**Міністерство освіти і науки України**
**Тернопільський національний технічний університет імені Івана Пулюя**

*Факультет комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету )
*Кафедра комп'ютерних систем та мереж*
(повна назва кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

*бакалавр*
(назва освітнього ступеня)

на тему: Комп'ютеризована система моделювання і документування мережевих ресурсів дата-центру

Виконав: студент *IV* курсу, групи *СІ-41*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

| | | |
|---|---|---|
| | (підпис) | Папка О. В. (прізвище та ініціали) |
| Керівник | (підпис) | Тиш Є. В. (прізвище та ініціали) |
| Нормоконтроль | (підпис) | (прізвище та ініціали) |
| Завідувач кафедри | (підпис) | Осухівська Г. В. (прізвище та ініціали) |
| Рецензент | (підпис) | (прізвище та ініціали) |

Тернопіль
2023

Міністерство освіти і науки України
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
« ____ » _____ 2023 р.

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Папці Олегу Вікторовичу*
**(прізвище, ім'я, по батькові)**

1. Тема роботи *Комп'ютеризована система моделювання і документування мережевих ресурсів дата-центру*

Керівник роботи *Тиш Євгенія Володимирівна*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*28*» *лютого* *2023* року № *4/7-238*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Розробка комп'ютеризованої система моделювання і документування мережевих ресурсів дата-центру

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1 Аналіз технічного завдання
2 Проєктна частина
3 Практична частина
4 Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 Блок-схема роботи розширення
2 Структурна схема веб-застосунку
3 Архітектура комп'ютеризованої системи
4 ER-діаграми бази даних

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Безпека життєдіяльності, основи хорони праці | *Пилипець М. І., д.т.н., проф. каф. МТ* | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Ознайомлення з завданням кваліфікаційної роботи | 30.02-05.03.2023 | *Виконано* |
| 2. | Огляд літератури за темою кваліфікаційної роботи | 05.03-20.03.2023 | *Виконано* |
| 3. | Аналіз особливостей проектування, та розробка застосунку для ІТ компанії. | 20.03-23.03.2023 | *Виконано* |
| 4. | Практична реалізація об’єкта проектування | 23.03-29.03.2023 | *Виконано* |
| 5. | Виконання завдання до підрозділу «Безпека життєдіяльності» | 29.03-05.04.2023 | *Виконано* |
| 6. | Виконання завдання до підрозділу «Основи охорони праці» | 05.04-09.04.2023 | *Виконано* |
| 7. | Оформлення кваліфікаційної роботи | 15.05-21.05.2023 | *Виконано* |
| 8. | Нормоконтроль | 12.06-13.06.2023 | *Виконано* |
| 9. | Перевірка на плагіат | 14.06-15.06.2023 | *Виконано* |
| 10. | Попередній захист кваліфікаційної роботи | 16.06-19.06.2023 | *Виконано* |
| 11. | Захист кваліфікаційної роботи | 20.06-23.06.2023 | *Виконано* |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ *Папка О. В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Тиш Є. В.*
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Комп’ютеризована система моделювання і документування мережевих ресурсів дата-центру // Кваліфікаційна робота освітнього рівня «Бакалавр» // Папка Олег Вікторович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних систем та мереж, група СІ-41 // Тернопіль, 2023 //с. - 49, рис. - 19, додат. - 2, бібліогр. - 21.

Ключові слова: модуль, фреймворк, база даних, сервер, інтерфейс, дата-центр, Python, Django.

Робота складається із вступу, чотирьох розділів, висновку, списку використаних джерел і додатків. У вступі описано чому дана тема є актуальною та важливою для ІТ-компаній будь-якого рівня. У першому розділі проведено загальний аналіз рішення та його призначення формується завдання та вимоги до проєктованої системи, та використовувані технології. У другому розділі проводиться розбір та проєктування системи її архітектури та структури та обґрунтовано даний вибір. У третьому розділі детально описано процес розробки та реалізації програмного рішення, його тестування та розгортання на сервері. У четвертому розділі описані питання із безпеки життєдіяльності, такі як естетичне оформлення робочого місця, та заходи електробезпеки при роботі з ПК. У висновках містяться поради щодо вдосконалення даного рішення та підсумовується загальний об’єм виконаної роботи.

Результатом кваліфікаційної роботи є готове розширення комп’ютеризованої системи моделювання і документування мережевих ресурсів дата-центру, написане на фреймворку Python Django.

# ABSTRACT

A computerized system for modeling and documenting data center network resources // Qualification work of the educational level «Bachelor» // Papka Oleh Viktorovych // Ivan Pulyuy Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Systems and Networks, group CI-41 // Ternopil, 2023 // pages. - 49, fig. - 19, appendix - 2, bibliography - 21.

Keywords: module, framework, database, server, interface. data center, Python, Django.

The paper consists of an introduction, four chapters, a conclusion, a list of references, and appendices. The introduction describes why this topic is relevant and important for IT companies of any level. The first section provides a general analysis of the solution and its purpose, formulating the tasks and requirements for the designed system, and the technologies used. The second section analyzes and designs the system, its architecture and structure, and justifies this choice. The third section describes in detail the process of developing and implementing a software solution, testing it, and deploying it on the server. The fourth section describes health and safety issues, such as the aesthetic design of the workplace and electrical safety measures when working with a PC. The conclusion contains tips on how to improve the solution and summarizes the total amount of work performed.

The result of the qualification work is a ready-made extension of the computerized system for modeling and documenting data center network resources, written in the Python Django framework.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ І СКОРОЧЕНЬ

КС – Комп’ютеризована система;

API – Application Programming Interface;

SSOT – Single source of truth;

MVC – Model View Controller;

MVT – Model View Template;

SLAAC – Stateless Address Autoconfiguration;

HTTP – HyperText Transfer Protocol;

HTML – HyperText Markup Language.

# ВСТУП

У сучасну цифрову епоху стрімкий розвиток Інтернету та комунікаційних технологій призвів до різкого збільшення кількості та різноманітності обладнання. Від приватних і домашніх серверів до великих центрів обробки даних транснаціональних корпорацій - усі вони використовують схожі механізми зберігання та обробки даних. Однак існує значний контраст у масштабах цих рішень. Більші системи потребують експоненціально більшої кількості обладнання для зберігання та обробки даних, а також значно розгалуженішої мережевої інфраструктури.

Традиційні методи документування цих ресурсів, такі як використання ручки та паперу або управління за допомогою електронних таблиць у популярних програмах, вже не є достатніми. Вони не здатні відобразити складність і масштаб цих складних систем, не кажучи вже про ефективне управління і планування.

Щоб заповнити цю прогалину, було розроблено концепцію комп'ютеризованої системи моделювання та документування мережевих ресурсів центрів обробки даних. Ця система покликана забезпечити централізовану, уніфіковану платформу для управління цими ресурсами, сприяючи підвищенню рівня контролю та ефективності.

Основною метою цієї кваліфікаційної роботи є розробка розширення, яке додасть додатковий рівень гнучкості до управління списками IP-адрес. Цей проект описує технології необхіді для створення, тестування та розгортання цього рішення.

Розуміючи гостру потребу в такому рішенні в сучасному цифровому полі, проект має на меті створити ефективну, зручну для користувача платформу для впорядкування документування та управління ресурсами центрів обробки даних.

# РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

## 1.1 Аналіз предметної області КС

Дата-центр (з англ. Data center) є беззаперечною основою для таких галузей, як телекомунікації, хмарні обчислення та бізнес-IT, і є важливою частиною інфраструктури для сучасних інформаційних технологій. У них розміщуються різноманітні суспільні ресурси, в тому числі системи зберігання даних, сервери, маршрутизатори, комутатори та інше обладнання, що відповідає за обробку та зберігання даних. Для ефективного управління та роботи із цими ресурсами підприємства використовують комп'ютеризовані системи моделювання і документування мережевих ресурсів.

Документування та моделювання ресурсів за допомогою КС привносить підвищення ефективності роботи. Моделювання ресурсів забезпечує оптимізацію операцій, зменшує ручні зусилля та дозволяє мережевим адміністраторам ефективно керувати та організовувати складну мережеву інфраструктуру. Також завдяки можливості моделювання та документування мережевих ресурсів, організації можуть отримати повне уявлення про інфраструктуру своїх центрів обробки даних. Така прозорість полегшує ефективний розподіл ресурсів, планування потужностей і оптимізацію, що призводить до економії коштів і підвищення продуктивності. Завдяки точному документуванню мережевих ресурсів адміністратори можуть легко виявити і виправити потенційні вузькі місця або проблеми до того, як вони вплинуть на загальну продуктивність мережі. Завдяки точній документації можна швидше виявляти та вирішувати мережеві проблеми, мінімізуючи час простою та підвищити загальну надійність інфраструктури.

Центри обробки даних обробляють також і конфіденційну інформацію, в даному випадку безпека є головним пріоритетом. Комп'ютеризовані системи забезпечують централізоване сховище для документування політик безпеки, контролю доступу та вимог відповідності. Крім того, ці системи можуть відстежувати і перевіряти зміни в мережевих конфігураціях. Оскільки центри обробки даних продовжують рости і розвиватися, важливо мати системи, які можуть пристосуватися до розширення і підтримувати планування на майбутнє. Комп’ютеризовані системи забезпечують легку масштабованість, дозволяючи адміністраторам моделювати та імітувати зміни в мережевій інфраструктурі. Ця можливість допомагає організаціям приймати обґрунтовані рішення щодо модернізації обладнання, вимог до пропускної здатності та зміни дизайну мережі цим самим продовжуючи термін експлуатації обладнання.

В основному процес документування та моделювання включає в себе два фундаментальні принципи:

– управління інфраструктурою дата-центрів (DCIM з англ. Data Center Infrastructure Management);

– управління IP-адресами (IPAM з англ. IP Address Management).

DCIM зосереджується на моніторингу, управлінні та забезпеченні видимості усієї можливої фізичної інфраструктури, що власне і надає адміністраторам можливості оптимізації.

У свою чергу IPAM відповідає за ефективне управління IP-адрес в мережі дата-центру. Адже при збільшенні кількості пристроїв та служб, що підключених до мережі, IPAM стає важливим для підтримання точних записів про призначення IP-адрес, уникнення конфліктів і забезпечення ефективного використання доступного адресного простору.

IP-адреса, або адреса Інтернет-протоколу, - це унікальний цифровий ідентифікатор, який присвоюється кожному пристрою, що бере участь у комп’ютерній мережі, яка використовує Інтернет-протокол для зв’язку.

IP-адреса виконує дві основні функції: ідентифікує хост або мережевий інтерфейс і вказує на місцезнаходження хоста в мережі. IP-адреси, як правило, записуються у зрозумілій для людини формі, наприклад, 105.16.224.11 для IPv4 і 2001:db8:0:1234:0:567:8:1 для IPv6. З іншого боку, мережевий префікс - це частина IP-адреси, яка вказує на розмір мережі. Він часто представляється у вигляді косої риски, за якою слідує число, наприклад /24, що означає, що перші 24 біти використовуються для мережевого префікса, а решта бітів - для адрес хостів у цій мережі. Такий запис мережевого префікса відомий як Classless Inter-Domain Routing (CIDR з англ. безкласова міждоменна маршрутизація) і є фундаментальним елементом управління IP-мережею, що забезпечує ефективне використання доступного простору IP-адрес.

IP-адреси складаються з двох основних частин: ідентифікатора мережі (network ID) та ідентифікатора хоста (host ID). Ідентифікатор мережі вказує на конкретну мережу в Інтернеті, тоді як ідентифікатор хоста вказує на конкретний комп’ютер (або інший мережевий пристрій) в межах цієї мережі.

В мережі, яка використовує протокол IPv4, адреси складаються з 32 бітів.

Маска підмережі (subnet mask) використовується для визначення того, яка частина IP-адреси використовується як ідентифікатор мережі, а яка - як ідентифікатор хоста. Вона складається з 32 бітів, подібно до IP-адреси, і також записується як чотири десяткових числа, відокремлені крапками. Біти, які встановлені в 1 в масці підмережі, відповідають ідентифікатору мережі в IP-адресі.

Для IPv6, який використовує 128-бітні адреси, принцип діє так само, але через більший розмір адреси та більш складний запис адрес (шістнадцятковий запис, відокремлений двокрапками), маска підмережі зазвичай вказується як кількість відповідних бітів для ідентифікатора мережі, наприклад, /64.

Протокол 4-ї версії (IPv4) був стандартом декілька десятиліть і є в основі більшості інтернет-мереж. Однак, цей протокол має обмежену кількість адрес, що спричинило проблему вичерпання IPv4 адрес. Зокрема, в IPv4 доступно

приблизно 4,3 мільярди адрес. Незважаючи на те, що це велика кількість, їх все менше залишається через зростання кількості підключених до Інтернету пристроїв.

Протокол 6-ї версії (IPv6) було створено, щоб вирішити цю проблему. Він має набагато більше адресного простору - приблизно 340 ундециліонів адрес. Більше того, IPv6 вводить кілька поліпшень, таких як автоматична конфігурація IP-адрес (що відома як Stateless Address Autoconfiguration, SLAAC) і вбудовану підтримку шифрування та інтернет-протоколу безпеки (IPSec). Однак, перехід на IPv6 має власні труднощі, як потреба в оновленні мережевого обладнання і програмного забезпечення, а також потреба в додатковому навчанні для інженерів та адміністраторів мережі.

Однією з фундаментальних основ документування КС є узагальнення усієї зібраної інформації про фізичну та мережеву інфраструктуру в одному місці. Цей принцип називається SSOT (single source of truth, з англ. єдине джерело істини). SSOT - гарантує, що всі зацікавлені сторони в організації мають доступ до узгоджених і надійних даних. Документуючи усі дані в одному централізованому місці усувається не потрібне дублювання, можливі розбіжності та плутанина, яка може виникати у разі використання кількох різних джерел збереження інформації про систему.

## 1.2 Аналіз рішень поставленого завдання

На ринку представлено в достатній кількості безліч систем для документування та моделювання мережевої та фізичної інфраструктури, як платних та повністю пропрієтарних так і розроблених спільнотою користувачів таких як: phpipam, nautobot, використання таблиць Excel, проте домінантом на ринку є NetBox завдяки своїй всеохопності, простоті використання, зручному і сучасному інтерфейсу та відкритому коду, що підтримується сторонніми розробниками та корпораціями [14]. NetBox поєднує в собі принципи DCIM та

IPAM, виступає в ролі SSOT, охоплює наступні аспекти документування та моделювання:

– пристроїв та їх місця встановлення;

– підключення пристроїв;

– віртуальних машин та кластерів;

– схеми передачі даних та провайдерів;

– управління контактами;

– збереження секретних даних (паролів доступу, файлів конфігурацій тощо.)

– Створення звітів та користувацьких задач для виконання у зазначений час доби.

Також доступні інтеграції із сторонніми програмними рішеннями та можливість розширення функціоналу вбудованими засобами персоналізації також за допомогою написання власних розширень, використання REST та GraphQL API так і власне модифікації наявного програмного забезпечення, яке знаходиться у відкритому доступі.

Поставленою задачею є розробка розширення для доповнення функціоналу КС NetBox а саме модуль IPCheck для ефективного управління списками заблокованих та розблокованих мереж IP-адрес підрядника;

Модуль має схожу архітектуру із вбудованими модулями NetBox, завдяки чому спрощується інтегрування даного рішення в наявну КС, забезпечується розширення вбудованого функціоналу потрібним замовнику рішенням , що відповідають його специфічним потребам і вимогам у зручному для користування форматі.

Структура проєктованої КС складається з апаратного та системного програмного забезпечення для функціонування КС та бази даних для зберігання усієї інформації.

Функціональні вимоги до проєктованої КС:

– можливість створення, зміни та вилучення відповідних сутностей;

– можливість зміни та вилучення сутностей групами;

– можливість зручного перегляду сутностей з усією інформацією про них;

– можливість сортування сутностей за вказаними критеріями;

– можливість пошуку сутностей за вказаними критеріями;

– можливість взаємодії з сутностями за допомогою REST API.

Для зручності відображення можливих варіантів використання КС можна використати діаграму використання. На діаграмі користувача зображено у вигляді фігурки людини, а можливі варіанти використання ним КС у вигляді еліпсів на рис. 1.1.

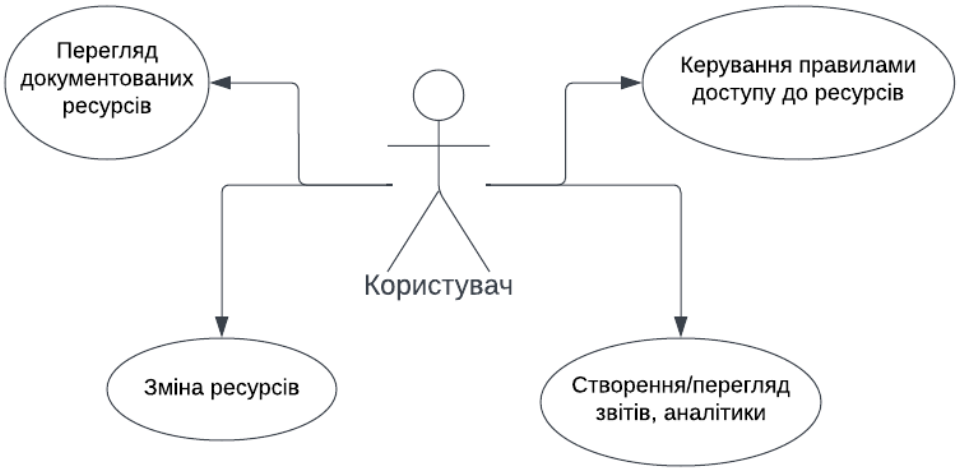


Рисунок 1.1 – Діаграма варіантів використання КС

Для функціонування проєктованої КС потрібне відповідне апаратне забезпечення, як зі сторони сервера на якому буде запущена дана система, так і зі сторони клієнта, який має користуватись системою.

Вимоги до сервера:

– Процесор із тактовою частотою не менше 2.4 ГГЦ з кількістю логічних не менше 8.

– Об’єм оперативної пам’яті не менше 16 ГБ.

– Об’єм на жорсткому диску (SSD/HDD) не менше 512 ГБ.

Вимоги до клієнтського ПК:

– Процесор із тактовою частотою не менше 2.4 ГГЦ з кількістю логічних не менше 2.

– Об’єм оперативної пам’яті не менше 4 ГБ.

– Об’єм на жорсткому диску (SSD/HDD) не менше 240 ГБ.

Програмне забезпечення КС не залежне від операційної системи як сервера так і клієнтського ПК, кросплатформенність досягнута завдяки розгортанню даного програмного забезпечення засобами Docker, та дозволяє ефективно використовувати його на будь-якому апаратному забезпеченні.

## 1.3 Використовувані технології в розробці КС

Комп’ютеризована система моделювання і документування NetBox розробляється за допомогою одного з найпопулярніших фреймворків Django на мові програмування Python, яка є однією з лідерів ринку на даний момент. NetBox працює як служба WSGI (gunicorn або uWSGI залежно від потреб користувача) на обраному HTTP-сервері (nginx або Apache залежно від потреб користувача). Також для постійного збереження усієї важливої інформації використовується реляційна база даних PostgreSQL версії 11 та новіше. Для зберігання кешованих та тимчасових даних в оперативній пам’яті, використовується база даних Redis версії 4.0 та новіше а також модуль django-rq для створення черг виконання операцій. Даний стек технологій є актуальним на сьогоднішній день та достатньо високопродуктивним для виконання поставлених задач, схематичне подання стеку технологій КС NetBox зображено на рис. 1.2.

Стек технологій, який буде використовуватись для виконання задачі чітко вивірений із бажанням замовника та наведений в таблиці 1.1 для зручного подання.

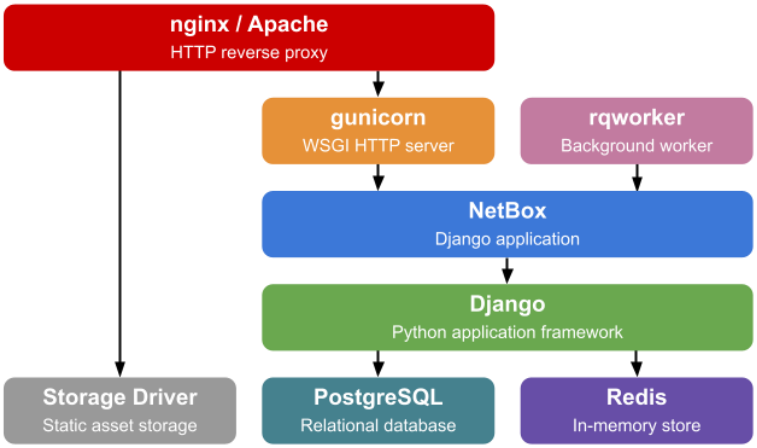


Рисунок 1.2 – Спрощена структура стеку технологій NetBox

Таблиця 1.1 – Використовувані технології при розробці

| | |
|---|---|
| WSGI HTTP сервер | gunicorn |
| Мова програмування/фреймворк | Python 3.11/Django 4.1 |
| Бази даних | PostgreSQL 14, Redis 4.0 |
| Додаткові засоби | Docker, Postman, PyCharm IDE |

Оскільки розробка та відлагодження програми здійснюється локально, то HTTP сервер не використовуватиметься та запускатиметься за допомогою вбудованого в фреймворк сервера. Даний тип сервера призначений саме для запуску в режимі розробки та не має використовуватись для готового рішення з метою забезпечення надійної роботи та безпеки даних.

Для зручності розробки використовується Docker – платформа із відкритим кодом який застосовується автоматизації розгортання масштабування та управління додатками. Docker використовувався для збірки контейнерів із базами даних PostgreSQL та Redis. Дане програмне рішення забезпечує узгодженність у різних системах, полегшує розробку та тестування, дозволяє реалізувати кросплатформенність.

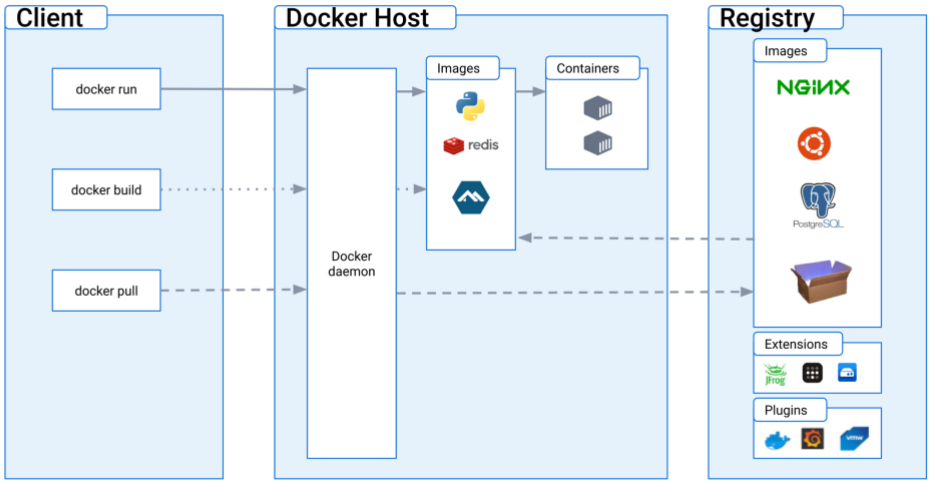


Рисунок 1.3 – Архітектура Docker

У Docker реєстри використовуються для зберігання та поширення контейнерних образів, архітектура реєстрів у Docker представлена на рис. 1.2. Розробники можуть завантажувати свої образи в публічні або приватні реєстри та використовувати їх для розгортання контейнерів. Це дозволяє зручний доступ до готових образів та просте керування інфраструктурою контейнеризації.

Postman – інструмент для тестування, відлагодження та розробки REST API. Полегшує створення запитів, валідацію відповідей для різних API ендпоінтів.

PostgreSQL – це одна з найпоширеніших об’єктно-реляційних систем баз даних з відкритим вихідним кодом, забезпечує надійність, стійкість і продуктивність завдяки контролю паралелізму та цілісності транзакцій. Дана база підтримує великий спектр типів даних, що надає розробникам більшу гнучкість при написанні програмних рішень.

Redis – сховище структур даних в пам’яті, використовується як база даних, кеш і брокер повідомлень. Дана база підтримує такі структури даних, як рядки, хеші, списки, множини тощо. Забезпечує високошвидкісні операції над цими структурами, Redis підвищує продуктивність додатків, зменшуючи час доступу до часто використовуваних даних. Основні варіанти використання Redis представлено на рис. 1.4.

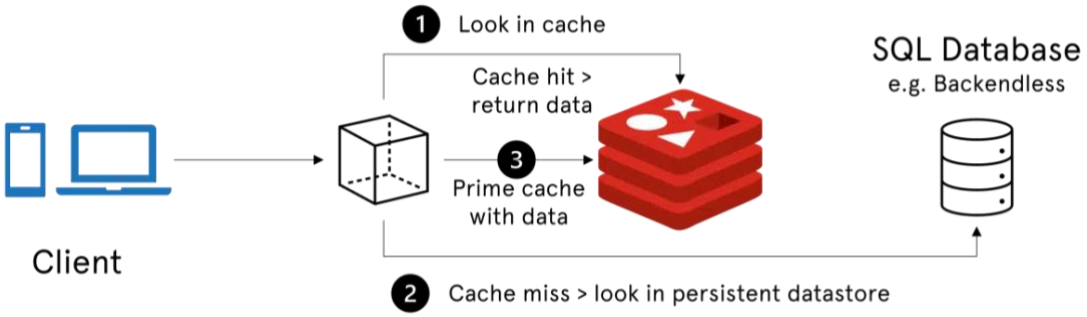


Рисунок 1.4 – Узагальнена схема використання Redis

Python Django – це високорівневий веб-фреймворк, в основі, якого лежить концепція швидкого написання чистого коду. Даний фреймворк бере на себе більшу частину усієї роботи веб-розробника, що дозволяє швидко і ефективно виконувати поставлену задачу. Django також відомий своєю надійністю та масштабованістю, що робить його придатним для високонавантажених веб-додатків.

PyCharm – популярне інтегрована середовище розробки для Python, що пропонує широкий спектр функцій для підвищення продуктивності розробників. Ця IDE має підтримку Django для веб-розробки та інтегрується з іншими мовами програмування, роблячи його незамінним інструментом для розробників Python.

# РОЗДІЛ 2 ПРОЄКТНА ЧАСТИНА

## 2.1 Опис та налаштування засобів розробки

Для початку проєктування комп'ютеризованої системи моделювання і документування мережевих ресурсів потрібно провести налаштування засобів та середовища розробки даного програмного забезпечення.

Одним з важливих аспектів розробки є створення баз даних, які забезпечать збереження і доступ до інформації. Для зручності і ефективності розробки, використано платформу Docker. В рамках однієї групи зібрано 3 контейнери із базами даних PostgreSQL та Redis та для зручності відлагодження виведено для комунікації з локальною мережею відповідні порти. Для зібрання відповідних контейнерів із базами даних використано код із офіційного репозиторію NetBox, зміни до якого представлено у лістингу 2.1.

Лістинг 2.1 – Програмний код файлу docker-compose.override.yml

```
version: '3.5'
services:
  redis:
    ports:
      - 6379:6379
  redis-cache:
    ports:
      - 6565:6379
  postgres:
    ports:
      - 5432:5432
```

Контейнер Redis із внутрішнім портом 6379 та зовнішнім портом 6379 є швидким кеш-сховищем та базою даних типу ключ-значення. Цей контейнер являє собою Redis-сервер, що доступний на порті 6379.

Контейнер Redis Cache із внутрішнім портом 6379 та зовнішнім портом 6565 теж використовується Redis, але в даному випадку використовується як кеш-сховище. Цей контейнер являє собою Redis-сервер, що доступний на порті 6565.

Контейнер PostgreSQL із внутрішнім портом 5432 та зовнішнім портом 5432. Цей контейнер являє собою сервер PostgreSQL, який буде доступний на порті 5432.

Для початку розробки розширення для КС NetBox необхідно завантажити відповідний програмний код із офіційного GitHub репозиторія та провести першочергове налаштування відповідних параметрів баз даних. Створити профіль користувача-адміністратора в рамках NetBox та провести оновлення системи. Усі виконувані команди зібрані в лістинг 2.2.

Лістинг 2.2 – Команди налаштування NetBox

```
sudo mkdir -p /opt/netbox/
cd /opt/netbox/
sudo git clone -b master --depth 1 https://github.com/netbox-
community/netbox.git .
sudo adduser --system --group netbox
sudo chown --recursive netbox /opt/netbox/netbox/media/
sudo chown --recursive netbox /opt/netbox/netbox/reports/
sudo chown --recursive netbox /opt/netbox/netbox/scripts/
cd /opt/netbox/netbox/netbox/
sudo cp configuration_example.py configuration.py
python3 ../generate_secret_key.py
sudo /opt/netbox/upgrade.sh
source /opt/netbox/venv/bin/activate
cd /opt/netbox/netbox
python3 manage.py createsuperuser
```

## 2.2 Проєктування архітектури розширення для КС

Django, високорівневий веб-фреймворк на Python, використовує модифіковану версію архітектурного шаблону Model-View-Controller (MVC),

відомого як шаблон Model-View-Template (MVT). Цей шаблон проектування є чітко визначеним способом розділення проблем у додатку, що полегшує його розробку та підтримку.

Model (з англ. модель) - є єдиним, остаточним джерелом інформації про дані. Вона містить основні поля та поведінку даних, які зберігаються в базу даних. Моделі Django є підкласом django.db.models.Model і використовуються для прозорого створення, отримання, оновлення та видалення записів у базі даних. Моделі також використовуються для визначення схеми даних додатку, оскільки вони визначають структуру таблиць бази даних та зв’язки між ними.

View (з англ. представлення) - в Django обробляють логіку, що відповідає за обробку запиту користувача і за повернення відповіді користувачеві. Представлення в Django створюються шляхом написання функцій або класів Python. Кожна функція представлення приймає веб-запит і повертає веб-відповідь. Цією відповіддю може бути HTML-вміст документа, перенаправлення, XML-документ, зображення або практично будь-який інший тип файлу, або це може бути помилка HTTP, наприклад, 404 або 500 [8]. Представлення також може делегувати деяку обробку шаблону або іншому поданню. Завдяки представленням також надається доступ до даних користувачеві - реалізується контроль доступу.

Template (з англ. шаблон) - в Django це частина, яка відповідає за представлення інформації користувачеві. Django використовує свій механізм шаблонів для визначення HTML або інших форматів виводу. Шаблон описує, як дані, отримані з представлення і повинні бути перетворені у вихідні формати. Шаблони часто використовуються для створення HTML, але шаблони Django можуть генерувати будь-який текстовий формат. В шаблонах Django використовується просунутий шаблонізатор Jinja - програмний засіб для полегшення створення шаблонів, що надає програмістам більшу гнучкість під час написання програмного забезпечення.

Користувач взаємодіє з представленням, яке за необхідності робить запити до моделі. Модель повертає дані у шаблон, який потім опрацьовує та відображає

шаблон з отриманими даними. Цей патерн дозволяє логічно розмежувати компоненти, що приводить до більшої гнучкості та зручності коду. Розробники можуть змінювати структуру бази даних (модель) або дизайн інтерфейсу (шаблони) незалежно від логіки програми (представлення), і навпаки.

NetBox має модульну архітектуру, що наслідує патерн Model-View-Template (MTV) у Django (див. рис. 2.1), який розділяє модель даних, інтерфейс користувача та бізнес-логіку програми. Модель даних NetBox реалізована за допомогою Django ORM, яка забезпечує високорівневу абстракцію для доступу та управління даними в реляційній базі даних. Базується на реляційній базі даних PostgreSQL для зберігання даних і використовує Redis для кешування і розробки черг задач. Інтерфейс користувача реалізовано за допомогою системи шаблонів Django, яка дозволяє створювати динамічні та адаптивні веб-сторінки. Бізнес-логіка реалізована на мові Python і відповідає за обробку користувацького вводу, виконання запитів до даних та оновлення бази даних. Всі ці компоненти можна розгорнути як окремі служби, що дозволяє гнучкість управління масштабуванням.

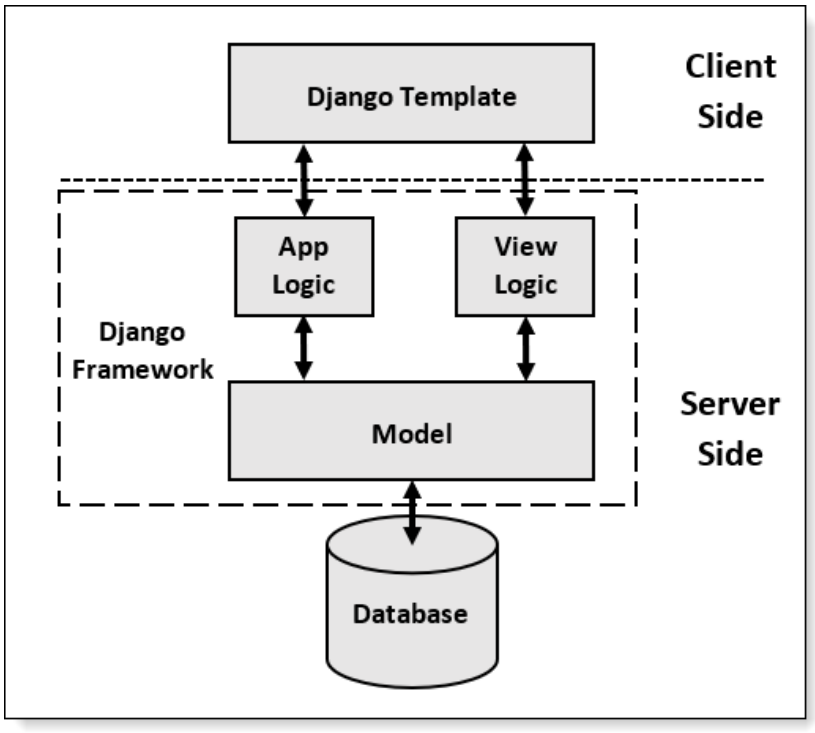


Рисунок 2.1 – Схема архітектури ПЗ розробленої на Django

NetBox поділяється на декілька основних модулів:

– DCIM – модуль для управління даними центру обробки даних, включаючи пристрої, розташування, роз’єми, волокна та інше.

– IPAM – модуль для управління IP-адресами, включаючи діапазони IP, підмережі, VLAN, VRF, ASN тощо.

– Віртуалізація – модуль, що відслідковує віртуальні машини та їх інтерфейси.

– Circuits – модуль для керування мережевими картами і провайдерами.

– Power – модуль для керування джерелами живлення та з’єднаннями.

– Secrets – модуль для безпечного зберігання чутливих даних, таких як паролі, ключі шифрування, тощо.

Відповідно розробка розширення для КС NetBox має наслідувати дану архітектуру виключаючи втручання в основний код системи задля збереження зворотної сумісності та поширюваності створюваного програмного рішення. Розширення встановлюються суміжно із Netbox та є окремими компонентами, завдяки такому підходу, можна легко підключати та відключати потрібні розширення без зайвої збірки та простоювання усього програмного рішення без дії у період оновлення, та в разі виявлення критичних неполадок дозволяє з легкістю ізолювати компоненту.

## 2.3 Визначення сутностей та атрибутів КС

Розроблюване розширення для КС моделювання і документування КС виконує функцію управління списками заблокованих IP-адрес та їх підмереж відповідного підрядника, для забезпечення глобального огляду усіх можливих заблокованих мереж та їх винятків (IP-адрес, що були розблоковані в даній підмережі).

Заблокована IP-адреса являє собою відповідну сутність, яка у рамках фреймворку Django називається моделлю. Дана модель має містити атрибут власне IP-адреси з мережевою маскою в форматі CIDR, атрибут з описом.

Розблокована IP-адреса являє собою відповідну сутність, яка містить атрибут IP-адреси з мережевою маскою /32 (для позначення одиничної IP-адреси в рамках мережі), атрибут з описом цієї IP-адреси та атрибут з ідентифікатором заблокованої мережі для створення з'єднання між моделями типу «один-до-багатьох» (що буде відповідати приналежності багатьох розблокованих IP-адрес до однієї заблокованої підмережі).

У кожної сутності присутнє також ще одне з'єднання «один-до-багатьох» у вигляді атрибута локація (поле з описом приналежності даної підмережі IP-адрес до певного дата-центру чи місця в дата-центрі), даний атрибут посилається на уже реалізовану вбудовану сутність NetBox під назвою Sites.

Заблокована підмережа IP-адрес відображає логіку вміщення в собі набору можливих IP-адрес, які позначені як заблоковані оскільки знаходяться в рамках даної підмережі. Проте сутність розблокованих IP-адрес представляє вийняток із цього правила та описує одиничну адресу, яка розблокована в рамках даної підмережі. Опис сутностей та їх типи представлено у табл. 2.1-2.2.

Таблиця 2.1 – «DissociatedIP» (заблокована IP-адреса)

| Атрибут | Тип атрибута | Примітка |
|---|---|---|
| address | IP-адреса, cidr | IP-адреса мережі |
| description | текст, varchar(300) | Опис мережі |
| site | зовнішній ключ (число), Foreign key int | Локація IP-адреси мережі |

Таблиця 2.2 – «AssociatedIP» (розблокована IP-адреса)

| Атрибут | Тип атрибута | Примітка |
|---|---|---|
| address | IP-адреса, inet | IP-адреса в мережі |
| description | текст, varchar(300) | Опис мережі |
| dissociated | зовнішній ключ (число), Foreign key int | Заблокована IP-адреса |
| site | зовнішній ключ (число), Foreign key int | Локація IP-адреси мережі |

Для позначення адреси підмережі в сутності DissociatedIP використовується тип даних cidr у PostgreSQL [17]. Даний тип даних, який використовується для зберігання та маніпулювання IP-адресами та підмережами. Він записується згідно CIDR, тому обов'язково складається з самої IP-адреси та маски підмережі у форматі «x.x.x.x/y» - для IPv4, де «x.x.x.x» це чотири октети IP-адреси та «y» позначає префікс підмережі, який вказує на кількість мережевих бітів, та «xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/y» - для IPv6, де «xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx» позначає IPv6 адресу та «y» позначає префікс підмережі, який вказує на кількість мережевих бітів. У PostgreSQL cidr тип даних дозволяє виконувати базові функції з IP-адресами та мережами такі, як перевірка наявності даної адреси в поточній мережі, об'єднання мереж та дозволяє булеві операції з IP-адресами.

Для позначення адреси підмережі в сутності AssociatedIP використовується тип даних inet у PostgreSQL [17]. Цей тип даних відрізняється від cidr хоч і має схожий функціонал. Він надає можливість зберігання IP-адрес в форматі CIDR із маскою підмережі так і зберігати одиничну IP-адресу без маски підмережі у форматі «x.x.x.x» - для IPv4 адрес та «xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx» - для IPv6 адрес. У PostgreSQL inet тип даних дозволяє використання мережевих префіксів проте більше підходить для зберігання однієї IP-адреси. У контексті даної проєктованої КС відповідний тип даних буде зберігати IP-адреси із мережевим префіксом «/32», який позначає єдину адресу в мережі, власне потрібну IP-адресу.

## 2.4 Визначення та опис зав'язків між сутностями КС

Зв'язки між сутностями лежать в основі побудови будь-якої великої системи адже без їх використання неможливо уявити повноцінне існування КС в цілому. У реляційній базі даних зв'язки - це зв'язок між двома або більше таблицями. З'єднання використовуються для представлення зав'язків між сутностями в базі

даних. Ці зв’язки забезпечують ефективне зберігання та пошук складних структур даних. Вони реалізуються за допомогою первинних і зовнішніх ключів, які слугують унікальними ідентифікаторами і перехресними посиланнями для даних у різних таблицях.

Первинний ключ - це унікальний ідентифікатор запису в таблиці, який гарантує, що кожен запис може бути чітко ідентифікований зазвичай використовується число, яке щоразу інкрементується при поступленні нового запису в таблицю. Зовнішній ключ - це поле або колекція полів в одній таблиці, які однозначно ідентифікують рядок іншої таблиці. Зовнішній ключ в одній таблиці вказує на первинний ключ в іншій таблиці, створюючи зв'язок або відношення між ними.

Типи зв’язків або відношень у реляційній базі даних зазвичай поділяються на три типи: один-до-одного, один-до-багатьох і багато-до-багатьох. Зв'язок «один-до-одного» існує, коли один запис в одній таблиці пов'язаний з одним записом в іншій таблиці. Наприклад, кожен клієнт у базі даних має унікальний ідентифікатор клієнта.

Зв’язок «один-до-багатьох» існує, коли один запис в одній таблиці пов'язаний з декількома записами в іншій таблиці. Це найпоширеніший тип зв'язку.

Зв’язок «багато-до-багатьох» існує, коли кілька записів в одній таблиці пов’язані з кількома записами в іншій таблиці. Цей тип зв'язку реалізується за допомогою з’єднання або зв’язуючої таблиці.

Реалізація таких зв’язків у реляційних базах даних є критично важливою з кількох причин. По-перше, це сприяє цілісності даних, забезпечуючи узгодженість і точність даних. По-друге, це дозволяє виконувати складні запити та маніпулювати даними, надаючи можливість переглядати та аналізувати дані з різних точок зору. Також, він підтримує принципи нормалізації - процесу організації даних для мінімізації надмірності та покращення їхньої цілісності. Розбиваючи великі таблиці на менші, менш надлишкові таблиці та визначаючи

зв’язки між ними, нормалізація допомагає зменшити надлишковість даних та уникнути потенційних аномалій даних. Таким чином, ці зв’язки або відносини в реляційних базах даних є не тільки фундаментальними, але й корисними для забезпечення надійного, ефективного і точного управління даними.

Схематичне подання зв’язків представлене на рис. 2.3.


Рисунок 2.3 – Структура зв’язків таблиць проєктованих сутностей

# РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

## 3.1 Реалізація програмної архітектури КС

Реалізація розробки комп'ютеризованої системи моделювання і документування мережевих ресурсів дата-центру тісно пов'язана із розумінням логіки роботи кінцевого користувача із створюваним продуктом. Для даного рішення підходить діаграма послідовностей, яка чітко може описати принцип роботи Netbox, зображена на рис. 3.1.

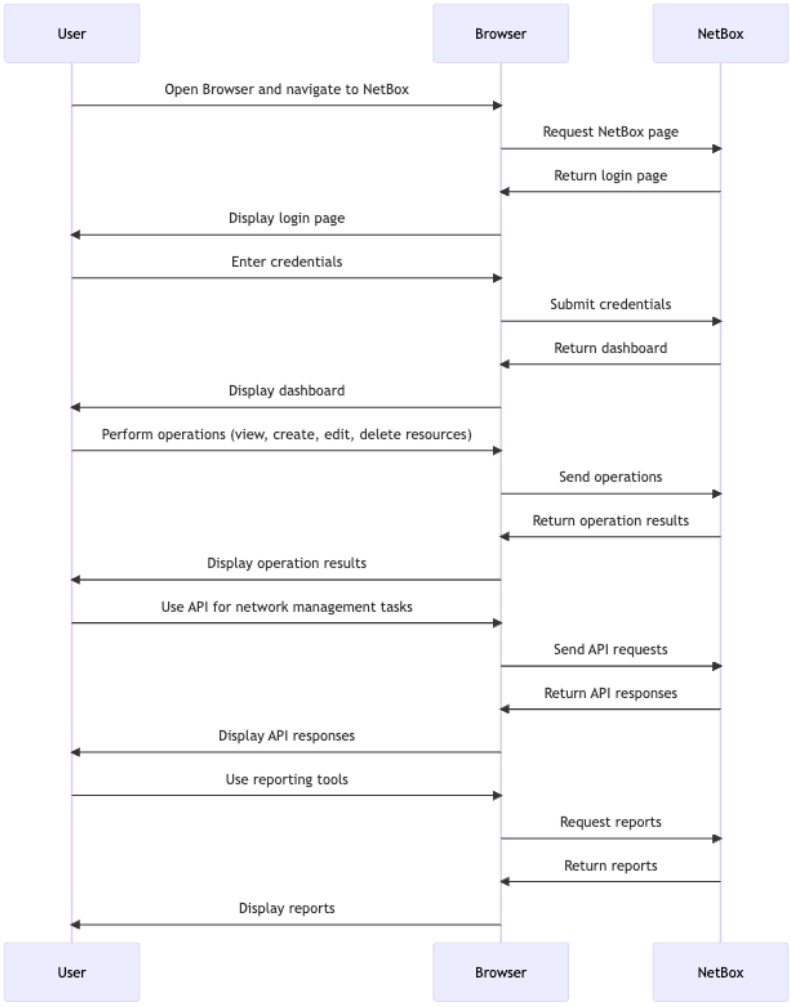


Рисунок 3.1 – Діаграма послідовностей користування Netbox

З вище представленої діаграми (див. рисунок 3.1) можна виокремити кілька основних варіантів користування користувачем КС NetBox, так і власне проєктованого розширення:

– вхід в систему (авторизація користувача в системі);

– перегляд наданої інформації;

– виконання операцій із сутностями (створення, зміна , видалення , пошук , перевірка сутності, що надається функціоналом розроблюваного розширення);

– створення та перегляд звітів і аналітик.

Систематизоване і узагальнення представлення зображено на блок-схемі рис. 3.2.

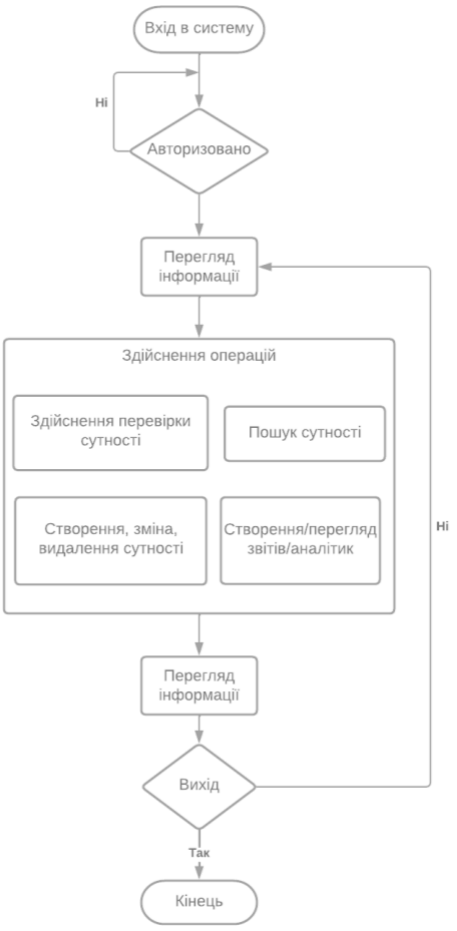


Рисунок 3.2 – Блок-схема узагальненого алгоритму роботи із NetBox

Для створення відповідного розширення, потрібно створити структуру модуля в фреймворку Django, яка буде доповнювати наявні, структура наявних модулів представлена на рис. 3.3.

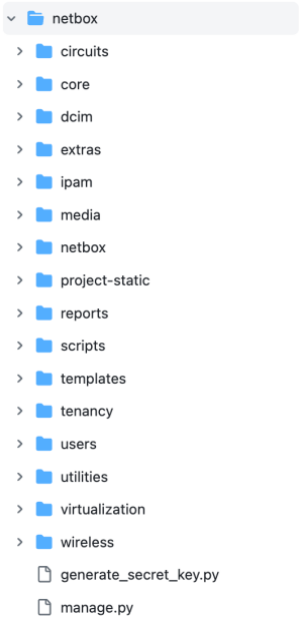


Рисунок 3.3 – Представлення структури NetBox

В середині директорії netbox створюється відповідна структура:

– __init__.py – файл необхідний для того, щоб Python розпізнав директорію як пакет Python.

– models.py – файл, в якому визначаються моделі бази даних, які використовуються плагіном. Це включає таблиці і відношення між ними.

– templates/ – директорія, яка містить HTML шаблони, які використовуються плагіном.

– navigation.py – файл який відповідає за навігацію в рамках NetBox, додає ще один рівень абстракції.

– views.py – файл, що визначає представлення плагіна (функції, які приймають веб-запит і повертають веб-відповідь).

– urls.py – файл, в якому визначаються URL-шляхи для плагіна.

– forms.py – файл, в якому визначаються форми, які використовуються у представленнях плагіна.

– api/ – директорія для файлів, які визначають API ендпоїнти плагіна.

– migrations/ – директорія для файлів, які використовуються при міграціях відповідних моделей плагіна.

– tables.py – файл, в якому знаходяться усі налаштування для відображення даних у вигляді таблиць всередині NetBox.

– filtersets.py – файл, в якому містяться правила фільтрування моделей для розширеного пошуку.

– utils.py – файл із власними допоміжними функціями.

– setup.py – файл, в якому описуються правила збірки даного розширення, як модуля Python вбудованими засобами.

Структура директорії розширення для NetBox подана на рис. 3.4.

Для створення моделей, а з них таблиць в базі даних використовується Django ORM (Object Relation Mapping, з англ. об’єктно-реляційне відображення) - це частина фреймворку Django, яка спрощує взаємодію з базою даних. Вона перетворює моделі в Python у SQL-запити і навпаки, чим дозволяє працювати з базами даних в об’єктно-орієнтованому стилі, лістинг код для створення відповідних представлено в лістингах 3.1-3.2.

Лістинг 3.1 – Створення моделі DissociatedIP

```
class DissociatedIP(NetBoxModel):
    address = IPNetworkField(
        blank=False,
        help_text='IPv4 or IPv6 address (with mask)'
    )
    description = models.CharField(
        max_length=300,
        blank=True
    )
    site = models.ForeignKey(
        to=Site,
        on_delete=models.SET_NULL,
        related_name='dissociatedips',
        blank=True,
```

```
        null=True
    )
```

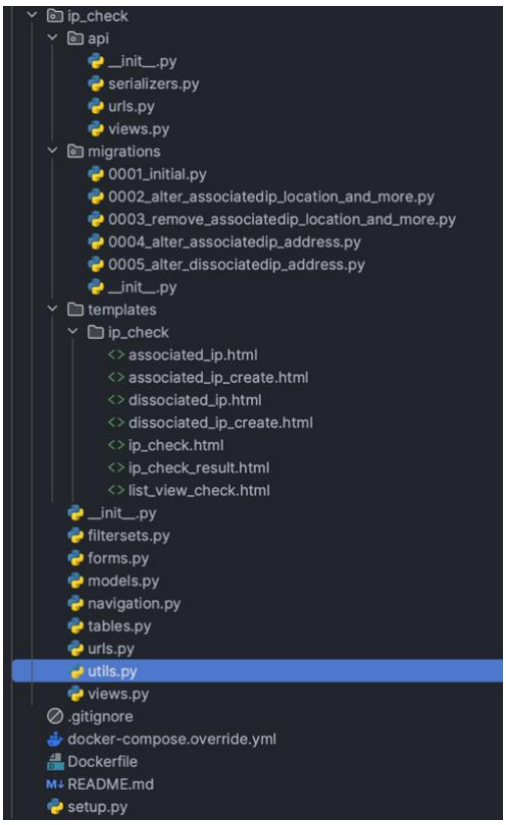


Рисунок 3.4 – Структура розроблюваного розширення КС

Лістинг 3.2 – Створення моделі AssociatedIP

```
class AssociatedIP(NetBoxModel):
    address = IPAddressField(
        blank=False,
        help_text='IPv4 or IPv6 address (with mask)'
    )
    description = models.CharField(
        max_length=300,
        blank=True
    )
    site = models.ForeignKey(
        to=Site,
        on_delete=models.SET_NULL,
        related_name='associatedips',
        blank=True,
        null=True
    )
    dissociated = models.ForeignKey(
        to=DissociatedIP,
        on_delete=models.SET_NULL,
        related_name='associatedips',
```

```
            blank=True,
            null=True
        )
```

Для створення відповідних файлів міграцій та таблиць потрібно виконати команди вбудованого в фреймворк Django засобу manage.py (приклад наведено в лістингу 3.3).

Лістинг 3.3 – Створення таблиць та файлів міграцій

```
./manage.py makemigrations ip_check
./manage.py migrate ip_check
```

Для створення базових представлень використано вбудовану реалізацію в NetBox для зручності написання програмного коду та його уніфікації, відповідний представлено у лістингу 3.4.

Лістинг 3.4 – Пердставлення для переголяду списків сутності DissociatedIP

```
class DissociatedIPListView(generic.ObjectListView):
    template_name = "ip_check/list_view_check.html"
    queryset = models.DissociatedIP.objects.all()
    table = tables.DissociatedIPTable
    filterset = filtersets.DissociatedIPFilterSet
    filterset_form = forms.DissociatedIPFilterForm
    def get_extra_context(self, request):
       return
{'check_link':reverse("plugins:ip_check:ip_check}
```

Для відповідного відображення представлень їх необхідно зареєструвати в системі маршрутизації запитів до сервера за допомогою змінної urlpatterns (приклад представлено у лістингу 3.5). В дану змінну входить масив усіх шляхів за якими буде переходити користувач для перегляду відповідної інформації, а йому буде видаватись відповідне представлення, яке буде відображати відповідний шаблон.

Лістинг 3.5 – Неповний набір маршрутизації представлень

```
     urlpatterns = (
    # DissociatedIP
    path("dissociatedip/", views.DissociatedIPListView.as_view(),
name="dissociatedip_list"),
    path("dissociatedip/add/",
views.DissociatedIPCreateView.as_view(), name="dissociatedip_add"),
    path("dissociatedip/import/",
views.DissociatedIPBulkImportView.as_view(),
name="dissociatedip_import"),
    path("dissociatedip/edit/",
views.DissociatedIPBulkEditView.as_view(),
name="dissociatedip_bulk_edit"),
    path("dissociatedip/delete/",
views.DissociatedIPBulkDeleteView.as_view(),
name="dissociatedip_bulk_delete"),
    path("dissociatedip/<int:pk>/",
views.DissociatedIPView.as_view(), name="dissociatedip"),
    path("dissociatedip/<int:pk>/edit/",
views.DissociatedIPEditView.as_view(), name="dissociatedip_edit"),
    path("dissociatedip/<int:pk>/delete/",
views.DissociatedIPDeleteView.as_view(),
name="dissociatedip_delete"),
    path("dissociatedip/<int:pk>/changelog/",
ObjectChangeLogView.as_view(), name="dissociatedip_changelog",
         kwargs={"model": models.DissociatedIP}),
     ...
     )
```

Основною задачею в розробці було створення форми та ендоїнта для перевірки вказаної користувачем IP-адреси на наявність в одному з двох типів списків. Для цього створено відповідну форму, яка посилає POST запит із вказаним переліком адрес від користувача. У формі наявна базова валідація входжуваних даних та присутня логіка пошуку відповідної адреси на наявність спочатку в списку блокованих підмереж IP-адрес, а потім за наявності відповідної підмережі у списку блокованих адрес, пошук в списку не блокованих. При наявності вказаної адреси в списках повертається тип списку, мережа до якої відноситься та повідомлення зручне для користувача (приклад виконання представлений на лістингу 3.6).

Лістинг 3.6 – Форма пошуку IP-адреси

```
    def post(self, request):
        raw_data = request.POST.get("data", "").strip()
        result = []
        ips = re.split(r"[^0-9.]", raw_data)
        for ip in ips:
            if not is_ip(ip):
                continue
            elif ip == '0.0.0.0':
                result.append({"ip": ip,
                               "message": "Wrong IP address
given!"})
                continue
            ip_obj, list_type, network = None, None, None
            message = f"IP address {ip} not found in any of the
lists."
            if in_dissociated_list :=
models.DissociatedIP.objects.filter(
    Q(address__net_contains_or_equals=netaddr.IPNetwork(ip).ip)):
                list_type = "Dissociated"
                network = in_dissociated_list[0]
                if in_associated_list :=
models.AssociatedIP.objects.filter(Q(address=ip)):
                    ip_obj = in_associated_list[0]
                    list_type = "Associated"
                message = f"IP address {ip} present in {list_type}
IP list."
            result.append({
                "ip": ip,
                "ip_obj": ip_obj,
                "network": network,
                "list_type": list_type,
                "message": message
            })
```

Для створення відповідного ендпоїнта використовується аналогічна логіка, із незначними змінами в класах та із зміненим поверненням значень користувачеві.

## 3.2 Реалізація користувацького інтерфейсу

Реалізація користувацького інтерфейсу представлено у стандартному вигляді веб-сторінки з бічною панеллю для керування та навігації, та основним тілом сторінки в якому елементи взаємодії та наповнення змінюються відповідно

до вибраної користувачем сутності. Приклад основної сторінки NetBox представлено на рис. 3.5.

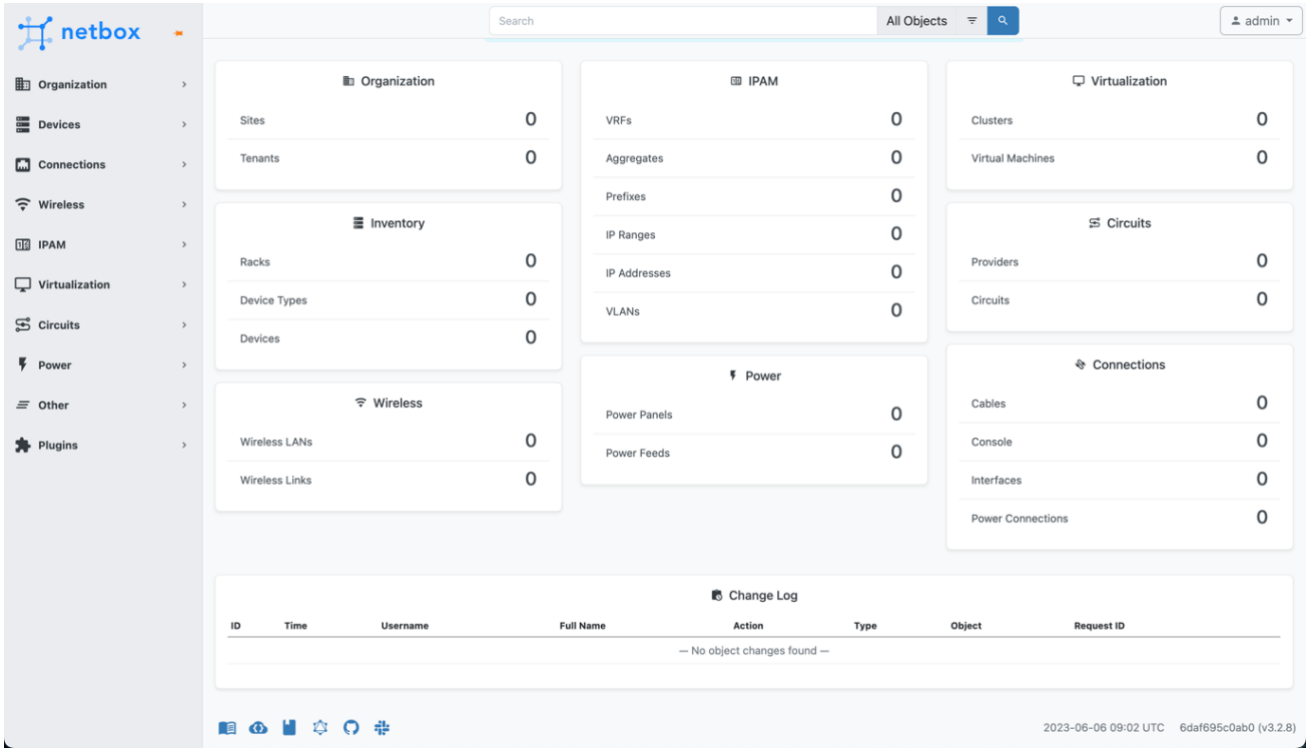


Рисунок 3.5 – Головна сторінка NetBox

У розширення IPCheck присутні наступні основні користувацькі інтерфейси:

– відображення одного запису сутності;
– відображення усіх записів сутності (таблиця);
– форма створення/зміни/пошук сутності;
– відображення результатів пошуку.

Кожен з цих інтерфейсів наслідує базову структуру вбудованих шаблонів NetBox, що дозволяє взяти максимум від переваги шаблонізатора Jinja. Приклад вигляду відповідних інтерфейсів зображено на рис. 3.6-3.9.

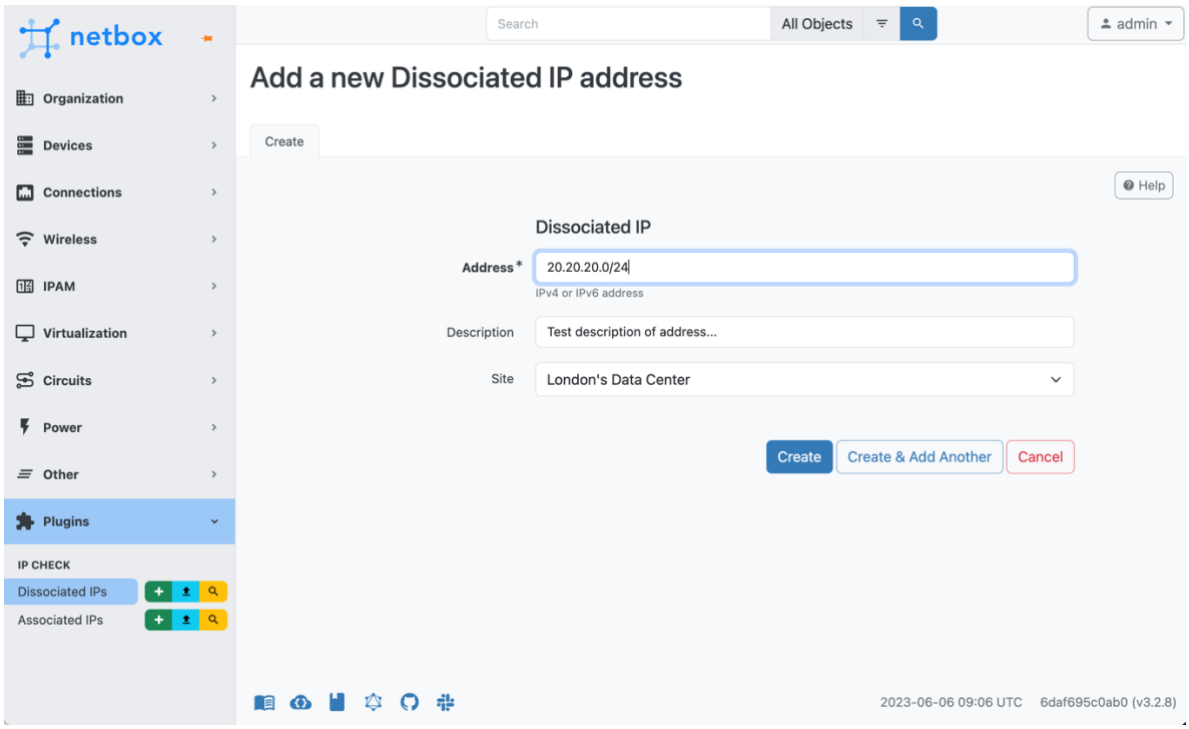


Рисунок 3.6 – Форма створення блокованого адресу

Лістинг 3.7 – Код шаблону створення блокованого адресу

```
{% extends 'generic/object_edit.html' %}
{% load form_helpers %}
{% block form %}
    <div class="field-group my-5">
        <div class="row mb-2">
            <h5 class="offset-sm-3">Dissociated IP</h5>
        </div>
        {% render_field form.address %}
        {% render_field form.description %}
        {% render_field form.site %}
    </div>
{% endblock %}
```

У даному випадку шаблон розширює вбудований в NetBox шаблон-заготовку generic/object_edit.html, і виводить відповідні поля для заповнення (див. рис. 3.6). Кожен шаблон змінюється та проєктується схожим принципом із зміною під відповідні поля та з наслідуванням потрібних шаблонів. Вбудованим шаблонізатором не обмежується кількість наслідуваних елементів що дозволяє створювати безліч рівнів абстракції.

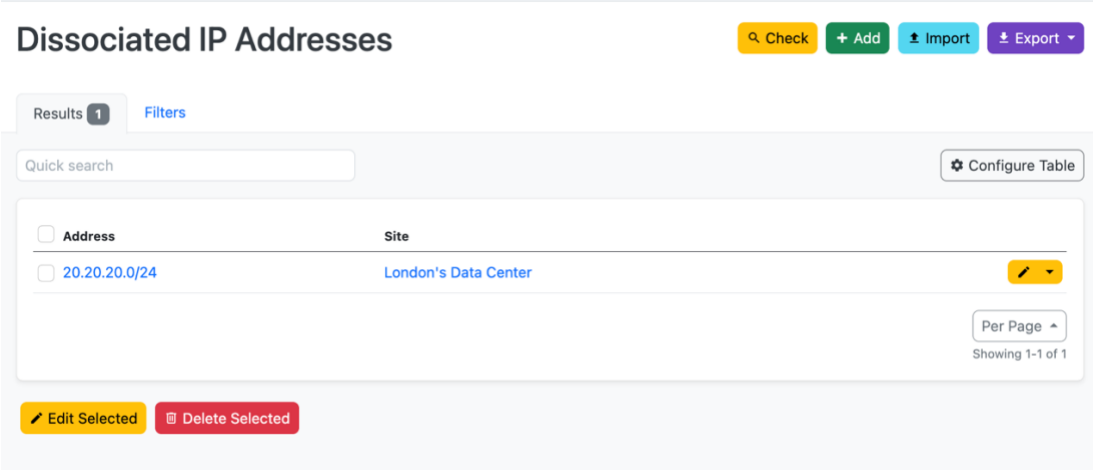


Рисунок 3.7 – Приклад табличного відображення блокованих адресів

У випадку табличного відображення доступні функції швидкої зміни (редагування, видалення) одного чи кількох записів одночасно (див. рис. 3.7). Також вигляд може коригуватись користувачем за потреби, також доступний імпорт та експорт даних, пошук та перевірка адрес.

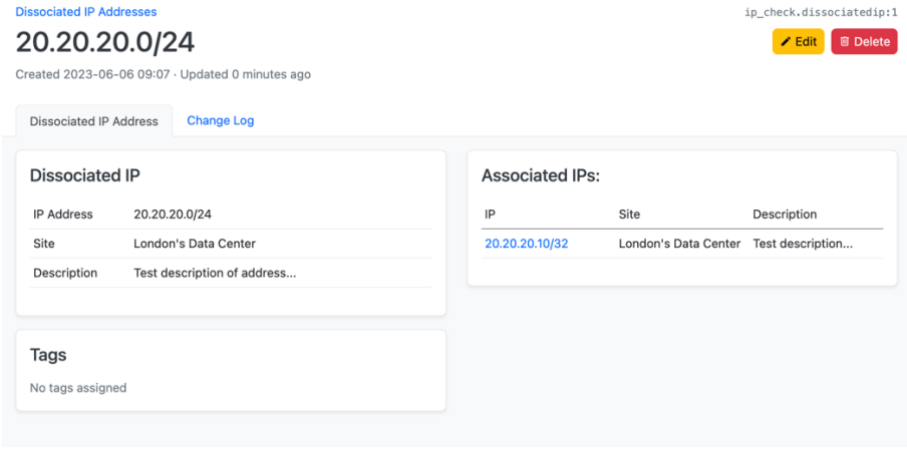


Рисунок 3.8 – Приклад одиничного запису блокованого адресу

У випадку із одиничним представленням, базовий шаблон розширено додатковим функціоналом у вигляді панелі зліва, яка відображає список адрес, які не є заблокованими для даної підмережі (див. рис. 3.8). Приклад коду для даної панелі наведено в лістингу 3.8.

Лістинг 3.8 – Код для доповнення шаблону адрес панеллю зліва

```
<div class="col col-12 col-xl-6"><div class="card">
<h5 class="card-header">Associated IPs:</h5>
<div class="card-body">
<table class="table table-hover attr-table">
<thead class="thead-light">
<tr>
<th scope="col">IP</th>
<th scope="col">Site</th>
<th scope="row">Description</th>
</tr></thead><tbody>
{% for associated_ip in associated %}
<tr><td>
<a href="{{ associated_ip.get_absolute_url }}">{{
associated_ip.address }}</a>
</td><td>{{ associated_ip.site|placeholder }}</td>
<td>{{ associated_ip.description|placeholder }}</td>
</tr>{% endfor %}
</tbody></table></div></div></div>
```

Для відображення результатів перевірки адреси на факт наявності в списках блокованих або не блокованих адрес створено окремий шаблон, який дозволяє виводити перелік адрес у зручних картках із усією потрібною інформацією щодо них, зображено на рис. 3.9.

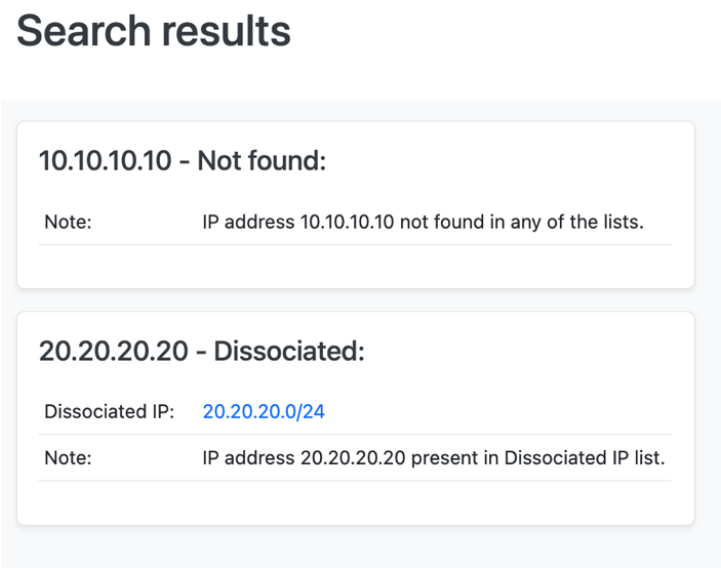


Рисунок 3.9 – Приклад результатів перевірки

## 3.3 Тестування програмного забезпечення

Для тестування даного програмного забезпечення використано ручне тестування (передбачає перевірку готового програмного забезпечення вручну набираючи та обираючи відповідні вхідні дані) та тестування за допомогою програми Postman, яка дозволяє протестувати автоматично так і в напів-ручному форматі ендпоїнти розробленої API. В тестуванні були залучені, як справжні вхідні дані так і генеровані випадкові дані, а також дані з помилками для перевірки коректності валідації вхідних даних. Для перевірки також можна використовувати автоматично згенеровану документацію API КС NetBox таку як Swagger (див. рис. 3.10).

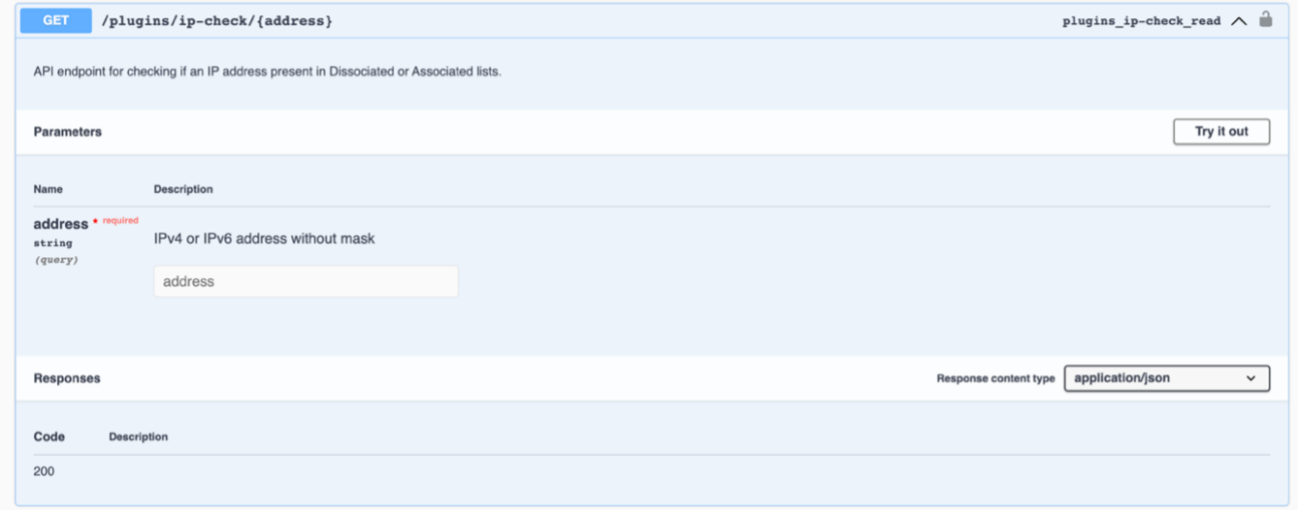


Рисунок 3.10 – Приклад ендпоїнта для перевірки одиничної адреси

У випадку із тестуванням за допомогою Postman (див. рис. 3.11) використовується ендпоїнт для перевірки кількох IP-адрес. У даному випадку відбувається POST запит до сервера із тілом в якому передаються відповідні дані у форматі представленому у лістингу 3.9.

Лістинг 3.9 – Приклад запиту до сервера

```
curl --location 'http://localhost:8000/api/plugins/ip-check/bulk-check/' \
```

```
--header 'Content-Type: application/json' \
--data '{
    "ip_addresses": [
        "127.0.0.1"
    ]
}'
```

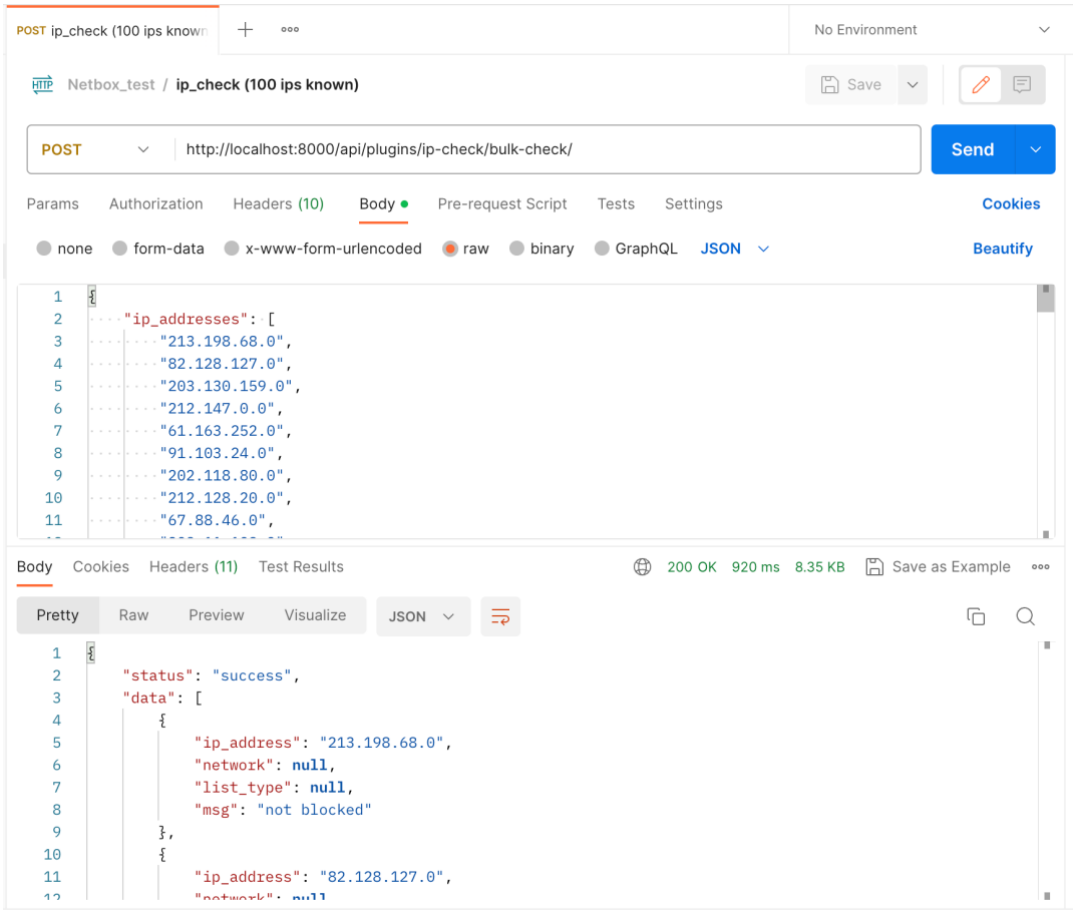


Рисунок 3.11 – Приклад використання програми для тестування

## 3.4 Розгортання програмного забезпечення КС

Для розгортання даного програмного рішення використовується платформа Docker, що полегшує розгортання даної КС на сервері не залежно від операційної системи даного сервера. Єдиним важливим критерієм є наявність встановленого Docker та docker-compose. Для прикладу обрано сервер під управлінням Ubuntu Server 20.04.

Перш за все, встановлення цього плагіна вимагає клонування потрібного репозиторію netbox-docker за допомогою команди git clone. Коли репозиторій

було склоновано, наступний крок полягає в тому, щоб додати певні налаштування до конфігураційного файлу netbox-docker. Це необхідно для того, щоб активувати плагін в NetBox і дозволити йому коректно працювати. Наступним кроком буде внесення зміни в конфігураційний файл /env/netbox.env відповідно до специфічних вимог користувача. Цей файл містить різні параметри, які контролюють, як саме працює NetBox. Модифікація цього файлу може включати зміну параметрів мережі, налаштування бази даних або визначення інших параметрів, які специфічні для вашого середовища. Далі потрібно скопіювати каталог ip_check/, Dockerfile-Plugins, docker-compose.override.yml та setup.py до новоствореного каталогу netbox-docker. Це забезпечує, що всі необхідні для роботи плагіна файли знаходяться в правильному місці. Після виконання цих команд, IP Check NetBox Plugin буде встановлено і КС готова до використання. Командами для розгортання КС представлено у лістингу 3.10.

Лістинг 3.10 – Перелік команд для розгортання КС

```
    git clone -b release https://github.com/netbox-
community/netbox-docker.git
    echo "PLUGINS=["ip_check"]" >> netbox-
docker/configuration/configuration.py
    cp -R ./ip_check/ ./netbox-docker/ip_check/
    cp ./Dockerfile-Plugins ./docker-compose.override.yml
./setup.py ./netbox-docker
    cd netbox-docker
    docker-compose up -d
```

# РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

## 4.1 Естетичне оформлення робочого місця оператора ПК

Естетичне оформлення робочого місця оператора ПК, покращує настрій та продуктивність працівника, створює комфортну атмосферу і підвищує задоволення від роботи, що позитивно впливає на його працездатність.

Важливим аспектом є оптимальна організація робочого простору. Усі необхідні та використовувані об’єкти розташовуються таким способом, щоб вони були в межах досяжності. Також враховується ергономіка робочого місця, забезпечується комфортне меблювання та аксесуари, які враховують принципи ергономіки. Регульовані стільці, підлокітники, клавіатура та мишка, підставки для монітора – все це сприяє збереженню здоров’я та зниженню напруження під час роботи. Оптимальне робоче місце зображено на рис. 4.1.

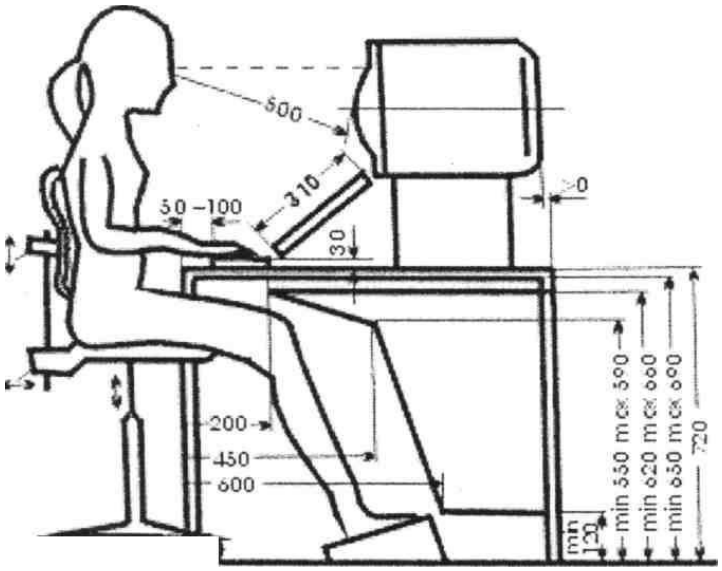


Рисунок 4.1 – Правильно оформлене робоче місце

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | *КС КРБ 123.052.00.00 ПЗ* | | |
| *Змн.* | *Арк.* | *№ докум.* | *Підпис* | *Дата* | | | |
| *Розроб.* | | *Папка О.В.* | | | *Безпека життєдіяльності, основи охорони праці* | *Літ.* | *Арк.* 42 / *Акрушів* 5 |
| *Перевір.* | | *Тиш Є. В.* | | | | | |
| *Консульт.* | | *Пилипець М. І.* | | | | *ТНТУ, каф. КС, гр. СІ-41* | |
| *Н. контр.* | | | | | | | |
| *Затверд.* | | *Осухівська Г. М.* | | | | | |

Згідно ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги, розміри стола залежать від розміру екрана, рекомендовані розміри: довжина 160 см, ширина 90 см, загальна площа 1,44 м$^2$. Висота стола регулюється відповідно до антропометричних даних (68-84 см). Екран розташовується нижче рівня очей з кутом зору, що забезпечує оптимальне розміщення символів (кут зору близько 0,5). Відстань від очей до екрану знаходиться в межах 40-90 см, залежно від висоти символів. Документи читаються з підставки з регульованою висотою та нахилом для зменшення відблисків. Відповідно до антропометричних характеристик працівника використовується підставка для ніг розмірами 40x30x15 см з кутом нахилу 30°.

Освітлення – це важливий аспект при роботі за ПК. Забезпечується достатнє природне або штучне освітлення, що сприяє покращенню настрою робітника, та убезпечує від травмувань та погіршення зору. Освітленість робочих місць знаходиться в межах 300-500 люкс для зони документів та клавіатури. Відношення яскравості поверхонь не перевищує 3:1, а між робочою поверхнею столу та навколишніми поверхнями 10:1, згідно ДСТУ EN 12464-1:2016 Світло та освітлення. Освітлення робочих місць. Частина 1. Внутрішні робочі місця (EN 12464-1:2011, IDT).

Чистота і порядок також покращують робочий процес, робоче місце регулярно прибирається, використовуються організатори, шафки або ящики для зберігання дрібниць та документів, це зобить робоче місце більш привабливим та зручним для роботи. Усе обладнання використовується акуратно та електричні прилади зберігаються подалі від води.

Декоративні елементи також привносять в загальний робочий процес покращення настрою робітника. Елементи, які відповідають смаку та інтересам робітника, особисті речі, фотографії, картини, рослини або інші предмети створюють приємну атмосферу та персоналізують робочий простір.

Для покращення психоемоційного стану за робочим місцем правильно підбирається колірна палітра. Використовуються приємні та заспокійливі кольори

на стінах або елементах оформлення, які впливають на настрій та концентрацію. Кольори блакитний або зелений вважаються такими, що сприяють спокою та зосередженості.

Враховуються індивідуальні вподобання та особистість працівника, адже кожна людина має свої унікальні вподобання та потреби.

## 4.2 Особливості заходів електробезпеки на підприємствах

Заходи електробезпеки на підприємствах під час використання персональних комп'ютерів (ПК) з використанням комп'ютеризованої системи моделювання і документування мережевої інфраструктури дата-центру, мають на меті забезпечити безпечні умови роботи з електричними пристроями та уникнути можливих небезпек, пов'язаних з електростатичним розрядом, перевантаженням мережі, пожежами та іншими проблемами.

Всі комп'ютери та електронні пристрої правильно заземлюються. Заземлення відіграє важливу роль у захисті від ураження струмом та уникненні статичного розряду. Це допомагає направити струм у разі виникнення витоку, забезпечуючи безпечний шлях для електричного струму, що перешкоджає ураженню людини.

Для забезпечення стабільного живлення ПК підключаються до стабільного та надійного джерела живлення. Стабілізатори напруги або безперебійні джерела живлення (UPS) використовуються для регулювання та забезпечення стабільного живлення пристроїв. Вони також надають додатковий час для збереження даних та безпечного вимкнення системи у випадку відключення живлення.

Електричні кабелі та розетки, які використовуються для підключення ПК, відповідають вимогам електробезпеки, знаходяться в цілості та не мають пошкоджень, зокрема тріщин або вигинів. Кабелі розташовуються таким чином, щоб їх не було легко перетнути або пошкодити. Розетки, до яких підключаються ПК, заземлюються та встановлюються на безпечній відстані від джерел вологи, щоб уникнути короткого замикання або пожеж.

Комп’ютери розташовуються у вентильованому приміщенні, де забезпечується нормальна циркуляція повітря. Висока температура може призвести до перегріву компонентів, що може спричинити несправності пристроїв або навіть пожежу. Забезпечується відповідне охолодження за допомогою вентиляційних отворів, вентиляторів або систем активного охолодження.

Статичний розряд небезпечний для електронних компонентів. Під час роботи з ПК, особливо при обслуговуванні апаратного забезпечення, використовуються антистатичні килимки, рукавички або запобіжні браслети. Ці пристрої допомагають розрядити статичний заряд з тіла працівника, запобігаючи його перенесенню на електронні компоненти та зменшуючи ризик їх пошкодження.

Також потрібно забезпечити регулярні інструктажі та навчання персоналу стосовно електробезпеки з обов’язковим підписом працівників у журналі техніки безпеки. Працівники ознайомлені з правилами безпечного користування ПК, включаючи правильні методи роботи, використання захисного обладнання та процедури поводження в екстрених ситуаціях. Це підвищує свідомість персоналу щодо потенційних ризиків та вмінням запобігати їм.

При виникненні пожеж гасити електроприлади водою категорично заборонено. Важливо відключити живлення електроприладу, якщо це безпечно зробити. Для гасіння пожежі спричиненими електроприладами використовуються вогнегасники, які призначені спеціально для цього типу пожеж: порошкові або вуглекислотні. Дані вогнегасники не проводять електричний струм і можуть загасити вогонь без ризику ураження струмом. Відповідні засоби гасіння пожеж розташовуються в кожному приміщенні з відповідним обладнанням. Особливо важливо мати вогнегасник у непосредственній близькості від області, де знаходяться ПК та електроніка. Встановлюються системи автоматичного вимкнення електропостачання в разі виявлення пожежі або диму, що сприяє швидкому припиненню живлення та запобігає подальшому розповсюдженню пожежі.

Ці заходи електробезпеки описані згідно ДСТУ Б В.2.5-82:2016 Електробезпека в будівлях і спорудах. Вимоги до захисних заходів від ураження електричним струмом та мають на меті забезпечення безпечних умов для роботи з ПК та запобігти можливим небезпекам, які можуть виникнути внаслідок електричних проблем. Регулярне перевіряння та підтримка обладнання також є важливими аспектами електробезпеки на підприємствах.

Збираючи підсумок усіх заходів електробезпеки на підприємствах можна створити блок-схему наведену на рис. 4.2 для кращого розуміння цієї проблеми.


Рисунок 4.2 – Блок-схема електробезпеки

# ВИСНОВКИ

Ця кваліфікаційна робота включає детальне дослідження процесу створення комп'ютерної системи для моделювання та документації мережевих ресурсів дата-центру. При її розробці ми використали широкий спектр технологій, включаючи Python, Django, PostgreSQL, а також середовище розробки PyCharm та системні інструменти NetBox.

Основну частину часу, витраченого на розробку, було приділено вивченню особливостей розробки REST API, зокрема формулювання та надсилання запитів до сервера. Не менш важливою була робота з конфігурацією сервера для розгортання програмного рішення, а також розуміння взаємодії таблиць у базі даних та розробки самої бази даних.

Результатом цього процесу є новий програмний продукт – розширення для існуючої комп’ютеризованої системи, призначене для доповнення її функціоналу. Це розширення успішно виконує поставлене завдання, відкриваючи шлях до майбутніх удосконалень і поліпшень. Серед потенційних напрямків доопрацювання – більш зручний інтерфейс та оптимізація для збільшення швидкості роботи, що значно підвищить ефективність та зручність роботи з системою.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Ерік М. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. 2019. 540 с.
2. Вільям С. Django for APIs: Build web APIs with Python & Django. 2018. 190 с.
3. Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development 3rd Edition. O'Reilly Media, 2022. 546 с.
4. Learning Python, 5th Edition. O'Reilly Media, 2013. 1643 с.
5. HTML & CSS: Design and Build Web Sites. John Wiley & Sons, 2011. 490 с.
6. Designing Web APIs: Building APIs That Developers Love. O'Reilly Media, 2018. 230 с.
7. Linux Bible 10th Edition. Willey, 2020. 928 с.
8. Restfulapi. URL: https://restfulapi.net/(Дата звернення 10.03.2023)
9. Medium. URL: https://medium.datadriveninvestor.com/django-data-migrations-1e13fdb916d4 (Дата звернення 19.03.2023)
10. Hashnode. URL: https://sarahthedeveloper.hashnode.dev/how-to-create-a-rest-api-with-django-rest-framework (Дата звернення 20.03.2023)
11. Dev. URL: https://dev.to/kostjapalovic/django-project-tutorial-for-beginners-settings-docker-compose-postgres-and-redis-1gdj(Дата звернення 15.05.2023)
12. ttl255 URL: https://ttl255.com/developing-netbox-plugin-part-1-setup-and-initial-build/ (Дата звернення 12.04.2023)
13. Django Documentation. URL: https://docs.djangoproject.com/en/4.2/ (Дата звернення 20.03.2023)
14. NetBox Documentation. URL: https://docs.netbox.dev/en/stable/ (Дата звернення 22.03.2023)
15. Docker Docs. URL: https://docs.docker.com/ (Дата звернення 02.04.2023)

16. Python Documentation. URL: https://www.python.org/doc/(Дата звернення 05.04.2023)
17. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (Дата звернення 03.04.2023)
18. ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги. Київ, 2015. 9 с.
19. ДСТУ Б В.2.5-82:2016 Електробезпека в будівлях і спорудах. Вимоги до захисних заходів від ураження електричним струмом. Київ, 2016. 82 с.
20. ДСТУ 3675-98 Пожежна техніка. Вогнегасники переносні. Загальні технічні вимоги та методи випробувань. Зі зміною № 1. Київ, 1998. 66 с.
21. ДСТУ EN 12464-1:2016 Світло та освітлення. Освітлення робочих місць. Частина 1. Внутрішні робочі місця (EN 12464-1:2011, IDT). Київ, 2016. 11 с.

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра комп'ютерних систем та мереж

«Затверджую»

завідувач кафедри КС

_____ Осухівська Г.М.

"____ "_____ 2023 р.

# Комп'ютеризована система моделювання і документування мережевих ресурсів дата-центру

## ТЕХНІЧНЕ ЗАВДАННЯ

На 6 листах

Вид робіт: кваліфікаційна робота

На здобуття освітньо ступеня «Бакалавр»

Спеціальність 123 «Комп'ютерна інженерія»

«УЗГОДЖЕНО»

Керівник кваліфікаційної роботи

_____ к.т.н., доц., Тиш Є. В.

« ____ » _____ 2023 р.

«ВИКОНАВЕЦЬ»

Студент групи СІ-41

_____ Папка О. В.

« ____ » _____ 2023 р.

Тернопіль 2023

# 1 Загальні відомості

## 1.1 Повна назва та її умовне позначення

Повна назва теми кваліфікаційної роботи: «Комп’ютеризована система моделювання і документування мережевих ресурсів дата-центру».

Умовне позначення кваліфікаційної роботи: КС КРБ 123.052.00.00 .

## 1.2 Виконавець

Студент групи СІ-41, факультету комп’ютерно-інформаційних систем і програмної інженерії, кафедри комп’ютерних систем та мереж, Тернопільського національного технічного університету імені Івана Пулюя, Папка Олег Вікторович.

## 1.3 Підстава для виконання роботи

Підставою для виконання кваліфікаційної роботи є наказ по університету (№ 4/7-238 від 28.02.2023 р.)

## 1.4 Планові терміни початку та завершення роботи

Плановий термін початку виконання кваліфікаційної роботи – 28.02.2023 р.

Плановий термін завершення виконання кваліфікаційної роботи – 20.06.2023 р.

### 1.5 Порядок оформлення та представлення результатів роботи

Порядок оформлення пояснювальної записки та графічного матеріалу здійснюється у відповідності до чинних норм та правил ІСО, ГОСТ, ЕСКД, ЕСПД та ДСТУ.

Пред’явлення проміжних результатів роботи з виконання кваліфікаційної роботи здійснюється у відповідності до графіку, затвердженого керівником роботи.

Попередній захист кваліфікаційної роботи відбувається при готовності роботи на 90% , наявності пояснювальної записки та графічного матеріалу.

Пред’явлення результатів кваліфікаційної роботи відбувається шляхом захисту на відповідному засіданні ЕК, ілюстрацією основних досягнень з допомогою графічного матеріалу.

## 2 Призначення і цілі створюваної системи

### 2.1 Призначення системи

Призначенням системи є зручне зберігання та управління списками блокованих та не блокованих IP-адрес та їх підмереж, з реалізацією усіх базових функцій NetBox, та можливістю перевірки наявності потрібної адреси в даних списках.

### 2.2 Мета створення системи

Метою створення системи є виконання потреб наявних користувачів, задля розширення функціоналу наявної системи, для покращення взаємодії та збільшення можливостей NetBox.

# 3 Вимоги до системи

## 3.1 Вимоги до системи

Під час відправлення запиту від клієнта на сервер, повинна повертатись відповідь в залежності від обраної дії користувача або адміністратора. Система має працювати чітко згідно поставленої мети.

### 3.1.1 Вимоги до структури та функціонування системи

Структура системи складається з:

- Бази даних
- Веб-сервера
- КС NetBox
- Модуль розширення

### 3.1.2 Вимоги по діагностуванню

Діагностика та профілактика програмних і апаратних частин сайту відбувається за допомогою системного адміністратора.

### 3.1.3 Перспективи розвитку програмного продукту

До перспектив належать:

- Оптимізація розширення;
- Оптимізація користувацького інтерфейсу.

### 3.1.4 Вимоги до надійності системи

В КС має бути розроблена система авторизації користувачів. Надати можливість адміністраторам надавати та обмежувати права доступу до системи.

### 3.1.5 Вимоги до апаратного забезпечення

Вимоги до сервера:

– Процесор із тактовою частотою не менше 2.4 ГГЦ з кількістю логічних не менше 8.

– Об’єм оперативної пам’яті не менше 16 ГБ.

– Об’єм на жорсткому диску (SSD/HDD) не менше 512 ГБ.

Вимоги до клієнтського ПК:

– Процесор із тактовою частотою не менше 2.4 ГГЦ з кількістю логічних не менше 2.

– Об’єм оперативної пам’яті не менше 4 ГБ.

– Об’єм на жорсткому диску (SSD/HDD) не менше 240 ГБ.

### 3.1.6 Вимоги до програмного забезпечення

Програмне забезпечення сервера: Linux, Git, Ubuntu, Docker.

Програмне забезпечення клієнта: будь-яка ОС та браузер.

Використаний редактор коду: PyCharm.

## 4 Вимоги до документації

Документація повинна відповідати вимогам ЄСКД та ДСТУ. Комплект документації повинен складатись з пояснювальної записки.

## 5 Стадії та етапи проєктування

Таблиця 1 – Стадії та етапи виконання кваліфікаційної роботи бакалавра

| № з/п | Назва етапів роботи | Термін виконання етапів роботи |
|---|---|---|
| 1. | Ознайомлення з завданням кваліфікаційної роботи | 30.02-05.03.2023 |
| 2. | Огляд літератури за темою кваліфікаційної роботи | 05.03-20.03.2023 |
| 3. | Аналіз особливостей проектування, та розробка застосунку для ІТ компанії. | 20.03-23.03.2023 |
| 4. | Практична реалізація об’єкта проектування | 23.03-29.03.2023 |
| 5. | Виконання завдання до підрозділу «Безпека життєдіяльності» | 29.03-05.04.2023 |
| 6. | Виконання завдання до підрозділу «Основи охорони праці» | 05.04-09.04.2023 |
| 7. | Оформлення кваліфікаційної роботи | 15.05-21.05.2023 |
| 8. | Нормоконтроль | 12.06-13.06.2023 |
| 9. | Перевірка на плагіат | 14.06-15.06.2023 |
| 10. | Попередній захист кваліфікаційної роботи | 16.06-19.06.2023 |
| 11. | Захист кваліфікаційної роботи | 20.06-23.06.2023 |

Додаток Б

Лістинг розширення

```
import re
import netaddr
from django.core.exceptions import ValidationError
from django.db.models import Q
from django.shortcuts import render
from django.urls import reverse
from django.views import View

from netbox.views import generic
from utilities.forms import restrict_form_fields
from . import models, forms, tables, filtersets
from .utils import is_ip


# DissociatedIP

class DissociatedIPView(generic.ObjectView):
    template_name = "ip_check/dissociated_ip.html"
    queryset = models.DissociatedIP.objects.all()

    def get_extra_context(self, request, instance):
        return {
            "associated": instance.associatedips.all()
        }


class DissociatedIPListView(generic.ObjectListView):
    template_name = "ip_check/list_view_check.html"
    queryset = models.DissociatedIP.objects.all()
    table = tables.DissociatedIPTable
    filterset = filtersets.DissociatedIPFilterSet
    filterset_form = forms.DissociatedIPFilterForm

    def get_extra_context(self, request):
        return {
            'check_link': reverse("plugins:ip_check:ip_check"),
        }


class DissociatedIPCreateView(generic.ObjectEditView):
    template_name = "ip_check/dissociated_ip_create.html"
    queryset = models.DissociatedIP.objects.all()
    form = forms.DissociatedIPCreateForm


class DissociatedIPEditView(generic.ObjectEditView):
    template_name = "ip_check/dissociated_ip_create.html"
    queryset = models.DissociatedIP.objects.all()
    form = forms.DissociatedIPEditForm
```

```
class DissociatedIPDeleteView(generic.ObjectDeleteView):
    queryset = models.DissociatedIP.objects.all()


class DissociatedIPBulkImportView(generic.BulkImportView):
    queryset = models.DissociatedIP.objects.all()
    model_form = forms.DissociatedIPImportForm
    table = tables.DissociatedIPTable

    default_return_url = "plugins:ip_check:dissociatedip_list"

    def _create_objects(self, form, request):
        new_objs = []
        if request.FILES:
            headers, records = form.cleaned_data['csv_file']
        else:
            headers, records = form.cleaned_data['csv']

        seen = set()
        records_unique = []
        for record in records:
            ip = record['address']

            if ip not in seen:
                seen.add(ip)
                records_unique.append(record)

        for row, data in enumerate(records_unique, start=1):
            obj_form = self.model_form(data, headers=headers)
            restrict_form_fields(obj_form, request.user)

            if obj_form.is_valid():
                obj = self._save_obj(obj_form, request)
                new_objs.append(obj)
            else:
                for field, err in obj_form.errors.items():
                    form.add_error('csv', f'Row {row} {field}:
{err[0]}')
                raise ValidationError("")

        return new_objs


class DissociatedIPBulkEditView(generic.BulkEditView):
    queryset = models.DissociatedIP.objects.all()
    filterset = filtersets.DissociatedIPFilterSet
    table = tables.DissociatedIPTable
    form = forms.DissociatedIPBulkEditForm


class DissociatedIPBulkDeleteView(generic.BulkDeleteView):
    queryset = models.DissociatedIP.objects.all()
    filterset = filtersets.DissociatedIPFilterSet
    table = tables.DissociatedIPTable
```

```
# IP check

class IPCheckView(View):
    search_template = "ip_check/ip_check.html"

    def get(self, request):
        return render(request, self.search_template, {"return_url":
reverse("plugins:ip_check:dissociatedip_list")})

    def post(self, request):
        raw_data = request.POST.get("data", "").strip()
        result = []

        ips = re.split(r"[^0-9.]", raw_data)

        for ip in ips:
            # Check if this is an IP
            if not is_ip(ip):
                continue
            elif ip == '0.0.0.0':
                result.append({"ip": ip,
                               "message": "Wrong IP address
given!"})
                continue

            ip_obj, list_type, network = None, None, None
            message = f"IP address {ip} not found in any of the
lists."

            if in_dissociated_list :=
models.DissociatedIP.objects.filter(

Q(address__net_contains_or_equals=netaddr.IPNetwork(ip).ip)):
                list_type = "Dissociated"
                network = in_dissociated_list[0]

                if in_associated_list :=
models.AssociatedIP.objects.filter(Q(address=ip)):
                    ip_obj = in_associated_list[0]
                    list_type = "Associated"

                message = f"IP address {ip} present in {list_type}
IP list."

            result.append({
                "ip": ip,
                "ip_obj": ip_obj,
                "network": network,
                "list_type": list_type,
                "message": message
            })
```

```
        return render(request, "ip_check/ip_check_result.html",
{"results": result})


# AssociatedIP

class AssociatedIPView(generic.ObjectView):
    template_name = "ip_check/associated_ip.html"
    queryset = models.AssociatedIP.objects.all()


class AssociatedIPListView(generic.ObjectListView):
    template_name = "ip_check/list_view_check.html"
    queryset = models.AssociatedIP.objects.all()
    table = tables.AssociatedIPTable
    filterset = filtersets.AssociatedIPFilterSet
    filterset_form = forms.AssociatedIPFilterForm

    def get_extra_context(self, request):
        return {
            'check_link': reverse("plugins:ip_check:ip_check"),
        }


class AssociatedIPCreateView(generic.ObjectEditView):
    template_name = "ip_check/associated_ip_create.html"
    queryset = models.AssociatedIP.objects.all()
    form = forms.AssociatedIPCreateForm


class AssociatedIPEditView(generic.ObjectEditView):
    template_name = "ip_check/associated_ip_create.html"
    queryset = models.AssociatedIP.objects.all()
    form = forms.AssociatedIPEditForm


class AssociatedIPDeleteView(generic.ObjectDeleteView):
    queryset = models.AssociatedIP.objects.all()


class AssociatedIPBulkImportView(generic.BulkImportView):
    queryset = models.AssociatedIP.objects.all()
    model_form = forms.AssociatedIPImportForm
    table = tables.AssociatedIPTable

    default_return_url = "plugins:ip_check:associatedip_list"

    def _create_objects(self, form, request):
        new_objs = []
        if request.FILES:
            headers, records = form.cleaned_data['csv_file']
        else:
            headers, records = form.cleaned_data['csv']
```

```
        seen = set()
        records_unique = []
        for record in records:
            ip = record['address']

            if ip not in seen:
                seen.add(ip)
                records_unique.append(record)

        for row, data in enumerate(records_unique, start=1):
            obj_form = self.model_form(data, headers=headers)
            restrict_form_fields(obj_form, request.user)

            if obj_form.is_valid():
                obj = self._save_obj(obj_form, request)
                new_objs.append(obj)
            else:
                for field, err in obj_form.errors.items():
                    form.add_error('csv', f'Row {row} {field}:
{err[0]}')
                raise ValidationError("")

        return new_objs


class AssociatedIPBulkEditView(generic.BulkEditView):
    queryset = models.AssociatedIP.objects.all()
    filterset = filtersets.AssociatedIPFilterSet
    table = tables.AssociatedIPTable
    form = forms.AssociatedIPBulkEditForm


class AssociatedIPBulkDeleteView(generic.BulkDeleteView):
    queryset = models.AssociatedIP.objects.all()
    filterset = filtersets.AssociatedIPFilterSet
    table = tables.AssociatedIPTable
```