	SolarWinds Network Automation Manager	Cisco DNA	ManageEngine Network Configuration Management	Red Hat Ansible Automation Platform	Switch Management Tool
Автоматизація конфігурацій	+	+	+	+	+
Легке впровадження в інфраструктуру	+	+	+	+	+
Мультивендорність	+	-	+	+	+
Резервне копіювання	+	-	+	-	+
Імпорт та управління сценаріями	–	-	-	+	+

2 ОГЛЯД ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ДОДАТКА ДЛЯ НАЛАШТУВАННЯ ТА УПРАВЛІННЯ МЕРЕЖЕВИМИ КОМУТАТОРАМИ

Програмне забезпечення для управління та налаштування мережевих комутаторів буде розроблятись у вигляді веб-додатку, щоб забезпечити простий та легкий доступ для пристроїв у яких є веб-браузер. Головними завданнями додатку буде конфігурація мережевих комутаторів за допомогою Telnet, також зберігання та обробка автоматизованих сценаріїв та швидких функцій.

2.1 Веб-додаток

Це програмне забезпечення яке працює на віддаленому сервері та доступне для користувачів через веб-браузер. На відміну від традиційного програмного забезпечення не потребують встановлення на пристрій та дозволяють користувачеві взаємодіяти з даними та функціями в реальному часі.

Веб-додатки мають два основні компоненти:

- Фронтенд (клієнтська сторона): відповідає за візуальне відображення та взаємодію з користувачем, зазвичай створюється за допомогою технологій HTML, CSS та JavaScript.
- Бекенд (серверна сторона): відповідає за обробку даних, виконання функцій та взаємодіє з базами даних. Може бути реалізований на різних мовах програмування, таких як PHP, Java, Python, C# та багатьох інших мовах програмування.

2.2 Вимоги до веб-додатка

Провівши аналіз процесу налаштування та управління мережевими комутаторами, а також доступних засобів для реалізації цих задач, було

сформовано функційні та не функційні вимоги до майбутнього програмного забезпечення.

Функціональні вимоги:

- Можливість авторизації користувача.
- Можливість додавати, видаляти та редагувати в системі дані про комутатори, їх ір-адресу та назву.
- Можливість збору даних з комутаторів за допомогою протоколу Telnet, виробник, модель, версія прошивки.
- Підтримка сценаріїв конфігурацій.
- Можливість імпортувати та експортувати сценарії.
- Можливість виконувати заготовлені сценарії конфігурацій на комутаторах.
- Логування подій, що відбуваються з комутаторами.

Нефункціональні вимоги:

- Забезпечити захист від несанкціонованого доступу до веб-додатку.
- Можливість працювати з різних операційних систем та браузерів.

2.3 Протоколи керування комутаторами

Для підключення до комутаторів, управління та отримання з них інформації основними протоколами є Telnet, SSH, SNMP, розглянемо їх детальніше:

- Telnet - це мережевий протокол, який використовується для управління різноманітним обладнанням. Він надає можливість з’єднуватись з пристроями та виконувати текстові команди за допомогою командного рядка (CLI), завдяки яким можна змінювати конфігурації на комутаторах та отримувати інформацію про їх стан. Головним мінусом цього протоколу є те що надсилання даних відбувається в незашифрованому вигляді, що робить його небезпечним для використання. Попри це він все ще широко використовується для адміністрування комутаторів.

- SSH (Secure Shell) - був створений як безпечна заміна протоколу Telnet, головною його відмінністю є забезпечення зашифрованого з'єднання між клієнтом та сервером, а також механізми аутентифікації, це вирішує питання безпеки та робить цей протокол більш популярним та бажаним рішенням.
- SNMP (простий протокол управління мережею) - цей протокол прикладного рівня, використовується для управління та моніторингу мережевого обладнання. З його допомогою можна зчитувати різноманітну інформацію з пристроїв, наприклад стан інтерфейсів, налаштування vlan, навантаження трафіку. Також він надає можливість змінювати конфігурації.

Вибір протоколу для керування комутаторами залежить від вимог до функціональності, безпеки та зручності використання, а також потреб інфраструктури для якої розробляється додаток. В дипломній роботі буде використовуватись протокол Telnet для взаємодії з комутаторами, оскільки це є одна з функційних вимог, а також протокол широко підтримується різними пристроями та простий у реалізації. Оскільки telnet має обмеження щодо безпеки, його слід використовувати лише в захищених мережах.

2.4 Python

Це інтерпретована високорівнева та об'єктноорієнтована мова програмування, яка розроблена Гвідо ван Россумом у 1991 році. Має простий та читабельний синтаксис, гарно підходить для використання як мова сценаріїв. Для Python створено велику кількість бібліотек та фреймворків, які охоплюють майже всі сфери розробки, від веб-додатків до машинного навчання. Відомий за свою кросплатформеність, код написаний на Python може виконуватись в різних операційних системах таких як Windows, macOS, Linux, без необхідності суттєво його змінювати. Особливістю Python також є динамічна типізація, яка

спрощує процес розробки тому як змінні визначаються під час виконання програми, та автоматичне керування пам’яттю, яке зменшує ризик помилок пов’язаних з керування пам’яттю.

2.4.1 Обґрунтування вибору Python як мови програмування

Для розробки веб-додатку було обрано мову програмування Python. Цей вибір обумовлений її високою популярністю у сфері мережевої автоматизації, що підтверджується численними дослідженнями та опитуваннями серед фахівців галузі. Завдяки великій та активній спільноті розробників, Python має величезну кількість спеціалізованих бібліотек для роботи з мережевим обладнанням та протоколами, такими як Telnetlib, Paramiko, Netmiko, Scrapli та Ansible. Вони значно спрощують та прискорюють розробку застосунків для управління мережевою інфраструктурою. Крім того, Python відрізняється простотою та читабельністю синтаксису, що полегшує написання, читання та підтримку коду. Кросплатформеність Python дозволяє розгортати веб-додаток на різних операційних системах без суттєвих змін. Враховуючи всі ці переваги, Python є оптимальним вибором для розробки веб-додатку, забезпечуючи ефективність, масштабованість та зручність використання для мережевих адміністраторів [5].

2.5 Фреймворк Django

Це високорівневий фреймворк для веброзробки, він безкоштовний та має відкритий вихідний код, написаний мовою Python та призначений для швидкого створення безпечних та масштабованих проєктів []. Фреймворк використовуючи філософію “Все включено” має багато вбудованих функцій для реалізації різноманітних намірів розробників, він має вбудовану панель адміністратора, систему автентифікації, вбудована ORM (), шаблонізатор для побудови користувацького інтерфейсу та різноманітні інші компоненти, що значно спрощує процес розробки веб-додатків та дозволяє розробника сконцентрувати

на розробці бізнес-логіці застосунку.

2.5.1 Архітектура MVT

У Django дотримується архітектура MVT (Model-View-Template), яка є одним з різновидів відомої архітектури MVC (Model-View-Controller), але замість контролера за обробку запитів та даних тут відповідає представлення.

- **Model (Модель):** Відповідає за структуру даних та їх зберігання в базі даних. Моделі Django є об'єктами Python, які представляють таблиці в базі даних.
- **View (Представлення):** Відповідає за обробку запитів від користувача, отримання та обробку даних з моделей та передачу їх до шаблонів для відображення користувачу.
- **Template (Шаблон):** Відповідає за візуальне представлення даних, отриманих від представлення. Шаблони Django використовують спеціальний синтаксис для вставки динамічних даних.

Фреймворк також заохочує до принципу "Don't Repeat Yourself" (“не повторюй себе”) що сприяє групуванню функційних можливостей та створенню коду який легко підтримується та повторно використовується.

2.5.2 Вбудована ORM

Однією з основних особливостей Django є гнучка та потужна вбудована ORM (Object-relational mapping). Вона дозволяє спрощено працювати з різноманітними базами даних за допомогою мови програмування Python, без використання складних запитів SQL. Це суттєво спрощує розробку, робить код читабельнішим та безпечнішим, також дозволяє не залежати від конкретної СУБД.

До переваг ORM Django можна віднести:

- **Продуктивність:** Вона дозволяє писати менше коду та зосередитись на

функціоналу додатку.

- **Портативність:** В будь-який момент можна змінювати СУБД без суттєвих змін коду, наприклад SQLite, PostgreSQL, MySQL та багато інших.
- **Вбудована панель адміністратора:** Django автоматично створює панель адміністратора на основі моделей, що спрощує управління даними.

2.5.3 Безпека в Django

Фреймворк надає розробникам вбудовані механізми для захисту від поширених вебвразливостей. Нижче представлено декілька вбудованих механізмів в Django:

- **Захист від XSS (міжсайтовий скриптинг):** Django автоматично екранує дані, що виводяться користувачами, що запобігти виконанню шкідливого JavaScript-коду.
- **Захист від CSRF (підробка міжсайтових запитів):** Django автоматично генерує та перевіряє CSRF-токени для всіх POST-форм, запобігаючи несанкціонованим діям.
- **Захист від SQL-ін'єкцій:** ORM Django автоматично екранує SQL-запити, запобігаючи виконанню шкідливих запитів.
- **Захист паролів:** Django зберігає хеші паролів замість самих паролів, використовуючи надійні алгоритми хешування, такі як PBKDF2 та Argon2.
- **Система автентифікації та авторизації:** Django надає гнучку систему автентифікації та авторизації, яка дозволяє легко налаштовувати доступ до різних частин застосунку.

Враховуючи можливості Django, він гарно підходить для розробки веб-додатка з налаштування та управління мережевих комутаторів. Фреймворк дозволить створити безпечний та масштабований застосунок, який зможе задовольнити потреби мережевих адміністраторів та забезпечити ефективне

управління мережевою інфраструктурою.

2.6 Система управління базами даних SQLite

Для зберігання даних у веб-додатку для зберігання інформації про комутатори, сценарії конфігурацій та логування подій, буде застосовуватись вбудована в Django система управління базами даних SQLite та для взаємодії з нею буде застосовано ORM Django.

SQLite - це легка та швидка база даних, яка не потребує окремого сервера для роботи та широко використовується в різноманітних додатках. Її основні особливості:

- **Легкість:** Вона не потребує окремого сервера, внаслідок чого може швидко та ефективно працювати навіть на системах зі слабкими характеристиками.
- **Простота використання:** Вона не потребує складних налаштувань, та надає можливість взаємодіяти з базою за допомогою мов програмування та мови SQL.
- **Надійність:** вона підтримує транзакції, що забезпечує безпеку та цілісність даних.

2.7 Засоби для розробки користувацького інтерфейсу

Для побудови простого, зручного та функціонального користувацького інтерфейсу веб-додатку буде застосовуватись HTML, CSS та JavaScrypt [15].

2.7.1 HTML (Hypertext Markup Language)

Це основна мова розмітки веб-сторінок на сьогодення, вона описує структуру сторінок та повідомляє браузеру як саме її слід відображати. Він має багато тегів, таки як заголовки, параграфи, посилання, зображення, та багато інших. З їх допомогою можна будувати сторінки будь-якої складності.

2.7.2 CSS (каскадні таблиці стилів)

Це мова яка відіграє важливу роль у веб-дизайні, вона використовується для стилізації HTML сторінки та опису її зовнішнього вигляду. Ця технологія надає розробнику можливість контролювати розташування елементів на сторінці, їх розмір, кольори, стиль та має ще багато інших властивостей. Ще вона дозволяє створювати адаптивний дизайн, що додаток коректно відображався на різних екранах та пристроях, включаючи телефони, планшети та персональні комп'ютери.

2.7.3 JavaScript

JavaScript - це мова програмування, яка початково була створена, щоб зробити веб-сторінки живими та динамічними, та згодом ще з'явилась можливість використовувати JavaScript на боці серверу за допомогою платформи NodeJS. Застосунки цією мовою називаються скриптами та можуть додаватись в HTML-сторінок, що забезпечує їх автоматичне виконання при завантаженні сторінки. Він дозволяє створювати різноманітні функції такі як асинхронні запити на сервер, валідація даних введених користувачем перед відправленням на сервер, також дозволяє маніпулювати даними та елементами сторінки без необхідності її перезавантажувати.

2.7.4 Bootstrap

Це один з найпопулярніших фронтенд-фреймворків з відкритим кодом, він надає набір готових інструментів для створення адаптивних веб-сайтів та

веб-додатків. Містити в собі остовні технології для веб-розробки які були розглянуті раніше, HTML, CSS та JavaScript компоненти, які спрощують та прискорюють розробку інтерфейсу користувача.

Основні переваги Bootstrap:

- Можливість створювати адаптивний дизайн.
- Багато готових компонентів, таких як кнопки, форми, меню, таблиці, модульні вікна та інші компоненти під будь-які потреби.
- Використовує grid систему з 12 колонок, яка надає можливість легко організовувати елементи на сторінках та створювати складні структури.

Поєднання HTML, CSS та JavaScript з фреймворком Bootstrap надасть змогу створити сучасний, зручний та адаптивний користувацький інтерфейс для розробляємого веб-додатку.

2.8 Visual studio code

Visual studio code (VS Code) - безкоштовний, а головне потужний та простий редактор коду, створений компанією Microsoft. Має велику екосистему розширень для підтримки різних мов програмування, фреймворків та рішень для поліпшення розробки. Володіє такими функціями як доповнення коду, підтримку Git, можливість відлагодження коду та багато інших корисних функцій для розробки.

VC Code обрано як основне середовище для розробки веб-додатку за темою дипломної роботи, завдяки розширенням ми налаштуємо його для роботи з Python, Django, HTML, CSS та JavaScript, що робить його ідеальним інструментом для розробки веб-додатку.

2.9 Клієнт-серверна архітектура

Веб-додаток для налаштування та управління мережевими комутаторами буде побудована за класичною клієнт-серверною архітектурою, з використанням моделі MVT (Model-View-Template) фреймворку Django. Ця архітектура передбачає розподіл функцій між двома основними компонентами:

- Клієнт (веб-браузер) яки відповідає за відображення користувацького інтерфейсу, взаємодії з користувачем та надсилання HTTP-запитів на сервер. В цій роботі фронтенд буде реалізовуватись з використанням HTML, CSS, JavaScript та Bootstrap.
- Сервер (Django) який обробляє HTTP-запити від клієнта та виконує необхідні функції, наприклад отримання інформації з бази даних або взаємодія з комутаторами за допомогою Telnet, та в результаті повертає відповідь клієнту. Бекенд частина веб-додатка буде реалізована мовою Python з використанням фреймворку Django.

Використання клієнт-серверної архітектури дозволить розділити клієнтську та серверну частину додатка, що спрощує розробку, тестування та подальшу підтримку. В поєднанні з можливостями Django дозволить створити гнучкий, масштабований та зручний веб-додаток для управління мережевими комутаторами.

2.10 Взаємодія сервера з комутаторами

Для взаємодії з мережевими комутаторами сервер веб-додатка буде використовувати бібліотеку Telnetlib. Процес взаємодії буде включати наступні кроки:

- Сервер встановлює Telnet з’єднання з комутатором використовуючи його IP-адресу.
- Виконує аутентифікацію на комутаторі за дорогою логіна та пароля

отриманого від користувача.

- Відправляє команди на комутатор через Telnet з'єднання.
- Оброблює отриману відповідь від комутатора, яка може містити інформацію про стан комутатора або результати виконаних команд.
- Оброблює отримані відповіді та формує відповідь для користувача в JSON форматі яку далі обробить фронтенд.

Використання протоколу Telnet та бібліотеки Telnetlib забезпечує пряму та ефективну взаємодію сервера з мережевими комутаторами. Цей підхід надає змогу реалізувати широкий спектр функцій для налаштування та змін конфігурацій комутаторів через інтерфейс веб-додатка.

3 ОПИС РОЗРОБКИ ВЕБ-ДОДАТКА ДЛЯ НАЛАШТУВАННЯ ТА УПРАВЛІННЯ МЕРЕЖЕВИМИ КОМУТАТОРАМИ

3.1 Створення та налаштування проєкту в Django

Веб додаток для налаштування та управління комутаторами буде розроблятись за допомогою фреймворку Django та для початку нам треба створити проєкт в IDE (VS Code). Створення проєкта та початкової структури відбувається за допомогою команди в терміналі “django-admin startproject switch_management_tool”, в цій команді “switch_management_tool” назва проєкту та він виступає каркасом для майбутніх застосунків.

Наступним кроком нам необхідно створити застосунок, використаємо абревіатуру назви проєкту та в Django це робиться за допомогою команди “python manage.py startapp smt”, відповідно “smt” це назва додатку.

В результаті ми маємо базову структуру проєкту, але нам знадобиться ще вручну додати декілька директорій через те, що вони не створюються VS Code автоматично, перша “Static” для зберігання статичних файлів де в нас будуть зберігатись стилі CSS та Javasript файли, другою директорією буде “Script” де ми надалі будемо зберігати файл зі сценаріями конфігурацій. В результаті маємо базову структуру проєкту (Рис. 3.1).

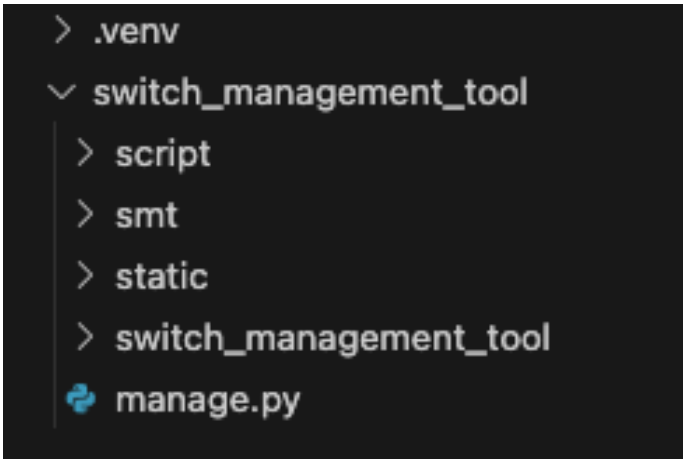


Рис. 3.1 базова структура проєкту

Для візуального представлення функціональних вимог та ілюстрації того, як користувач буде взаємодіяти з додатком, було розроблено діаграму прецедентів (Рис. 3.2). Ця діаграма відображає основні сценарії використання, акторів та взаємодія між ними.

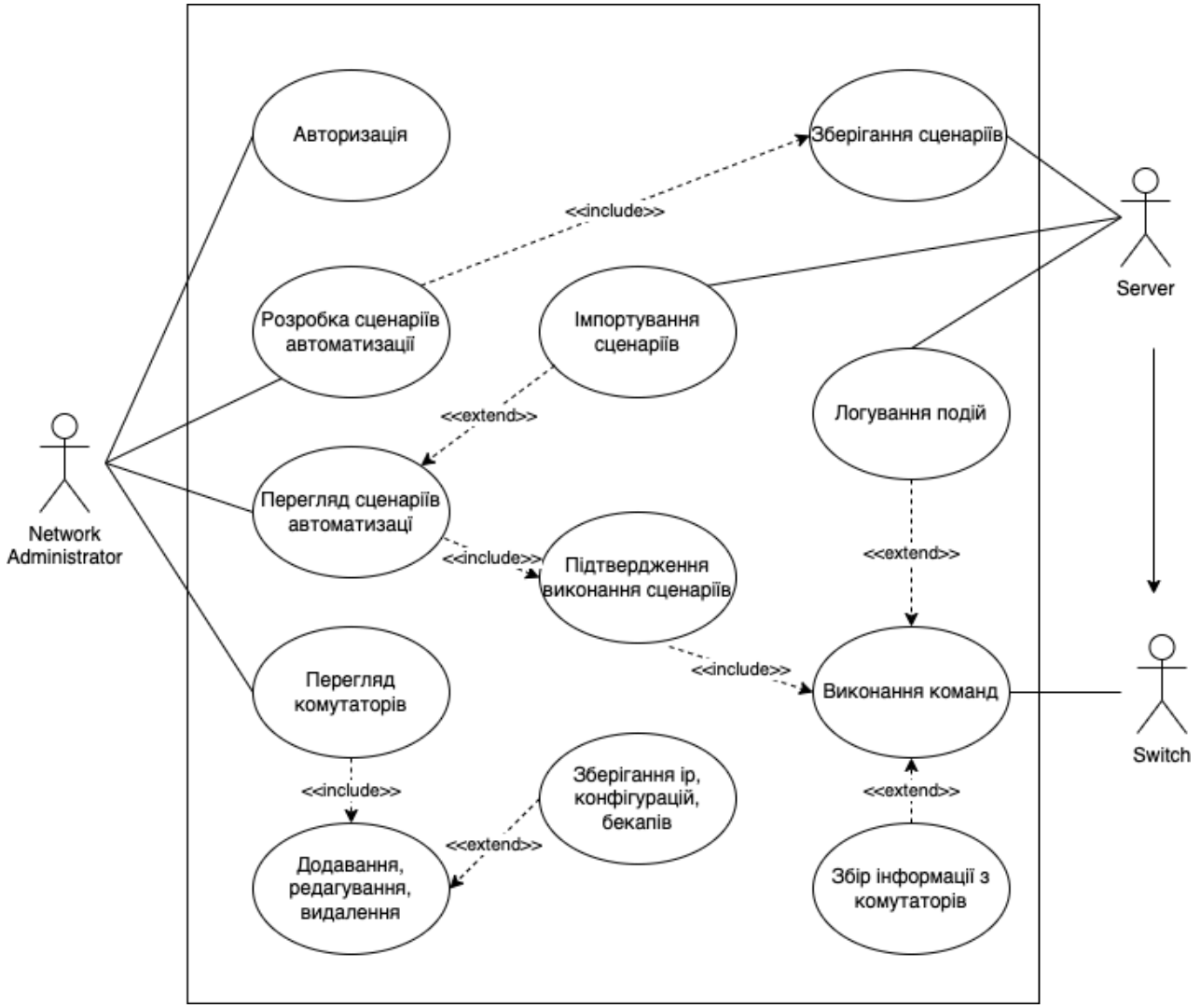


Рис. 3.2 діаграма прецедентів

3.2 Створення моделей даних (Models)

Моделі даних в Django є основою для зберігання інформації у веб-додатку. За допомогою Django ORM ми зможемо взаємодіяти з базою даних за допомогою класів Python, без необхідності заглиблюватись в специфіку

конкретної СУБД та написання прямих SQL-запитів, ORM автоматично перетворює код на Python в SQL-запити. Для потреб веб-додатку розроблено архітектуру бази даних (Рис. 3.3), яка містить в собі з чотири таблиці.

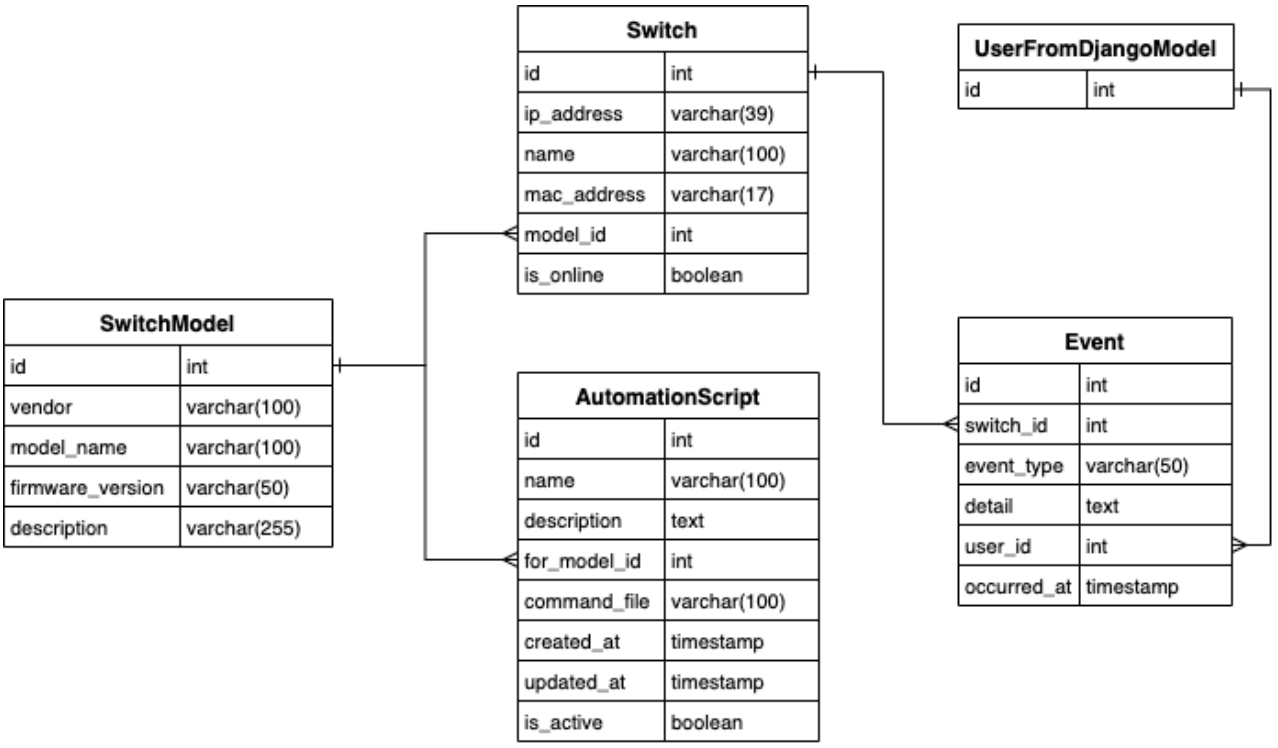


Рис. 3.3 Архітектура бази даних

Детальніше розглянемо таблиці які містяться в базі даних:

SwitchModel (моделі комутаторів рисунок 3.4):

- vendor - виробник
- model_name - модель комутатора
- frimware_version - версія програмного забезпечення
- description - додаткова інформація

```python
class SwitchModel(models.Model):
    vendor = models.CharField(max_length=100)
    model_name = models.CharField(max_length=100)
    firmware_version = models.CharField(max_length=50, null=True, blank=True)
    description = models.CharField(max_length=255, blank=True)

    def __str__(self):
        return f"{self.vendor} {self.model_name}"
```

Рис. 3.4 Модель “SwitchModel”

Switch (комутатори Рис. 3.5):

- ip_address - ip- адреса вузла
- name - ім’я
- mac_adress - mac- адреса вузла
- model_id - посилання на модель комутатора
- is_online - статус доступності комутатора

```python
class Switch(models.Model):
    ip_address = models.GenericIPAddressField(unique=True)
    name = models.CharField(max_length=100, unique=True)
    mac_address = models.CharField(max_length=17, null=True, blank=True, unique=True)
    model = models.ForeignKey(SwitchModel, on_delete=models.CASCADE, null=True, blank=True)
    is_online = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.name} {self.ip_address} {self.model.vendor} {self.model.model_name}"
```

Рис. 3.5 Модель “Switch”

AutomationScript (скрипти автоматизації Рис. 3.6):

- name - назва
- description - опис
- for_model_id - посилання на модель комутатора
- command_file - посилання на файл зі скриптом
- created_at - дата створення
- updated_at - дата оновлення
- is_active - статус

```python
class AutomationScript(models.Model):
    name = models.CharField(max_length=100, unique=True)
    description = models.TextField(blank=True)
    for_model = models.ForeignKey(SwitchModel, on_delete=models.CASCADE)
    command_file = models.FileField(upload_to='scripts/', null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    is_active = models.BooleanField(default=True)

    def __str__(self):
        return self.name
```

Рис. 3.6 Модель “AutomationScript”

Event (події Рис. 3.7):

- switch_id - посилання на комутатор
- event_type - тип події
- detail - журнал події
- user_id - автор події
- occurred_at - час коли відбулась подія

```python
class Event(models.Model):
    switch = models.ForeignKey(Switch, on_delete=models.CASCADE)
    event_type = models.CharField(max_length=50)
    detail = models.TextField()
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True)
    occurred_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.event_type} on {self.switch}"
```

Рис. 3.7 Модель “Event”

Розроблена архітектура забезпечує основу для роботи веб-додатку, що дозволяє зберігати, оновлювати та обробляти дані про комутатори, сценарії автоматизації та події. Також вони враховують функціональні вимоги застосунку та сприятимуть ефективній роботі.

3.3 Створення та обробка форм

Форми в Django є невід’ємною частиною веб-додатку, забезпечують ефективну взаємодію користувача з системою. Вони працюють в ролі інтерфейсу для введення, редагування та видалення даних про комутатори, та сценарії автоматизації.

Фреймворк Django має потужний механізм роботи з формами, що дозволяє легко створювати форми, пов’язувати їх з моделями, валідувати дані, обробляти та відображати їх в шаблонах. У веб-додатку SMT використовуються наступні

форми:

1. “SwitchForm” це ключовий елемент інтерфейсу користувача для взаємодії з даними про комутатори (Рис. 3.8). Форма забезпечує можливість збору та валідації даних, які потім будуть збережені у базі даних. Включає поля для введення IP-адреси комутаторів, ім’я, MAC-адресу та вибір моделі зі списку доступних моделей.

```
class SwitchForm(forms.ModelForm):
    class Meta:
        model = Switch
        fields = ['ip_address', 'name', 'mac_address', 'model']
        widgets = {
            'ip_address': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'IP Address'}),
            'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Name'}),
            'mac_address': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'MAC Address'}),
            'model': forms.Select(attrs={'class': 'form-select'}),
        }
```

Рис. 3.8 Форма “SwitchForm”

2. “AutomationScriptForm” ця форма дозволяє користувачам взаємодіяти та завантажувати сценарії автоматизації (Рис. 3.9). Включає поля для назви, опису, вибору моделі та завантаження файлу з конфігураціями.

```
class AutomationScriptForm(forms.ModelForm):
    class Meta:
        model = AutomationScript
        fields = ['name', 'description', 'for_model', 'command_file']
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Script name'},),
            'description': forms.Textarea(attrs={'class': 'form-control', 'placeholder': 'Script description'}),
            'for_model': forms.Select(attrs={'class': 'form-select'}),
            'command_file': forms.ClearableFileInput(attrs={'class': 'form-control'}),
        }
```

Рис. 3.9 Форма “AutomationScriptForm”

3. “CustomLoginForm” додаток не має окремої моделі для користувачів тому ця форма успадковується від вбудованої в Django “AuthenticationForm” та використовується для автентифікації користувачів (Рис. 3.10). Має поля для вводу імені користувача та паролю.

```python
class CustomLoginForm(AuthenticationForm):
    username = forms.CharField(
        widget=forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Email address', 'id': 'floatingInput'})
    )
    password = forms.CharField(
        widget=forms.PasswordInput(attrs={'class': 'form-control', 'placeholder': 'Password', 'id': 'floatingPassword'})
    )
```

Рис. 3.10 Форма “AuthenticationForm”

3.4 Розробка представлень (Views) та логіки роботи

Представлення в Django відповідають за обробку HTTP-запитів користувача, отримання даних з моделей, виконання необхідних функцій та в кінцевому результаті формування HTTP-відповіді користувачу. Вони реалізовуються в стандартному файлі “views.py” у вигляді функцій або класів, які можуть успадковуватись від базових класів Django.

У Django використовується система маршрутизації, яка дозволяє зв’язати URL-шаблони з відповідними представленнями, що дозволяє користувачу отримати доступ до різних функцій веб-додатку за допомогою унікальних HTTP-запитів.

Приклад реалізації представлень:

- Представлення “execute_script” обробляє POST-запит на виконання сценарію автоматизації на вибраному комутаторі. Воно отримує ідентифікатори сценарію (script_id) та комутатора (switch_id), а також облікові дані (username, password) з POST-запиту. Потім перевіряє наявність файлу сценарію, ініціалізує відповідний об'єкт класу Telnet в залежності від вендора комутатора, виконує команди сценарію та повертає результат виконання у вигляді JSON-об'єкта. Якщо виконання сценарію було успішним, у журналі подій створюється відповідний запис (Рис. 3.11).
- Представлення “switches” обробляє GET-запит на відображення списку комутаторів. Воно отримує всі об'єкти Switch з бази даних, використовуючи prefetch_related для оптимізації запитів до пов'язаної

моделі SwitchModel. Потім воно передає отримані дані та список всіх моделей комутаторів у шаблон smt/switches.html для рендерингу (Рис. 3.12).

- Представлення “switch_add” обробляє запити на додавання нового комутатора до системи. У разі GET-запиту, воно відображає форму SwitchForm для введення даних про комутатор (IP-адреса, ім'я, MAC-адреса, модель). При отриманні POST-запиту, представлення валідує дані з форми та, у разі успішної валідації, зберігає новий об'єкт Switch у базі даних. Після цього користувач перенаправляється на сторінку зі списком комутаторів (/switches/) (Рис. 3.13).

```python
@login_required
def execute_script(request, script_id):
    if request.method == 'POST':
        switch_id = request.POST.get('switch_id')
        username = request.POST.get('username')
        password = request.POST.get('password')
        script = get_object_or_404(AutomationScript, pk=script_id)
        switch = get_object_or_404(Switch, pk=switch_id)
        try:
            if not script.command_file:
                return JsonResponse({"success": False, "message": "No command file associated with the script."})
            # Отримання повного шляху до файлу за допомогою default_storage
            file_path = default_storage.path(script.command_file.name)
            if switch.model.vendor == 'Raisecom':
                telnet = RaisecomTelnet(switch.ip_address, username, password)
            elif switch.model.vendor == 'D-link':
                telnet = DlinkTelnet(switch.ip_address, username, password)
            else:
                return JsonResponse({"success": False, "message": "Unsupported switch vendor"})

            # Виконання скрипту
            telnet.execute_script(file_path)

            # Створення запису в моделі Event
            Event.objects.create(
                switch=switch,
                event_type="Execute Script",
                detail="\n".join([f"{'_'*20}\n Command: {cmd},\n{'*'*20}\n Response:\n {resp}\n" for cmd, resp in telnet.command_log]),
                user=request.user
            )
            return JsonResponse({"success": True, "message": "Script executed successfully."})
        except Exception as e:
            return JsonResponse({"success": False, "message": f"Error executing script: {e}"})
    return JsonResponse({"success": False, "message": "Invalid request method."})
```

Рис. 3.11 Представлення “execute_script”

```python
@login_required
def switches(request):
    switch_list = Switch.objects.all().prefetch_related('model').order_by('-is_online')
    models = SwitchModel.objects.all()
    data = {
        'title': 'Switches',
        'switches': switch_list,
        'models': models
    }
    return render(request, 'smt/switches.html', context=data)
```

Рис. 3.12 Представлення “switches”

```python
@login_required
def switch_add(request):
    if request.method == 'POST':
        form = SwitchForm(request.POST)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect(reverse('switch_list'))
    return HttpResponseRedirect(reverse('switch_list'))
```

Рис. 3.13 Представлення “switch_add”

3.5 Розробка функціонала для взаємодії з комутаторами

Для забезпечення функціональності веб-додатку з налаштування та управління комутаторами, сервер додатку використовує протокол Telnet, який дозволяє встановлювати з'єднання з віддаленими пристроями та взаємодіяти з ними через інтерфейс командного рядка. Для роботи з Telnet використовується вбудована в Python бібліотека Telnetlib, вона надасть додатку можливість встановлювати та закривати Telnet з’єднання, відправляти команди на комутатори та отримувати та обробляти відповідь, що допоможе нам автоматизовувати взаємодію з комутаторами.

Для реалізації функціоналу керування комутаторами в додатку було створено спеціальні класи, ключовим з них є клас “BaseTelnet” який реалізовує основні функції, такі як з’єднання з комутаторами, відправка команд, закриття сесії, перетворення тексту в байти тому як саме з ними працює бібліотека Telnetlib, ці функції будуть подібними для різних вендорів, тому й винесені в окремий клас (Рис. 3.14).

```python
class BaseTelnet:
    def __init__(self, host, username, password, enable_password=None):
        self.host = host
        self.username = username
        self.password = password
        self.enable_password = enable_password
        self.telnet = telnetlib.Telnet(host)

    def to_bytes(self, text):
        return text.encode('ascii') + b'\n'

    def get_mac(self, string):
        return re.search(r"([0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4})", str(string))

    def login(self, login_prompt=b'Login:', password_prompt=b'Password:'):
        self.telnet.read_until(login_prompt, timeout=5)
        self.telnet.write(self.to_bytes(self.username))
        self.telnet.read_until(password_prompt, timeout=5)
        self.telnet.write(self.to_bytes(self.password))

        if self.enable_password:
            self.telnet.write(b"enable\n")
            self.telnet.read_until(b"Password: ", timeout=5)
            self.telnet.write(self.to_bytes("\n"))
            self.telnet.write(self.to_bytes("\n"))

    def send_command(self, command, delay=1):
        self.telnet.write(self.to_bytes(command))
        time.sleep(delay)
        return self.telnet.read_very_eager().decode('utf-8')

    def close(self):
        self.telnet.close()
```

Рис. 3.14 Клас “BaseTelnet”

Під кожного вендора якого буде підтримувати застосунок створено окремі класи “RaisecomTelent” та “DlinkTelent”, вони успадковуються від основного класу “BaseTelnet” та розширюються його.

“RaisecomTelent” має адаптований функціонал для роботи з комутаторами Raisecom, він додає наступні методи:

- configure_switch - налаштувати інший комутатор віддалено (Рис. 3.15).

```
def configure_switch(self, new_sw, commands_file, vlan, gateway):
    self.login(login_prompt=b'Login:', password_prompt=b'Password:')
    sw1_ver = self.get_version()

    self.send_command("telnet 192.168.0.1")
    self.login(login_prompt=b'Login:', password_prompt=b'Password:')
    sw2_ver = self.get_version()

    if self.get_mac(sw1_ver) != self.get_mac(sw2_ver):
        with open(commands_file, "r") as file:
            commands = file.readlines()

        for command in commands:
            command = command.strip()
            if command.startswith("vlan"):
                command = f"vlan {vlan}"
            elif command.startswith("interface vlan"):
                command = f"interface vlan {vlan}"
            elif command.startswith("ip address"):
                command = f"ip address {new_sw}  255.255.255.0"
            elif command.startswith("ip route"):
                command = f"ip route 0.0.0.0 0.0.0.0 {gateway}"

            if command:
                output = self.send_command(command)
                if "'y' to confirm" in output:
                    self.telnet.write(self.to_bytes("y"))
                elif "'yes' to confirm" in output:
                    self.telnet.write(self.to_bytes("yes"))
                elif "Raisecom" in output:
                    ...
                else:
                    break
                self.telnet.write(b"exit\n")
            self.log_command(command, output)
    self.close()
```

Рис. 3.15 Функція configure_switch

- add_vlan, delete_vlan, show_vlan - методи для роботи з Vlan (Рис. 3.16, 3.17, 3.18).

```python
def add_vlan(self, vlan_id, tagged_port, untagged_port):
    try:
        self.login()
        commands = [
            f"conf",
            f"vlan {vlan_id}",
            f'exit',
            f"interface ten 1/1/{tagged_port}",
            f"switchport mode trunk",
            f"switchport trunk allowed vlan add {vlan_id}",
            f"interface gi 1/1/{untagged_port}",
            f"switchport mode access",
            f"switchport access vlan {vlan_id}",
            f'exit'
        ]
        for command in commands:
            info = self.send_command(command)
            self.log_command(command, info)
        self.close()
        return "VLAN added successfully"
    except Exception as e:
        self.close()
        raise
```

Рис. 3.16 Функція add_vlan

```python
def delete_vlan(self, vlan_id):
    try:
        self.login()
        commands = [
            f"conf",
            f"no vlan {vlan_id}",
            f"exit"
        ]
        for command in commands:
            response = self.send_command(command)
            self.log_command(command, response)
        self.close()
        return f"VLAN deleted successfully."
    except Exception as e:
        self.close()
        raise
```

Рис. 3.17 Функція delete_vlan

```python
def show_vlan(self, vlan_id):
    try:
        self.login()
        command = f"show vlan {vlan_id}"
        response = self.send_command(command)
        self.log_command(command, response)
        self.close()
        return response
    except PermissionError as pe:
        self.close()
        return f"Authentication error: {str(pe)}"
    except Exception as e:
        self.close()
        raise
```

Рис. 3.18 Функція show_vlan

- disable_port, enable_port - методи для керування портами (Рис. 3.18).

```python
def disable_port(self, port):
    try:
        self.login()
        commands = [
            f"conf",
            f"interface gigabitethernet 1/1/{port}",
            f"shutdown",
            f"exit"
        ]
        for command in commands:
            self.send_command(command)
        self.close()
        return "Port administrative disable"
    except Exception as e:
        self.close()
        raise

def enable_port(self, port):
    try:
        self.login()
        commands = [
            f"conf",
            f"interface gigabitethernet 1/1/{port}",
            f"no shutdown",
            f"exit"
        ]
        for command in commands:
            self.send_command(command)
        self.close()
        return "Port administrative enable"
    except Exception as e:
        self.close()
        raise
```

Рис. 3.19 Функції disable_port, enable_port

- show_info - отримує загальну інформацію про комутатор (Рис. 3.20).

```
def show_info(self):
    try:
        self.login()
        command = "show version"
        response = self.send_command("show version")
        self.log_command(command, response)
        self.close()
        return response
    except Exception as e:
        self.close()
        raise
```

Рис. 3.20 Функції show_info

- execute_script - виконання сценаріїв конфігурацій (Рис. 3.21).

```
def execute_script(self, file_path):
    try:
        self.login()
        with open(file_path, 'r') as f:
            commands = f.readlines()
            full_respones = ''
            for command in commands:
                command = command.strip()
                if command:
                    self.telnet.write(self.to_bytes(command))
                    response = self.telnet.read_until(b"#", timeout=3).decode('utf-8')
                    while '--More--' in response:
                        full_respones += response.replace('--More--', '')
                        self.telnet.write(self.to_bytes(' '))
                        response = self.telnet.read_until(b"--More--", timeout=3).decode('utf-8')
                    full_respones += response
                    self.log_command(command, response)
        self.close()
        return full_respones
    except Exception as e:
        self.close()
        raise
```

Рис. 3.21 Функції execute_script

- log_command - записує команди та відповіді в журнал подій (Рис. 3.22).
- clear_response - очищає відповідь від зайвих символів (Рис. 3.22).

```
def log_command(self, command, response):
    cleaned_response = self.clean_response(response)
    self.command_log.append((command, cleaned_response))

def clean_response(self, response):
    lines = response.split('\n')
    cleaned_lines = [line.strip() for line in lines if line.strip()]
    return '\n'.join(cleaned_lines)
```

Рис. 3.2 Функції log_command, clear_response

Для роботи з іншим вендором використовується клас “DlinkTelnet”, він має аналогічні методи з попереднім класом, але синтаксис команд відрізняється тому функції в ньому адаптовані під виробника D-Link.

Використання об'єктноорієнтованого підходу та успадкування надає змогу створювати гнучку та розширювану систему для взаємодії з комутаторами різних вендорів. Основний клас надає загальний функціонал, а класи під конкретного виробника реалізовує специфічні команди. Це спрощує підтримку коду та надає можливість легко додавати підтримку нових вендорів за необхідності.

3.6 Створення шаблонів (Templates)

Шаблони відповідають за візуальне представлення коду, це HTML-файли які містять статичний контент, або в Django для відображення динамічних даних застосовуються спеціальні теги та змінні. Користувацький інтерфейс розробляється з використанням шаблонів Django, Bootstrap, та JavaScrip з застосуванням концепції AJAX для динамічного відображення даних користувачу без перезавантаження сторінки.

HTML-файли зберігаються в директорії додатку SMT у папці “Template”, веб-додаток має 5 сторінок та використовує 6 шаблонів (Рис. 3.23).

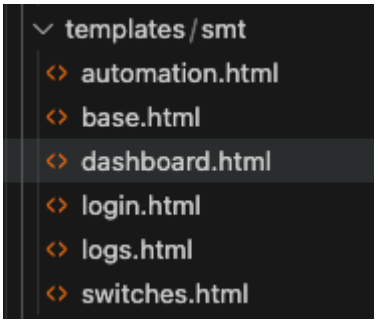


Рис. 3.23 Структура HTML шаблонів

Основний шаблон знаходиться в файлі base.html від нього всі інші сторінки будуть успадковувати базовий каркас, розглянемо детальніше шаблон та його компоненти (Рис. 3.24). В ньому ми одразу можемо під'єднати необхідні CSS та JavaScript файли, а також інші зовнішні бібліотеки, наприклад bootstrap який ми використовуємо. В результаті ці елементи будуть успадковані іншими сторінками веб-додатку.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}SMT{% endblock %}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css">
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
    {% block extra_head %}{% endblock %}
</head>
<body class="bg-dark text-light">
    <nav class="navbar navbar-expand-lg navbar-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Switch managment tools</a>
        </div>
    </nav>
    <div class="container-fluid">
        <div class="row">
            <nav class="col-md-2 sidebar">…
            </nav>
            <main class="col-md-10 content">
                {% block content %}
                {% endblock %}
            </main>
        </div>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
    {% block extra_js %}{% endblock %}
</body>
</html>
```

Рис. 3.24 - Структура базового шаблону base.html

Для зміни наповнення різних сторінок використовуються теги Django, завдяки яким дочірні шаблони зможуть перевизначати окремі блоки контенту. Основні блоки контенту, що використовуються в base.html:

- {% block title %} - визначає заголовок сторінки.
- {% block extra_header %} - дозволяє додавати додаткові елементи в розділ <head> сторінки.
- {% block content %} - визначає основний вміст сторінки.
- {% block extra_js %} - дозволяє додавати додатковий JavaScript код в кінці сторінки.

Шаблон головної сторінки dashboard.html успадковується від основного шаблону та відображає головну панель з основною інформацією про систему та останні події. Для успадкування використовується тег {% extends 'base.html' %} та потім використовуються блоки контенту, наприклад {% block title %}Dashboard{% endblock %} для перевизначення title з базового шаблона. Основний контент міститься в блоці {% block content %}...{% endblock %} (Рис. 3.25). Подібним методом будуються інші сторінки веб-додатку та завдяки блокам та унікальному наповненню для кожної.

```
{% extends 'base.html' %}

{% block title %}Dashboard{% endblock %}

{% block content %}
<h2>Dashboard</h2>

<div class="row justify-content-center mb-4">…
</div>

<h3>System Logs</h3>
<table class="table table-dark table-striped table-hover">…
</table>

<!-- Event Detail Modal -->
<div class="modal fade" id="eventDetailModal" tabindex="-1" aria-labelledby="eventDetailModalLabel" aria-hidden="true">…
</div>
{% endblock %}

{% block extra_js %}
<script>
document.addEventListener('DOMContentLoaded', function () {…
});
</script>
{% endblock %}
```

Рис. 3.25 Шаблон головної сторінки

Для динамічного формування таблиці подій на головній сторінці використовуються теги та змінні Django. Цикл {% for event in events %} перебирає список подій events, які він отримує з бази даних за допомогою раніше створеного представлення, а за допомогою тегів <tr> та <td> створюються елементи таблиці (Рисунок 3.26). Умовний тег {% if %} використовується для відображення імені користувача, якщо воно доступне, або дефісу, якщо ні.

```
    <thead>
        <tr>
            <th>Time</th>
            <th>Host</th>
            <th>Action</th>
            <th>User</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        {% for event in events %}
        <tr>
            <td>{{ event.occurred_at }}</td>
            <td>{{ event.switch.ip_address }}</td>
            <td>{{ event.event_type }}</td>
            <td>{% if event.user %}{{ event.user.username }}{% else %}-{% endif %}</td>
            <td>⋯
            </td>
        </tr>
        {% empty %}
        <tr>
            <td colspan="5">No events available.</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

Рис. 3.26 Формування таблиці з подіями

Використовує JavaScript який відправляє AJAX-запит на сервер для отримання детальної інформації про подію. Отримані дані потім відображаються у модальному вікні, що дозволяє користувачу переглянути деталі події без перезавантаження сторінки (Рис. 3.27).

```
{% block extra_js %}
<script>
document.addEventListener('DOMContentLoaded', function () {
    var eventDetailModal = document.getElementById('eventDetailModal');
    eventDetailModal.addEventListener('show.bs.modal', function (event) {
        var button = event.relatedTarget;
        var eventId = button.getAttribute('data-event-id');

        fetch(`/logs/event_detail/${eventId}/`, {
            method: 'GET',
            headers: {
                'X-Requested-With': 'XMLHttpRequest',
            },
        })
        .then(response => response.json())
        .then(data => {
            var eventDetailContent = document.getElementById('eventDetailContent');
            if (data.success) {
                eventDetailContent.innerText = data.detail;
            } else {
                eventDetailContent.innerText = 'Failed to load event details.';
            }
        })
        .catch(error => {
            var eventDetailContent = document.getElementById('eventDetailContent');
            eventDetailContent.innerText = 'An error occurred. Please try again.';
        });
    });
});
</script>
{% endblock %}
```

Рис. 3.27 AJAX-запит на сервер для отримання інформації

4 ТЕСТУВАННЯ ВЕБ-ДОДАТКА

Тестування програмного забезпечення є критично важливим етапом розробки, оскільки воно дозволяє забезпечити якість, надійність та відповідність функціональним вимогам. Існує багато різних видів тестування, кожен з яких має свою мету та завдання. Основні види тестування включають:

Функціональне тестування (Functional testing) перевіряє роботу застосунку з точки зору користувача, включаючи тестування всіх функцій, описаних у вимогах. Мета функціонального тестування - гарантувати, що застосунок відповідає очікуванням користувачів та готовий до використання.

Інтеграційне тестування (Integration testing): перевіряє взаємодію між різними компонентами системи. Воно допомагає виявити помилки, які виникають при інтеграції окремих компонентів, та забезпечити їхню спільну роботу.

Модульне тестування (Unit testing): фокусується на перевірці окремих компонентів коду, таких як функції, класи та методи, в ізоляції від інших частин системи. Це дозволяє виявити помилки на ранніх стадіях розробки та забезпечити коректність роботи кожного компонента окремо.

4.1 Обґрунтування вибору виду тестування

З огляду на обмежені ресурси та часові рамки дипломного проєкту, було обрано функціональне тестування як основний метод перевірки якості веб-додатку. Це рішення обґрунтоване наступними факторами:

- Основною метою дипломного проекту є розробка функціонального веб-додатку, який відповідає заявленим вимогам. Функціональне тестування дозволяє безпосередньо перевірити, чи всі функції працюють коректно.
- Тестування можна проводити вручну, без необхідності розробки складних

автоматизованих тестів.

4.2 Результати тестування

Для перевірки функціональності веб-додатку було проведено ручне тестування, під час якого виконувалися різні сценарії використання застосунку. Кожен сценарій включав певну послідовність дій користувача та очікуваний результат. Результати тестування, представлені у таблиці 4.1, підтверджують коректну роботу всіх основних функцій

Таблиця 4.1

Результати проведеного тестування

№	Назва	Дія	Очікуваний результат	Статус
1	Авторизація користувача	Введення коректних облікових даних (логін та пароль) та натискання кнопки "Увійти"	Успішний вхід до системи, перенаправлення на головну сторінку (dashboard).	Пройдено
2	Додавання нового комутатора	Заповнення форми додавання комутатора з коректними даними (IP-адреса, назва) та натискання кнопки "Додати"	Комутатор успішно додається до бази даних, користувач отримує повідомлення про успіх та перенаправляється на сторінку зі списком комутаторів.	Пройдено
3	Видалення існуючого комутатора	Вибір комутатора зі списку, натискання кнопки "Видалити" та підтвердження дії	Комутатор видаляється з бази даних, користувач отримує повідомлення про успіх та перенаправляється на сторінку зі списком комутаторів.	Пройдено
4	Редагування існуючого комутатора	Вибір комутатора зі списку, натискання кнопки "Редагувати", зміна даних у формі та натискання кнопки "Зберегти"	Дані про комутатор успішно оновлюються в базі даних, користувач отримує повідомлення про успіх та перенаправляється на сторінку зі списком комутаторів.	Пройдено

Продовження таблиці 4.1

Результати проведеного тестування

№	Назва	Дія	Очікуваний результат	Статус
5	Збір даних з комутатора	Вибір комутатора зі списку, натискання кнопки "Отримати інформацію"	Дані про виробника, модель та версію прошивки комутатора успішно отримані та відображені користувачу.	Пройдено
6	Виконання сценарію конфігурації на комутаторі	Вибір сценарію зі списку, вибір комутатора, натискання кнопки "Виконати"	Сценарій успішно виконаний на вибраному комутаторі, користувач отримує повідомлення про успіх (або помилку), подія записується в лог.	Пройдено
7	Імпорт сценарію	Вибір файлу сценарію, натискання кнопки "Імпортувати"	Сценарій успішно імпортований в систему, користувач отримує повідомлення про успіх.	Пройдено
8	Експорт сценарію	Вибір сценарію зі списку, натискання кнопки "Експортувати"	Файл сценарію успішно завантажується на пристрій користувача.	Пройдено
9	Перегляд логів подій	Перейти на сторінку "Logs"	Відкривається сторінка з логами подій, що відбувалися з комутаторами.	Пройдено
10	Перегляд детальної інформації про подію	Натискання на кнопку "Деталі" біля відповідної події в списку логів	Детальна інформація про подію успішно відображається користувачу.	Пройдено

ВИСНОВКИ

В ході виконання дипломної роботи, успішно розроблено веб-додаток для налаштування та управління мережевими комутаторами. Проведене дослідження літературних джерел дозволило виявити актуальні проблеми та потреби галузі мережевого адміністрування, що стало основою для визначення функціональних та нефункціональних вимог до застосунку.

Аналіз існуючих рішень для автоматизації взаємодії з комутаторами допоміг виявити їх сильні та слабкі сторони, що враховано при проектуванні архітектури та функціоналу розробленого веб-додатку. Завдяки використанню сучасних технологій та підходів, таких як Python, Django, Telnetlib та SQLite, було створено ефективний та гнучкий інструмент для керування мережевими комутаторами.

Детальне проектування структури веб-додатку, включаючи визначення основних компонентів (моделі даних, представлення, форми) та їх взаємодії, забезпечило чіткий план розробки та сприяло створенню якісного та надійного програмного продукту.

Реалізація веб-додатку включала розробку зручного та інтуїтивно зрозумілого інтерфейсу користувача, створення механізмів взаємодії з комутаторами через Telnet, а також впровадження функцій для додавання, редагування та видалення комутаторів, створення та виконання сценаріїв конфігурацій, збору інформації про комутатори та логування подій.

Проведене функціональне тестування підтвердило коректну роботу всіх реалізованих функцій та відповідність застосунку заявленим вимогам. Розроблений веб-додаток успішно пройшов тестування на різних типах комутаторів та у різних сценаріях використання, що демонструє його ефективність та практичну цінність для мережевих адміністраторів.

Таким чином, у результаті виконання дипломної роботи було створено повноцінний веб-додаток для налаштування та управління мережевими

комутаторами, який може бути використаний для підвищення ефективності та автоматизації процесів мережевого адміністрування. Застосунок має потенціал для подальшого розвитку та вдосконалення, наприклад, шляхом додавання підтримки інших протоколів управління (SSH, SNMP), функціонал моніторингу та аналізу мережевого трафіку, створення API інтерфейсу для інтеграції з іншими системами управління мережею.

Робота пройшла апробацію та опубліковані наступні тези:

1. Тищенко Д.С. Негоденко О.В. Розробка веб-додатку для налаштування та управління мережевих комутаторів // Матеріали всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. С. 74-75
2. Тищенко Д.С. Аналіз існуючих рішень для розробки веб–додатку з налаштування та управління мережевими комутаторами // Збірник матеріалів Всеукраїнської конференції молодих учених "Інформаційні технології". 16.05.2024, КУБГ, м. Київ. С. 179-181

ПЕРЕЛІК ПОСИЛАНЬ

1. Fraihat, A. Computer networking layers based on the OSI model. Test Eng. Manag, 83, 2021, ISSN: 0193-4120.
2. Network switch - [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Network_switch
3. Adhikari, N., Logeshwaran, J., & Kiruthiga, T. The Artificially Intelligent Switching Framework for Terminal Access Provides Smart Routing in Modern Computer Networks. BOHR International Journal of Smart Computing and Information Technology, 3(1), 45-50, 2022, ISSN 2583-2026.
4. Jayasekara, G. P. D. C. M. (2022). FranPyCisco 2022: Network Automation & Abstraction Solutions To Simplify Configuration Complexity. Available at SSRN 4176096.
5. Chou, Eric, Michael Kennedy, and Mandy Whaley. Mastering Python Networking: Your one-stop solution to using Python for network automation, programmability, and DevOps. Packt Publishing Ltd, 2020. ISBN 978-1-83921-467-7. 539 с.
6. Network Automation Market Size & Share Analysis - Growth Trends & Forecasts (2024 - 2029) - [Електронний ресурс] – Режим доступу до ресурсу: https://www.mordorintelligence.com/industry-reports/network-automation-market-market
7. Samoylenko, N. "Python for Network Engineers." Самопублікація 2015-2022 - [Електронний ресурс] - Режим доступу - https://pyneng.readthedocs.io/en/
8. Network Automation Manager - [Електронний ресурс] – Режим доступу до ресурсу: https://www.solarwinds.com/network-automation-manager
9. Network Automation Tools for Config, Change Management, and Compliance - [Електронний ресурс] – Режим доступу до ресурсу: https://www.solarwinds.com/network-configuration-manager/use-cases/network-automation

10. Cisco DNA Software - [Електронний ресурс] – Режим доступу до ресурсу: https://www.cisco.com/c/en_uk/products/software/dna-software/index.html#~software
11. ManageEngine Network Configuration Management - [Електронний ресурс] – Режим доступу до ресурсу: https://www.manageengine.com/network-configuration-manager/
12. What is Network Configuration Management (NCM) - [Електронний ресурс] – Режим доступу до ресурсу: https://www.manageengine.com/network-configuration-manager/what-is-network-configuration-management.html
13. Red Hat Ansible Automation Platform - [Електронний ресурс] – Режим доступу до ресурсу: https://www.redhat.com/en/technologies/management/ansible
14. What's new in Ansible Automation Platform 2.3 - [Електронний ресурс] – Режим доступу до ресурсу: https://developers.redhat.com/blog/2022/11/29/whats-new-ansible-automation-platform-23#
15. Документація Django - [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/5.0/
16. Documenting web technologies, including CSS, HTML, and JavaScript, since 2005. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/
17. Kim, H., & Feamster, N. Improving network management with software defined networking. IEEE Communications Magazine, 51(2), 114-119, 2013, DOI: 10.1109/MCOM.2013.6461195.
18. Oswalt, Matt, et al. Network Programmability and Automation: Skills for the Next-generation Network Engineer. O'Reilly Media, Incorporated, 2023, 825, ISBN: 978-1-098-11083-3.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка веб-додатку для налаштування та управління мережевих комутаторів мовою Python

Виконав студент 5 курсу
групи ППЗ-51
Тищенко Дмитро Станіславович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Гребенюк Віктор Вікторович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** спростити процес налаштування та керування мережевими комутаторами шляхом використання веб-додатку.
- **Об’єкт дослідження:** процес налаштування та адміністрування мережевих комутаторів.
- **Предмет дослідження:** програмне забезпечення та засоби для керування та налаштування комутаторів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз літературних джерел, для визначення вимог, потреб та проблем галузі мережевого адміністрування та мережевої автоматизації.
2. Провести аналіз наявних засобів для автоматизації взаємодії з комутаторами та програмних засобів для керування мережевими пристроями.
3. Спроектувати структуру проєкту.
4. Розробити та протестувати веб-додаток для автоматизації керування та налаштування комутаторів.

3

АНАЛІЗ АНАЛОГІВ

Застосунок	SolarWinds Network Automation Manager	Cisco DNA	ManageEngine Network Configuration Management	Red Hat Ansible Automation Platform	Switch Management Tool
Автоматизація конфігурації	+	+	+	+	+
Легке впровадження в інфраструктуру	-	-	-	-	+
Мультивендорність	+	-	+	+	+
Резервне копіювання	+	-	+	-	+
Імпорт та управління сценаріями	–	-	-	+	+

4

ВИМОГИ ДО ВЕБ-ДОДАТКА

Функціональні вимоги:

1. Можливість авторизації користувача.
2. Можливість додавати, видаляти та редагувати в системі дані про комутатори, їх ip-адресу та назву.
3. Можливість збору даних з комутаторів, виробник, модель, версія прошивки.
4. Підтримка сценаріїв конфігурацій.
5. Можливість імпортувати та експортувати сценарії.
6. Можливість виконувати заготовлені сценарії конфігурацій на комутаторах.
7. Логування подій, що відбуваються з комутаторами.

Нефункціональні вимоги:

1. Забезпечити захист від несанкціонованого доступу до веб-додатку.
2. Можливість працювати з різних операційних систем та браузерів.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

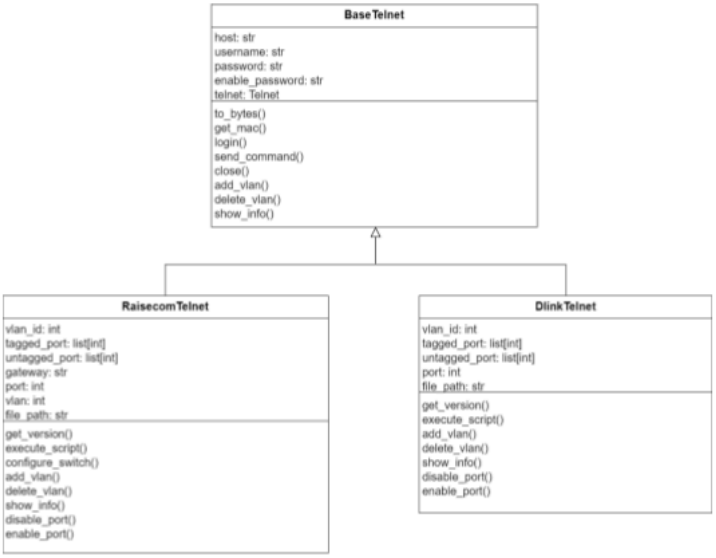
Telnetlib

6

Діаграма варіантів використання

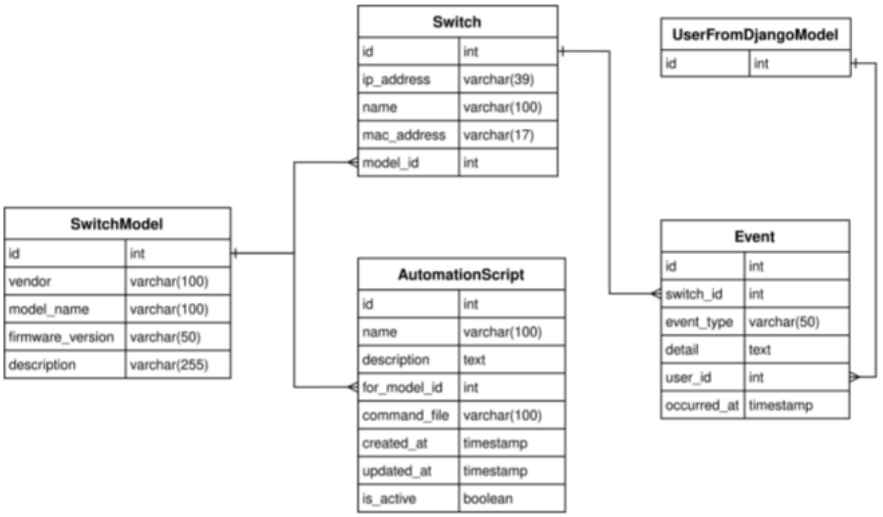
7

Діаграма класів



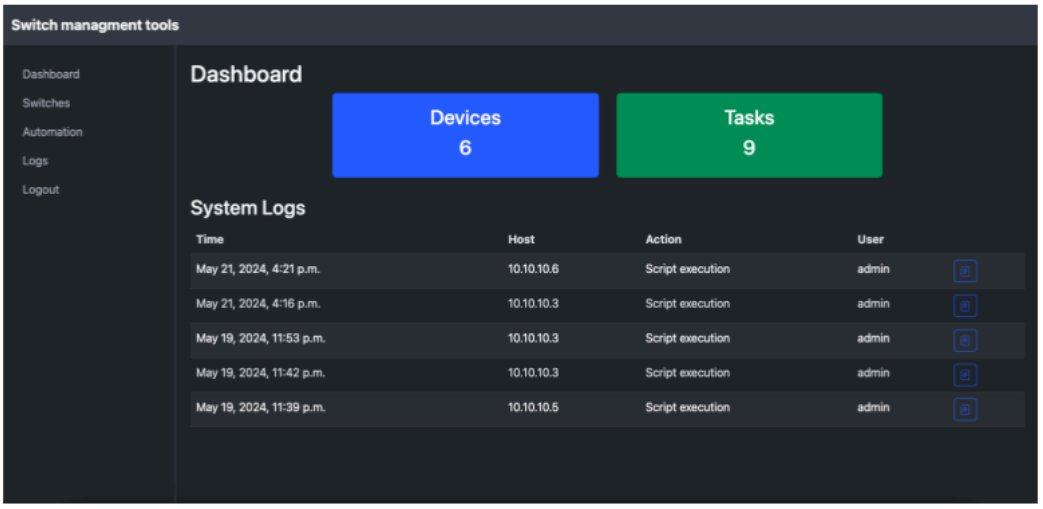
8

Схема бази даних



9

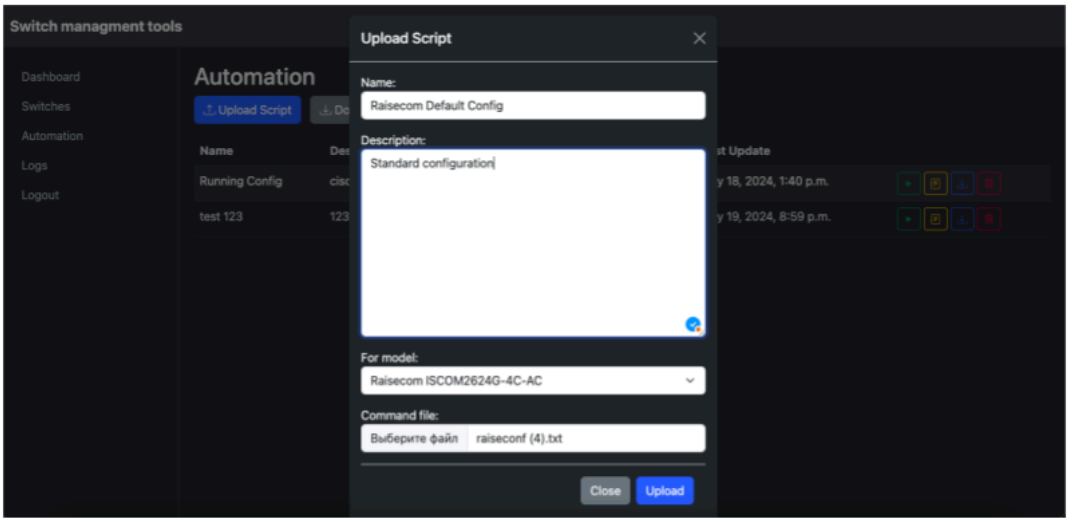
ЕКРАННІ ФОРМИ



Головна сторінка додатку

10

ЕКРАННІ ФОРМИ



Додавання файлу з командами

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Тищенко Д.С. Негоденко О.В. Розробка веб-додатку для налаштування та управління мережевих комутаторів // Матеріали всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. С. 74-75

2. Тищенко Д.С. Аналіз існуючих рішень для розробки веб–додатку з налаштування та управління мережевими комутаторами // Збірник матеріалів Всеукраїнської конференції молодих учених "Інформаційні технології". 16.05.2024, КУБГ, м. Київ. С. 179-181

ВИСНОВКИ

1. Проведено дослідження літературних джерел, що дозволяє виявити актуальні проблеми та потреби галузі мережевого адміністрування, та допомагає визначити основні вимоги для майбутнього додатку.
2. Проведено аналіз існуючих рішень для автоматизації взаємодії з комутаторами та виявлено їх сильні та слабкі сторони.
3. Спроєктовано детальну структуру веб-додатку, визначено основні компоненти та їх взаємодію.
4. Розроблений веб-додаток успішно пройшов тестування, підтвердивши свою відповідність до вимог та ефективність в автоматизації керування та налаштування мережевих комутаторів.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

models.py :
from django.db import models from django.contrib.auth.models import User # Create your models here. class SwitchModel(models.Model): vendor = models.CharField(max_length=100) model_name = models.CharField(max_length=100) firmware_version = models.CharField(max_length=50, null=True, blank=True) description = models.CharField(max_length=255, blank=True) def __str__(self): return f"{self.vendor} {self.model_name}" class Switch(models.Model): ip_address = models.GenericIPAddressField(unique=True) name = models.CharField(max_length=100, unique=True) mac_address = models.CharField(max_length=17, null=True, blank=True, unique=True) model = models.ForeignKey(SwitchModel, on_delete=models.CASCADE, null=True, blank=True) is_online = models.BooleanField(default=False) def __str__(self): return f"{self.name} {self.ip_address} {self.model.vendor} {self.model.model_name}" class AutomationScript(models.Model): name = models.CharField(max_length=100, unique=True) description = models.TextField(blank=True) for_model = models.ForeignKey(SwitchModel, on_delete=models.CASCADE) command_file = models.FileField(upload_to='scripts/', null=True, blank=True) # Поле для завантаження файлів created_at = models.DateTimeField(auto_now_add=True) updated_at = models.DateTimeField(auto_now=True) is_active = models.BooleanField(default=True) def __str__(self): return self.name class Event(models.Model): switch = models.ForeignKey(Switch, on_delete=models.CASCADE) event_type = models.CharField(max_length=50) detail = models.TextField() user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True) occurred_at = models.DateTimeField(auto_now_add=True) def __str__(self): return f"{self.event_type} on {self.switch}"
form.py:
class SwitchForm(forms.ModelForm): class Meta: model = Switch fields = ['ip_address', 'name', 'mac_address', 'model'] class AutomationScriptForm(forms.ModelForm): class Meta: model = AutomationScript fields = ['name', 'description', 'for_model', 'command_file'] widgets = { 'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Script name'},), 'description': forms.Textarea(attrs={'class': 'form-control', 'placeholder': 'Script description'}), 'for_model': forms.Select(attrs={'class': 'form-select'}), 'command_file': forms.ClearableFileInput(attrs={'class': 'form-control'}), } class CustomLoginForm(AuthenticationForm): username = forms.CharField(widget=forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Email address', 'id': 'floatingInput'})) password = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-control', 'placeholder': 'Password', 'id': 'floatingPassword'}))
views.py:
class CustomLoginView(LoginView): template_name = 'login.html' authentication_form = CustomLoginForm @login_required def switches(request): switch_list = Switch.objects.all().prefetch_related('model').order_by('-is_online') models = SwitchModel.objects.all() data = { 'title': 'Switches', 'switches': switch_list, 'models': models } return render(request, 'smt/switches.html', context=data) @login_required def switch_add(request): if request.method == 'POST': form = SwitchForm(request.POST) if form.is_valid(): form.save() return HttpResponseRedirect(reverse('switch_list')) return HttpResponseRedirect(reverse('switch_list')) @login_required def switch_update(request, pk): switch = get_object_or_404(Switch, pk=pk) if request.method == 'POST': form = SwitchForm(request.POST, instance=switch) if form.is_valid(): form.save() return redirect('switch_list') return redirect('switch_list') @login_required def switch_delete(request, pk): switch = get_object_or_404(Switch, pk=pk) if request.method == 'POST': switch.delete() return redirect('switch_list') return redirect('switch_list') @login_required def automation(request): scripts = AutomationScript.objects.all() form = AutomationScriptForm() switches = Switch.objects.all() data = { 'title': 'Automation', 'scripts': scripts, 'form': form, 'switches': switches, } return render(request, 'smt/automation.html', context=data) @login_required def upload_script(request): if request.method == 'POST': form = AutomationScriptForm(request.POST, request.FILES) if form.is_valid(): form.save() return HttpResponseRedirect(reverse('automation')) return HttpResponseRedirect(reverse('automation')) @login_required def download_script(request, pk): script = get_object_or_404(AutomationScript, pk=pk) file_path = script.command_file.path if os.path.exists(file_path): response = FileResponse(open(file_path, 'rb'), as_attachment=True) return response return HttpResponse("File not found.") @login_required def delete_script(request, pk): script = get_object_or_404(AutomationScript, pk=pk) if request.method == 'POST': script.delete() return HttpResponseRedirect(reverse('automation')) return

```
HttpResponseRedirect(reverse('automation')) @login_required def logs(request): events = Event.objects.all().order_by('-occurred_at') data = { 'title': 'Logs', 'events': events } return render(request, 'smt/logs.html', context=data) @login_required def dashboard(request): switch_count = Switch.objects.count() event_count = Event.objects.count() events = Event.objects.all().order_by('-occurred_at')[:5] # Отримати останні 5 подій data = { 'title': 'Dashboard', 'switch_count': switch_count, 'event_count': event_count, 'events': events } return render(request, 'smt/dashboard.html', context=data) @login_required def execute_script(request, script_id): if request.method == 'POST': switch_id = request.POST.get('switch_id') username = request.POST.get('username') password = request.POST.get('password') script = get_object_or_404(AutomationScript, pk=script_id) switch = get_object_or_404(Switch, pk=switch_id) try: if not script.command_file: return JsonResponse({"success": False, "message": "No command file associated with the script."}) # Отримання повного шляху до файлу за допомогою default_storage file_path = default_storage.path(script.command_file.name) if switch.model.vendor == 'Raisecom': telnet = RaisecomTelnet(switch.ip_address, username, password) elif switch.model.vendor == 'D-link': telnet = DlinkTelnet(switch.ip_address, username, password) else: return JsonResponse({"success": False, "message": "Unsupported switch vendor"}) # Виконання скрипту telnet.execute_script(file_path) # Створення запису в моделі Event Event.objects.create( switch=switch, event_type="Execute Script", detail="\n".join([f"{'_'*20}\n Command: {cmd},\n{'*'*20}\n Response:\n {resp}\n" for cmd, resp in telnet.command_log]), user=request.user ) return JsonResponse({"success": True, "message": "Script executed successfully."}) except Exception as e: return JsonResponse({"success": False, "message": f"Error executing script: {e}"}) return JsonResponse({"success": False, "message": "Invalid request method."}) @login_required def add_vlan(request): if request.method == 'POST': switch_id = request.POST.get('switch_id') vlan_id = request.POST.get('vlan_id') tagged_port = request.POST.get('tagged_port') untagged_port = request.POST.get('untagged_port') username = request.POST.get('username') password = request.POST.get('password') switch = get_object_or_404(Switch, pk=switch_id) try: if switch.model.vendor == 'Raisecom': telnet = RaisecomTelnet(switch.ip_address, username, password) elif switch.model.vendor == 'D-link': telnet = DlinkTelnet(switch.ip_address, username, password, untagged_port, tagged_port) else: return JsonResponse({"success": False, "message": "Unsupported switch vendor"}) # Виконання скрипту telnet.add_vlan(vlan_id, tagged_port, untagged_port) # Збереження логу в подію Event.objects.create( switch=switch, event_type="Add vlan execution", detail="\n".join([f"{'_'*20}\n Command: {cmd},\n{'*'*20}\n Response:\n {resp}\n" for cmd, resp in telnet.command_log]), user=request.user ) return JsonResponse({"success": True, "message": "Vlan add successfully."}) except Exception as e: return JsonResponse({"success": False, "message": f"Error executing script: {e}"}) return JsonResponse({"success": False, "message": "Invalid request method."}) @login_required def delete_vlan(request): if request.method == 'POST': switch_id = request.POST.get('switch_id') vlan_id = request.POST.get('vlan_id') username = request.POST.get('username') password = request.POST.get('password') switch = get_object_or_404(Switch, pk=switch_id) if switch.model.vendor == 'Raisecom': try: telnet = RaisecomTelnet(switch.ip_address, username, password) message = telnet.delete_vlan(vlan_id) # Створення запису в моделі Event Event.objects.create( switch=switch, event_type="Delete VLAN", detail="\n".join([f"{'_'*20}\n Command: {cmd} \n{'*'*20}\n Response:\n {resp}\n" for cmd, resp in telnet.command_log]), user=request.user ) return JsonResponse({"success": True, "message": f"Delete vlan: {vlan_id} successfully"}) except Exception as e: return JsonResponse({"success": False, "message": str(e)}) else: return JsonResponse({"success": False, "message": "Unsupported switch vendor"}) return JsonResponse({"success": False, "message": "Invalid request method."}) @login_required def show_info(request): if request.method == 'POST': switch_id = request.POST.get('switch_id') username = request.POST.get('username') password = request.POST.get('password') switch = get_object_or_404(Switch, pk=switch_id) if switch.model.vendor == 'Raisecom': try: raisecom_telnet = RaisecomTelnet(switch.ip_address, username, password) info = raisecom_telnet.show_info() # Створення запису в моделі Event Event.objects.create( switch=switch, event_type="Show Info", detail="\n".join([f"Command: {cmd}\n Response: {'*'*20}\n {resp.strip()} \n{'*'*20}" for cmd, resp in raisecom_telnet.command_log]), user=request.user ) return JsonResponse({"success": True, "info": clean_response(info)}) except Exception as e: return JsonResponse({"success": False, "message": str(e)}) else: return JsonResponse({"success": False, "message": "Unsupported switch vendor"}) return JsonResponse({"success": False, "message": "Invalid request method."}) @login_required def configure_switch(request): if request.method == 'POST': switch_id = request.POST.get('switch_id') new_sw = request.POST.get('new_sw') vlan = request.POST.get('vlan') gateway = request.POST.get('gateway') username = request.POST.get('username') password = request.POST.get('password') switch = get_object_or_404(Switch, pk=switch_id) if switch.model.vendor == 'Raisecom': try: raisecom_telnet = RaisecomTelnet(switch.ip_address, username, password)
```

raisecom_telnet.configure_switch(new_sw, 'path/to/commands_file.txt', vlan, gateway) return JsonResponse({"success": True, "message": "Switch configured successfully."}) except Exception as e: return JsonResponse({"success": False, "message": str(e)}) else: return JsonResponse({"success": False, "message": "Unsupported switch vendor"}) return JsonResponse({"success": False, "message": "Invalid request method."}) @login_required def event_detail(request, event_id): event = get_object_or_404(Event, pk=event_id) return JsonResponse({ "success": True, "detail": event.detail }) @login_required def show_vlan(request): if request.method == 'POST': switch_id = request.POST.get('switch_id') vlan_id = request.POST.get('vlan_id') username = request.POST.get('username') password = request.POST.get('password') if not switch_id or not username or not password: return JsonResponse({"success": False, "message": "Username and Password are required."}) switch = get_object_or_404(Switch, pk=switch_id) if switch.model.vendor == 'Raisecom': try: raisecom_telnet = RaisecomTelnet(switch.ip_address, username, password) info = raisecom_telnet.show_vlan(vlan_id) # Створення запису в моделі Event Event.objects.create(switch=switch, event_type="Show VLAN", detail="\n".join([f"Command: {cmd}, Response: {resp}" for cmd, resp in raisecom_telnet.command_log]), user=request.user) cleaned_info = clean_response(info) return JsonResponse({"success": True, "info": cleaned_info}) except PermissionError as pe: return JsonResponse({"success": False, "message": str(pe)}) except Exception as e: return JsonResponse({"success": False, "message": str(e)}) else: return JsonResponse({"success": False, "message": "Unsupported switch vendor"}) return JsonResponse({"success": False, "message": "Invalid request method."}) def clean_response(response): lines = response.split('\n') cleaned_lines = [line.strip() for line in lines if line.strip()] return '\n'.join(cleaned_lines) @login_required def edit_script(request, script_id): script = get_object_or_404(AutomationScript, pk=script_id) if request.method == 'POST': script.name = request.POST.get('name') script.description = request.POST.get('description') command_file_content = request.POST.get('command_file') # Оновлення файлу зі скриптами with open(script.command_file.path, 'w') as file: file.write(command_file_content) script.save() return JsonResponse({"success": True, "message": "Script updated successfully."}) return JsonResponse({"success": False, "message": "Invalid request method."})

base_telnet.py:

class BaseTelnet: def __init__(self, host, username, password, enable_password=None): self.host = host self.username = username self.password = password self.enable_password = enable_password self.telnet = telnetlib.Telnet(host) def to_bytes(self, text): return text.encode('ascii') + b'\n' def get_mac(self, string): return re.search(r"([0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4}\.[0-9A-Fa-f]{4})", str(string)) def login(self, login_prompt=b'Login:', password_prompt=b'Password:'): self.telnet.read_until(login_prompt, timeout=5) self.telnet.write(self.to_bytes(self.username)) self.telnet.read_until(password_prompt, timeout=5) self.telnet.write(self.to_bytes(self.password)) if self.enable_password: self.telnet.write(b"enable\n") self.telnet.read_until(b"Password: ", timeout=5) self.telnet.write(self.to_bytes("\n")) self.telnet.write(self.to_bytes("\n")) def send_command(self, command, delay=1): self.telnet.write(self.to_bytes(command)) time.sleep(delay) return self.telnet.read_very_eager().decode('utf-8') def close(self): self.telnet.close() def add_vlan(self, vlan_id, tagged_port, untagged_port): def validate_ports(ports, min_port, max_port): for port in ports: if not (min_port <= port <= max_port): raise ValueError(f"Invalid port number: {port}. Must be between {min_port} and {max_port}") validate_ports(tagged_port, 25, 28) validate_ports(untagged_port, 1, 24) raise NotImplementedError("This method should be overridden by subclasses") def delete_vlan(self, vlan_id): raise NotImplementedError("This method should be overridden by subclasses") def show_info(self): raise NotImplementedError("This method should be overridden by subclasses")

raisecom_telnet.py:

from .base_telnet import BaseTelnet class RaisecomTelnet(BaseTelnet): def __init__(self, host, username, password): super().__init__(host, username, password) self.command_log = [] def log_command(self, command, response): cleaned_response = self.clean_response(response) self.command_log.append((command, cleaned_response)) def clean_response(self, response): lines = response.split('\n') cleaned_lines = [line.strip() for line in lines if line.strip()] return '\n'.join(cleaned_lines) def get_version(self): return self.send_command(b"show version") def configure_switch(self, new_sw, commands_file, vlan, gateway): self.login(login_prompt=b'Login:', password_prompt=b'Password:') sw1_ver = self.get_version() self.send_command("telnet 192.168.0.1") self.login(login_prompt=b'Login:', password_prompt=b'Password:') sw2_ver = self.get_version() if self.get_mac(sw1_ver) != self.get_mac(sw2_ver): with open(commands_file, "r") as file: commands = file.readlines() for command in commands: command = command.strip() if command.startswith("vlan"): command = f"vlan {vlan}" elif command.startswith("interface vlan"): command = f"interface vlan {vlan}" elif command.startswith("ip address"): command = f"ip address {new_sw} 255.255.255.0" elif command.startswith("ip route"): command = f"ip route 0.0.0.0 0.0.0.0 {gateway}" if command: output = self.send_command(command) if "'y' to confirm" in output: self.telnet.write(self.to_bytes("y")) elif "'yes' to

```
confirm" in output: self.telnet.write(self.to_bytes("yes")) elif "Raisecom" in output: ... else: break self.telnet.write(b"exit\n") self.log_command(command, output) self.close() def add_vlan(self, vlan_id, tagged_port, untagged_port): try: self.login() commands = [ f"conf", f"vlan {vlan_id}", f'exit', f"interface ten 1/1/{tagged_port}", f"switchport mode trunk", f"switchport trunk allowed vlan add {vlan_id}", f"interface gi 1/1/{untagged_port}", f"switchport mode access", f"switchport access vlan {vlan_id}", f'exit' ] for command in commands: info = self.send_command(command)
self.log_command(command, info) self.close() return "VLAN added successfully" except Exception as e: self.close() raise def delete_vlan(self, vlan_id): try: self.login() commands = [ f"conf", f"no vlan {vlan_id}", f"exit" ] for command in commands: response = self.send_command(command) self.log_command(command, response) self.close() return f"VLAN deleted successfully." except Exception as e: self.close() raise def show_vlan(self, vlan_id): try: self.login() command = f"show vlan {vlan_id}" response = self.send_command(command) self.log_command(command, response) self.close() return response except PermissionError as pe: self.close() return f"Authentication error: {str(pe)}" except Exception as e: self.close() raise def show_info(self): try: self.login() command = "show version" response = self.send_command("show version") self.log_command(command, response) self.close() return response except Exception as e: self.close() raise def disable_port(self, port): try: self.login() commands = [ f"conf", f"interface gigabitethernet 1/1/{port}", f"shutdown", f"exit" ] for command in commands: self.send_command(command) self.close() return "Port administrative disable" except Exception as e: self.close() raise def enable_port(self, port): try: self.login() commands = [ f"conf", f"interface gigabitethernet 1/1/{port}", f"no shutdown", f"exit" ] for command in commands: self.send_command(command) self.close() return "Port administrative enable" except Exception as e: self.close() raise def execute_script(self, file_path): try: self.login() with open(file_path, 'r') as f: commands = f.readlines() full_respones = '' for command in commands: command = command.strip() if command: self.telnet.write(self.to_bytes(command)) response = self.telnet.read_until(b"#", timeout=3).decode('utf-8') while '--More--' in response: full_respones += response.replace('--More--', '') self.telnet.write(self.to_bytes(' ')) response = self.telnet.read_until(b"--More--",
timeout=3).decode('utf-8') full_respones += response self.log_command(command, response) self.close() return full_respones except Exception as e: self.close() raise
```

base.html:

```
<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>{% block title %}SMT{% endblock %}</title> <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet"> <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css"> {% comment %} <link rel="stylesheet" href="{% static 'css/style.css' %}"> {% endcomment %} <style> .sidebar { height: 100vh; background-color: #212529; padding-top: 20px; border-right: 1px solid #343a40; } .nav-link { color: #adb5bd; } .nav-link.active { color: white; } .content { padding: 20px; border-left: 1px solid #343a40; } .navbar { background-color: #343a40; } .navbar-brand { font-weight: bold; } .table-dark { color: #dee2e6; } .bottom-nav { margin-top: auto; } .logout-link a { color: #adb5bd; text-decoration: none; } .logout-link a:hover { color: white; } </style> {% block extra_head %} {% endblock %} </head> <body class="bg-dark text-light"> <nav class="navbar navbar-expand-lg navbar-dark"> <div class="container-fluid"> <a class="navbar-brand" href="#">Switch managment tools</a> </div> </nav> <div class="container-fluid"> <div class="row"> <nav class="col-md-2 sidebar"> <ul class="nav flex-column"> <li class="nav-item"> <a class="nav-link" href="{% url 'dashboard' %}">Dashboard</a> </li> <li class="nav-item"> <a class="nav-link" href="{% url 'switch_list' %}">Switches</a> </li> <li class="nav-item"> <a class="nav-link" href="{% url 'automation' %}">Automation</a> </li> <li class="nav-item"> <a class="nav-link" href="{% url 'logs' %}">Logs</a> </li> </ul> <div class="bottom-nav"> <ul class="nav flex-column"> <!-- Додаткові кнопки можуть бути додані тут --> <li class="nav-item"> <a class="nav-link" href="{% url 'logout' %}" onclick="event.preventDefault();
document.getElementById('logout-form').submit();">Logout</a> </li> </ul> <form id="logout-form" action="{% url 'logout' %}" method="post" style="display: none;"> {% csrf_token %} </form> </div> </nav> <main class="col-md-10 content"> {% block content %} {% endblock %} </main> </div> </div> <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script> {% block extra_js %} {% endblock %} </body> </html>
```

dashboard.html :

```
{% extends 'base.html' %} {% block title %}Dashboard{% endblock %} {% block content %} <h2>Dashboard</h2> <div class="row justify-content-center mb-4"> <div class="col-md-4"> <div class="card text-center bg-primary text-white"> <div class="card-body"> <h3>Devices</h3> <div class="circle-diagram"> <h3>{{ switch_count }}</h3> </div> </div> </div> </div> <div class="col-md-4"> <div class="card text-center bg-success text-white"> <div class="card-body"> <h3>Tasks</h3> <div class="circle-diagram"> <h3>{{ event_count }}</h3> </div> </div> </div> </div> </div> <h3>System Logs</h3> <table class="table table-dark table-striped table-hover"> <thead> <tr> <th>Time</th> <th>Host</th> <th>Action</th> <th>User</th> <th></th> </tr> </thead> <tbody> {% for event in
```

```
events %} <tr> <td>{{ event.occurred_at }}</td> <td>{{ event.switch.ip_address }}</td> <td>{{ event.event_type }}</td> <td>{% if event.user %}{{ event.user.username }}{% else %}-{% endif %}</td> <td> <button class="btn btn-sm btn-outline-info" data-bs-toggle="modal" data-bs-target="#eventDetailModal" data-event-id="{{ event.id }}"><i class="bi bi-journal-text"></i></button> </td> </tr> {% empty %} <tr> <td colspan="5">No events available.</td> </tr> {% endfor %} </tbody> </table> <!-- Event Detail Modal --> <div class="modal fade" id="eventDetailModal" tabindex="-1" aria-labelledby="eventDetailModalLabel" aria-hidden="true"> <div class="modal-dialog modal-dialog-centered"> <div class="modal-content bg-dark text-light"> <div class="modal-header"> <h5 class="modal-title" id="eventDetailModalLabel">Event Details</h5> <button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal" aria-label="Close"></button> </div> <div class="modal-body"> <div id="eventDetailContent"></div> </div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> </div> </div> </div> </div> {% endblock %} {% block extra_js %} <script> document.addEventListener('DOMContentLoaded', function () { var eventDetailModal = document.getElementById('eventDetailModal'); eventDetailModal.addEventListener('show.bs.modal', function (event) { var button = event.relatedTarget; var eventId = button.getAttribute('data-event-id'); fetch(`/logs/event_detail/${eventId}/`, { method: 'GET', headers: { 'X-Requested-With': 'XMLHttpRequest', }, }) .then(response => response.json()) .then(data => { var eventDetailContent = document.getElementById('eventDetailContent'); if (data.success) { eventDetailContent.innerText = data.detail; } else { eventDetailContent.innerText = 'Failed to load event details.'; } }) .catch(error => { var eventDetailContent = document.getElementById('eventDetailContent'); eventDetailContent.innerText = 'An error occurred. Please try again.'; }); }); }); </script> {% endblock %}
```

automation.html :

```
{% extends 'base.html' %} {% block title %}Automation{% endblock %} {% block content %} <h2>Automation</h2> <div class="mb-3"> <button class="btn btn-primary me-2" data-bs-toggle="modal" data-bs-target="#uploadScriptModal"><i class="bi bi-upload"></i> Upload Script</button> <button class="btn btn-warning" data-bs-toggle="modal" data-bs-target="#downloadScriptModal"><i class="bi bi-download"></i> Download Script</button> <button class="btn btn-success" data-bs-toggle="modal" data-bs-target="#fastTaskModal"><i class="bi bi-lightning-fill"></i> Fast Task</button> </div> <table class="table table-dark table-striped"> <thead> <tr> <th>Name</th> <th>Description</th> <th>For Switch</th> <th>Last Update</th> <th></th> </tr> </thead> <tbody> {% for script in scripts %} <tr> <td>{{ script.name }}</td> <td>{{ script.description }}</td> <td>{{ script.for_model }}</td> <td>{{ script.updated_at }}</td> <td> <button class="btn btn-sm btn-outline-success" data-bs-toggle="modal" data-bs-target="#executeScriptModal{{ script.id }}"><i class="bi bi-play-fill"></i></button> <button class="btn btn-sm btn-outline-warning" data-bs-toggle="modal" data-bs-target="#editScriptModal{{ script.id }}"><i class="bi bi-journal-text"></i></button> <a href="{% url 'download_script' script.pk %}" class="btn btn-sm btn-outline-primary"><i class="bi bi-download"></i></a> <form action="{% url 'delete_script' script.pk %}" method="post" style="display:inline;"> {% csrf_token %} <button type="submit" class="btn btn-sm btn-outline-danger"><i class="bi bi-trash"></i></button> </form> </td> </tr> <!-- Execute Script Modal --> <div class="modal fade" id="executeScriptModal{{ script.id }}" tabindex="-1" aria-labelledby="executeScriptModalLabel{{ script.id }}" aria-hidden="true"> <div class="modal-dialog modal-dialog-centered"> <div class="modal-content bg-dark text-light"> <div class="modal-header"> <h5 class="modal-title" id="executeScriptModalLabel{{ script.id }}">Execute Script</h5> <button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal" aria-label="Close"></button> </div> <div class="modal-body"> <form method="post" action="{% url 'execute_script' script.id %}" class="execute-script-form" data-script-id="{{ script.id }}"> {% csrf_token %} <div class="mb-3"> <label for="switchSelect{{ script.id }}" class="form-label">Select Switch</label> <select class="form-select" id="switchSelect{{ script.id }}" name="switch_id"> <option value="" disabled selected>Select a switch</option> {% for switch in switches %} <option value="{{ switch.id }}">{{ switch.name }} ({{ switch.ip_address }})</option> {% endfor %} </select> </div> <div class="mb-3"> <label for="username{{ script.id }}" class="form-label">Username</label> <input type="text" class="form-control" id="username{{ script.id }}" name="username" required> </div> <div class="mb-3"> <label for="password{{ script.id }}" class="form-label">Password</label> <input type="password" class="form-control" id="password{{ script.id }}" name="password" required> </div> <div class="mb-3 text-success" id="successMessage{{ script.id }}" style="display: none;">Script executed successfully.</div> <div class="mb-3 text-danger" id="errorMessage{{ script.id }}" style="display: none;"></div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Execute</button> </div> </form> </div> </div> </div> </div> <!-- Edit Script Modal --> <div class="modal fade" id="editScriptModal{{
```

```
script.id }}" tabindex="-1" aria-labelledby="editScriptModalLabel{{ script.id }}" aria-hidden="true"> <div class="modal-dialog modal-dialog-centered"> <div class="modal-content bg-dark text-light"> <div class="modal-header"> <h5 class="modal-title" id="editScriptModalLabel{{ script.id }}">Edit Script</h5> <button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal" aria-label="Close"></button> </div> <div class="modal-body"> <form method="post" action="{% url 'edit_script' script.id %}" class="edit-script-form" data-script-id="{{ script.id }}"> {% csrf_token %} <div class="mb-3"> <label for="scriptName{{ script.id }}" class="form-label">Script Name</label> <input type="text" class="form-control" id="scriptName{{ script.id }}" name="name" value="{{ script.name }}" required> </div> <div class="mb-3"> <label for="scriptDescription{{ script.id }}" class="form-label">Description</label> <textarea class="form-control" id="scriptDescription{{ script.id }}" name="description" rows="3" required>{{ script.description }}</textarea> </div> <div class="mb-3"> <label for="scriptContent{{ script.id }}" class="form-label">Script Content</label> <textarea class="form-control" id="scriptContent{{ script.id }}" name="command_file" rows="10" required>{{ script.command_file.read|safe }}</textarea> </div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Save changes</button> </div> </form> </div> </div> </div> </div> {% endfor %} </tbody> </table> <div class="modal fade" id="uploadScriptModal" tabindex="-1" aria-labelledby="uploadScriptModalLabel" aria-hidden="true"> <div class="modal-dialog modal-dialog-centered"> <div class="modal-content bg-dark text-light"> <div class="modal-header"> <h5 class="modal-title" id="uploadScriptModalLabel">Upload Script</h5> <button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal" aria-label="Close"></button> </div> <div class="modal-body"> <form method="post" action="{% url 'upload_script' %}" enctype="multipart/form-data"> {% csrf_token %} {{ form.as_p }} <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Upload</button> </div> </form> </div> </div> </div> </div> <div class="modal fade" id="downloadScriptModal" tabindex="-1" aria-labelledby="downloadScriptModalLabel" aria-hidden="true"> <div class="modal-dialog modal-dialog-centered"> <div class="modal-content bg-dark text-light"> <div class="modal-header"> <h5 class="modal-title" id="downloadScriptModalLabel">Download Script</h5> <button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal" aria-label="Close"></button> </div> <div class="modal-body"> <p>Select a script to download:</p> <ul class="list-group"> {% for script in scripts %} <li class="list-group-item bg-dark text-light"> <a href="{% url 'download_script' script.pk %}" class="text-light">{{ script.name }}</a> </li> {% empty %} <li class="list-group-item bg-dark text-light">No scripts available.</li> {% endfor %} </ul> </div> </div> </div> </div> <!-- Fast Task Modal --> <div class="modal fade" id="fastTaskModal" tabindex="-1" aria-labelledby="fastTaskModalLabel" aria-hidden="true"> <div class="modal-dialog modal-dialog-centered modal-lg"> <div class="modal-content bg-dark text-light"> <div class="modal-header"> <h5 class="modal-title" id="fastTaskModalLabel">Fast Task</h5> <button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal" aria-label="Close"></button> </div> <div class="modal-body"> <form id="commonCredentialsForm"> <div class="mb-3"> <label for="commonUsername" class="form-label">Username</label> <input type="text" class="form-control" id="commonUsername" name="username" required> </div> <div class="mb-3"> <label for="commonPassword" class="form-label">Password</label> <input type="password" class="form-control" id="commonPassword" name="password" required> </div> </form> <ul class="nav nav-tabs" id="fastTaskTab" role="tablist"> <li class="nav-item" role="presentation"> <a class="nav-link active" id="show-vlan-tab" data-bs-toggle="tab" href="#show-vlan" role="tab" aria-controls="show-vlan" aria-selected="true">Show VLAN</a> </li> <li class="nav-item" role="presentation"> <a class="nav-link" id="add-vlan-tab" data-bs-toggle="tab" href="#add-vlan" role="tab" aria-controls="add-vlan" aria-selected="false">Add VLAN</a> </li> <li class="nav-item" role="presentation"> <a class="nav-link" id="delete-vlan-tab" data-bs-toggle="tab" href="#delete-vlan" role="tab" aria-controls="delete-vlan" aria-selected="false">Delete VLAN</a> </li> <li class="nav-item" role="presentation"> <a class="nav-link" id="show-info-tab" data-bs-toggle="tab" href="#show-info" role="tab" aria-controls="show-info" aria-selected="false">Show Info</a> </li> <li class="nav-item" role="presentation"> <a class="nav-link" id="configure-switch-tab" data-bs-toggle="tab" href="#configure-switch" role="tab" aria-controls="configure-switch" aria-selected="false">Configure Switch</a> </li> </ul> <div class="tab-content" id="fastTaskTabContent"> <!-- Show VLAN --> <div class="tab-pane fade show active" id="show-vlan" role="tabpanel" aria-labelledby="show-vlan-tab"> <form id="showVlanForm" method="post"> {%
```

```
csrf_token %} <div class="mb-3"> <label for="showVlanSwitch" class="form-label">Select Switch</label> <select class="form-select" id="showVlanSwitch" name="switch_id" required> <option value="" disabled selected>Select a switch</option> {% for switch in switches %} <option value="{{ switch.id }}">{{ switch.name }} ({{ switch.ip_address }})</option> {% endfor %} </select> </div> <div class="mb-3"> <label for="vlanId" class="form-label">VLAN ID</label> <input type="number" class="form-control" id="vlanId" name="vlan_id" min="1" max="4094" required> </div> <div class="mb-3 text-success" id="showVlanSuccess" style="display: none;">VLAN information retrieved successfully.</div> <div class="mb-3 text-danger" id="showVlanError" style="display: none;"></div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Show VLAN</button> </div> </form> <div id="vlanInfo" class="mt-3"></div> </div> <!-- Інші вкладки --> <div class="tab-pane fade" id="add-vlan" role="tabpanel" aria-labelledby="add-vlan-tab"> <form id="addVlanForm" method="post"> {% csrf_token %} <div class="mb-3"> <label for="addVlanSwitch" class="form-label">Select Switch</label> <select class="form-select" id="addVlanSwitch" name="switch_id" required> <option value="" disabled selected>Select a switch</option> {% for switch in switches %} <option value="{{ switch.id }}">{{ switch.name }} ({{ switch.ip_address }})</option> {% endfor %} </select> </div> <div class="mb-3"> <label for="vlanId" class="form-label">VLAN ID</label> <input type="number" class="form-control" id="vlanId" name="vlan_id" min="1" max="4094" required> </div> <div class="mb-3"> <label for="taggedPort" class="form-label">Tagged Port (25-28)</label> <input type="number" class="form-control" id="taggedPort" name="tagged_port" min="25" max="28" required> </div> <div class="mb-3"> <label for="untaggedPort" class="form-label">Untagged Port (1-24)</label> <input type="number" class="form-control" id="untaggedPort" name="untagged_port" min="1" max="24" required> </div> <div class="mb-3 text-success" id="addVlanSuccess" style="display: none;">VLAN added successfully.</div> <div class="mb-3 text-danger" id="addVlanError" style="display: none;"></div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Add VLAN</button> </div> </form> </div> <div class="tab-pane fade" id="delete-vlan" role="tabpanel" aria-labelledby="delete-vlan-tab"> <form id="deleteVlanForm" method="post"> {% csrf_token %} <div class="mb-3"> <label for="deleteVlanSwitch" class="form-label">Select Switch</label> <select class="form-select" id="deleteVlanSwitch" name="switch_id" required> <option value="" disabled selected>Select a switch</option> {% for switch in switches %} <option value="{{ switch.id }}">{{ switch.name }} ({{ switch.ip_address }})</option> {% endfor %} </select> </div> <div class="mb-3"> <label for="deleteVlanId" class="form-label">VLAN ID</label> <input type="number" class="form-control" id="deleteVlanId" name="vlan_id" min="1" max="4094" required> </div> <div class="mb-3 text-success" id="deleteVlanSuccess" style="display: none;">VLAN deleted successfully.</div> <div class="mb-3 text-danger" id="deleteVlanError" style="display: none;"></div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Delete VLAN</button> </div> </form> </div> <div class="tab-pane fade" id="show-info" role="tabpanel" aria-labelledby="show-info-tab"> <form id="showInfoForm" method="post"> {% csrf_token %} <div class="mb-3"> <label for="showInfoSwitch" class="form-label">Select Switch</label> <select class="form-select" id="showInfoSwitch" name="switch_id" required> <option value="" disabled selected>Select a switch</option> {% for switch in switches %} <option value="{{ switch.id }}">{{ switch.name }} ({{ switch.ip_address }})</option> {% endfor %} </select> </div> <div class="mb-3 text-success" id="showInfoSuccess" style="display: none;">Information retrieved successfully.</div> <div class="mb-3 text-danger" id="showInfoError" style="display: none;"></div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Show Info</button> </div> </form> <div id="switchInfo" class="mt-3"></div> </div> <div class="tab-pane fade" id="configure-switch" role="tabpanel" aria-labelledby="configure-switch-tab"> <form id="configureSwitchForm" method="post"> {% csrf_token %} <div class="mb-3"> <label for="configureSwitch" class="form-label">Select Switch</label> <select class="form-select" id="configureSwitch" name="switch_id" required> <option value="" disabled selected>Select a switch</option> {% for switch in switches %} <option value="{{ switch.id }}">{{ switch.name }} ({{ switch.ip_address }})</option> {% endfor %} </select> </div> <div class="mb-3"> <label for="newSwitchIP" class="form-label">New Switch IP</label> <input type="text" class="form-control" id="newSwitchIP" name="new_sw" required> </div> <div class="mb-3"> <label for="vlan" class="form-label">VLAN</label> <input type="number" class="form-control" id="vlan" name="vlan" min="1" max="4094" required> </div> <div class="mb-3"> <label for="gateway" class="form-label">Gateway</label> <input type="text" class="form-control" id="gateway" name="gateway" required> </div> <div class="mb-3
```

```
text-success" id="configureSwitchSuccess" style="display: none;">Switch configured successfully.</div> <div class="mb-3 text-danger" id="configureSwitchError" style="display: none;"></div> <div class="modal-footer"> <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button> <button type="submit" class="btn btn-primary">Configure</button> </div> </form> </div> </div> </div> </div> </div> </div> {% endblock %} {% block extra_js %} <script> document.addEventListener('DOMContentLoaded', function () { const forms = document.querySelectorAll('.execute-script-form, .edit-script-form'); forms.forEach(function (form) { form.addEventListener('submit', function (event) { event.preventDefault(); const formData = new FormData(form); const scriptId = form.getAttribute('data-script-id'); const successMessage = document.getElementById(`successMessage${scriptId}`); const errorMessage = document.getElementById(`errorMessage${scriptId}`); fetch(form.action, { method: 'POST', body: formData, headers: { 'X-Requested-With': 'XMLHttpRequest', }, }) .then(response => response.json()) .then(data => { if (data.success) { successMessage.style.display = 'block'; errorMessage.style.display = 'none'; } else { errorMessage.innerText = data.message; errorMessage.style.display = 'block'; successMessage.style.display = 'none'; } }) .catch(error => { errorMessage.innerText = 'An error occurred. Please try again.'; errorMessage.style.display = 'block'; successMessage.style.display = 'none'; }); }); }); // Add VLAN const addVlanForm = document.getElementById('addVlanForm'); addVlanForm.addEventListener('submit', function (event) { event.preventDefault(); const formData = new FormData(addVlanForm); const username = document.getElementById('commonUsername').value; const password = document.getElementById('commonPassword').value; formData.append('username', username); formData.append('password', password); const successMessage = document.getElementById('addVlanSuccess'); const errorMessage = document.getElementById('addVlanError'); // Додаємо індикатор завантаження const loader = document.createElement('div'); loader.classList.add('spinner-border', 'text-light'); loader.setAttribute('role', 'status'); const button = addVlanForm.querySelector('button[type="submit"]'); button.disabled = true; button.innerHTML = ''; button.appendChild(loader); fetch('{% url "add_vlan" %}', { method: 'POST', body: formData, headers: { 'X-Requested-With': 'XMLHttpRequest', }, }) .then(response => response.json()) .then(data => { // Приховуємо індикатор завантаження button.removeChild(loader); button.disabled = false; button.innerHTML = 'Add VLAN'; if (data.success) { successMessage.style.display = 'block'; errorMessage.style.display = 'none'; } else { errorMessage.innerText = data.message; errorMessage.style.display = 'block'; successMessage.style.display = 'none'; } }) .catch(error => { // Приховуємо індикатор завантаження button.removeChild(loader); button.disabled = false; button.innerHTML = 'Add VLAN'; errorMessage.innerText = 'An error occurred. Please try again.'; errorMessage.style.display = 'block'; successMessage.style.display = 'none'; }); }); // Delete VLAN const deleteVlanForm = document.getElementById('deleteVlanForm'); deleteVlanForm.addEventListener('submit', function (event) { event.preventDefault(); const formData = new FormData(deleteVlanForm); const username = document.getElementById('commonUsername').value; const password = document.getElementById('commonPassword').value; formData.append('username', username); formData.append('password', password); const successMessage = document.getElementById('deleteVlanSuccess'); const errorMessage = document.getElementById('deleteVlanError'); // Додаємо індикатор завантаження const loader = document.createElement('div'); loader.classList.add('spinner-border', 'text-light'); loader.setAttribute('role', 'status'); const button = deleteVlanForm.querySelector('button[type="submit"]'); button.disabled = true; button.innerHTML = ''; button.appendChild(loader); fetch('{% url "delete_vlan" %}', { method: 'POST', body: formData, headers: { 'X-Requested-With': 'XMLHttpRequest', }, }) .then(response => response.json()) .then(data => { // Приховуємо індикатор завантаження button.removeChild(loader); button.disabled = false; button.innerHTML = 'Delete VLAN'; if (data.success) { successMessage.style.display = 'block'; errorMessage.style.display = 'none'; } else { errorMessage.innerText = data.message; errorMessage.style.display = 'block'; successMessage.style.display = 'none'; } }) .catch(error => { // Приховуємо індикатор завантаження button.removeChild(loader); button.disabled = false; button.innerHTML = 'Delete VLAN'; errorMessage.innerText = 'An error occurred. Please try again.'; errorMessage.style.display = 'block'; successMessage.style.display = 'none'; }); }); // Show Info const showInfoForm = document.getElementById('showInfoForm'); showInfoForm.addEventListener('submit', function (event) { event.preventDefault(); const formData = new FormData(showInfoForm); const username = document.getElementById('commonUsername').value; const password = document.getElementById('commonPassword').value; formData.append('username', username); formData.append('password', password); const successMessage = document.getElementById('showInfoSuccess'); const errorMessage = document.getElementById('showInfoError'); const switchInfo = document.getElementById('switchInfo'); // Додаємо індикатор завантаження const loader =
```

```
document.createElement('div');
loader.classList.add('spinner-border', 'text-light');
loader.setAttribute('role', 'status'); const button =
showInfoForm.querySelector('button[type="submit"]');
button.disabled = true; button.innerHTML = '';
button.appendChild(loader); fetch('{% url "show_info"
%}', { method: 'POST', body: formData, headers: {
'X-Requested-With': 'XMLHttpRequest', }, })
.then(response => response.json()) .then(data => { //
Приховуємо індикатор завантаження
button.removeChild(loader); button.disabled = false;
button.innerHTML = 'Show Info'; if (data.success) {
successMessage.style.display = 'block';
errorMessage.style.display = 'none';
switchInfo.innerText = data.info; } else {
errorMessage.innerText = data.message;
errorMessage.style.display = 'block';
successMessage.style.display = 'none';
switchInfo.innerText = ''; } }) .catch(error => { //
Приховуємо індикатор завантаження
button.removeChild(loader); button.disabled = false;
button.innerHTML = 'Show Info';
errorMessage.innerText = 'An error occurred. Please
try again.'; errorMessage.style.display = 'block';
successMessage.style.display = 'none';
switchInfo.innerText = ''; }); }); // Show VLAN const
showVlanForm =
document.getElementById('showVlanForm');
showVlanForm.addEventListener('submit', function
(event) { event.preventDefault(); const formData =
new FormData(showVlanForm); const username =
document.getElementById('commonUsername').value;
const password =
document.getElementById('commonPassword').value;
formData.append('username', username);
formData.append('password', password); const
successMessage =
document.getElementById('showVlanSuccess'); const
errorMessage =
document.getElementById('showVlanError'); const
vlanInfo = document.getElementById('vlanInfo'); //
Додаємо індикатор завантаження const loader =
document.createElement('div');
loader.classList.add('spinner-border', 'text-light');
loader.setAttribute('role', 'status'); const button =
showVlanForm.querySelector('button[type="submit"]')
; button.disabled = true; button.innerHTML = '';
button.appendChild(loader); fetch('{% url "show_vlan"
%}', { method: 'POST', body: formData, headers: {
'X-Requested-With': 'XMLHttpRequest', }, })
.then(response => response.json()) .then(data => { //
Приховуємо індикатор завантаження
button.removeChild(loader); button.disabled = false;
button.innerHTML = 'Show VLAN'; if (data.success) {
successMessage.style.display = 'block';
errorMessage.style.display = 'none'; vlanInfo.innerText
= data.info; } else { errorMessage.innerText =
data.message; errorMessage.style.display = 'block';
successMessage.style.display = 'none';
vlanInfo.innerText = ''; } }) .catch(error => { //
Приховуємо індикатор завантаження
button.removeChild(loader); button.disabled = false;
button.innerHTML = 'Show VLAN';
errorMessage.innerText = 'An error occurred. Please
try again.'; errorMessage.style.display = 'block';
successMessage.style.display = 'none';
vlanInfo.innerText = ''; }); }); }); </script> {%
endblock %}
```

utils.py :

```
import subprocess def is_switch_online(ip_address):
try: subprocess.check_output(['ping', '-c', '1', '-W', '1',
ip_address]) return True except
subprocess.CalledProcessError: return False
```

update_switch_status.py:

```
from typing import Any from
django.core.management.base import BaseCommand
from smt.models import Switch from smt.utils.utils
import is_switch_online class
Command(BaseCommand): help = 'Update switch
online status' def handle(self, *args, **kwargs):
switches = Switch.objects.all() for switch in switches:
print(f'{ switch.is_online} and {switch.ip_address} and
check: {is_switch_online(switch.ip_address)}')
switch.is_online = is_switch_online(switch.ip_address)
print(f'after: {switch.is_online}') switch.save()
self.stdout.write(self.style.SUCCESS('Seccessfully
update switch statuses'))
```