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 наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка DHCP-сервера

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Мельник В.О.
(підпис) (прізвище та ініціали)

Керівник _____ Готович В.А.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Луцків А.М.
(підпис) (прізвище та ініціали)

Тернопіль
2021

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 наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Мельник Володимир Олегович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка DHCP-сервера

Керівник роботи к.т.н кафедри КН Готович Володимир Анатолійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «02» березня 2021 року № 4/7-171

2. Термін подання студентом завершеної роботи 22 червня 2021 року

3. Вихідні дані до роботи Літературні та інтернет джерела про розробку DHCP-серверів

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Протокол DHCP та проектування DHCP-сервера. 1.2. DHCP-протокол. 1.3 Причини використання DHCP. 1.4. Принцип роботи DHCP-протоколу. 1.5. Пошук актантів та варіантів використання. 1.6. Вибір засобів розробки. 1.7. Висновок до першого розділу. 2. Реалізація програмного забезпечення для DHCP-сервера. 2.1. Реалізація DHCP-сервера 2.2. Розробка модуля для керування DHCP-сервером.2.3. Розробка графічного інтерфейсу. 2.4. Тестування розробленого DHCP-сервера. 2.5. Висновки до другого розділу. 3. Безпека життєдіяльності, основи охорони праці. 3.1. Характеристика життєдіяльності людини у системі „людина-машина–середовище існування". 3.2. Аналіз потенційних шкідливостей на робочому місці. Заходи щодо їх зниженню. 3.3.Висновки до третього розділу. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульна сторінка. 2. Актуальність теми роботи. 3. Мета дослідження. 4. Практичне Значення отриманих результатів. 5. DHCP-протокол. 6. Причини використання. 7. Принципи роботи DHCP-протоколу. 8. Варіанти використання. 9. Вибір засобів розробки. 10. Структура проекту. 11. Життєвий цикл запиту. 12. Функція DHCPServe. 13. Модуль керування DHCP-сервером. 14. Основні операції в модулі керування. 15. Тестування . 16. Вигляд адмін-сторінки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Гурик О.Я., доцент кафедри МТ		

7. Дата видачі завдання 25 січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2021	*Виконано*
2.	Підбір джерел про DHCP-протокол та розробку	27.01.2021-31.01.2021	*Виконано*
	DHCP-сервера		
3.	Переклад та опрацювання джерел про розробку	01.02.2021-05.02.2021	*Виконано*
	DHCP-серверів		
4.	Виконання дослідження щодо розробки DHCP-сервер	06.02.2021-18.04.2021	*Виконано*
	Розроблення DHCP-сервера		
5.	Оформлення розділу «Протокол DHCP та проектува-	18.04.2021-10.05.2021	*Виконано*
	ння DHCP-сервера»		
6.	Оформлення розділу «Реалізація програмного забез-	11.05.2021-17.06.2021	*Виконано*
	печення для DHCP-сервера»	17.06.2021	
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»		*Виконано*
8.	Виконання завдання до підрозділу «Основи охрони	17.06.2021	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	18.06.2021	*Виконано*
10.	Нормоконтроль	19.06.2021	*Виконано*
11.	Перевірка на плагіат	19.06.2021	*Виконано*
12.	Попередній захист кваліфікаційної роботи	19.06.2021	*Виконано*
13.	Захист кваліфікаційної роботи	22.06.2021	

Студент _____ Мельник В.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка DHCP-сервера // Кваліфікаційна робота освітнього рівня «Бакалавр» // Мельник Володимир Олегович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2021 // С. 65, рис. – 19 , табл. – 1 , кресл. – 0, додат. – 2 , бібліогр. – 27.

Ключові слова: DHCP-сервер, проектування, UDP, TCP/IP, Go, React, OSI.

Кваліфікаційна робота присвячена розробці програмного забезпечення на прикладі DHCP-сервера для автоматичної конфігурації клієнтів в локальних мережі.

Мета роботи: розробити DHCP-сервер, який дозволить спростити процес підключення до локальної мережі, та моніторингу підключених клієнтів.

В першому розділі кваліфікаційної роботи розглянуто DHCP-протокол, а саме його історію розвитку, алгоритм роботи, причини його використання. А також вибір інструментів, які використовувалися під час розробки.

В другому розділі кваліфікаційної роботи розглянуто В другому розділі кваліфікаційної роботи розглянуто практичну реалізацію DHCP-сервера.

ANNOTATION

DHCP-server development // Qualification work of the educational level "Bachelor" // Melnyk Volodya // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, group SN -41 // Ternopil, 2021 // P. 65, pic. 19, table. 1, draw. 0, appendix. 2, bibliogr. 27.

Keywords: DHCP-server, designing, UDP, TCP/IP, Go, React, OSI.

Qualification work dedicated to DHCP-server for automated configurations clients in local network.

The goal of the work: development of DHCP-server that will which will simplify the process of connecting to the local network, and monitoring connected customers.

In the first section of the qualification work the DHCP protocol is considered, namely its history of development, algorithm of work, reasons of its use. As well as a selection of tools used during development.

In the second section of the qualification work the practical implementation of DHCP-server is considered.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application programming interface) — програмний інтерфейс.

ARP (англ. Address Resolution Protocol) — Протокол визначення адрес.

BOOTP (англ. Bootstrap Protocol) — мережевий протокол для автоматичного отримання клієнтом IP-адреси.

DHCP (англ. Dynamic Host Configuration Protocol) — протокол динамічної конфігурації вузла.

IP (англ. Internet Protocol) — протокол мережевого рівня для передачі датаграм між мережами.

ISP (англ. Internet Service Provider) провайдер послуг Інтернету

OSI — абстрактна мережева модель для комунікацій і розробки мережевих протоколів.

RARP (англ. Reverse Address Resolution Protocol) — Зворотній протокол визначення адрес.

RFC (англ. Request for Comments) — документ із серії пронумерованих інформаційних документів Інтернету.

TCP (англ. Transmission Control Protocol) — Протокол призначений для керування передачею даних у комп’ютерних мережах.

UDP (англ. User Datagram Protocol) — Протокол датаграм користувача.

UI (англ. User interface) – графічний інтерфейс з яким може працювати користувач.

ЗМІСТ

ВСТУП

Актуальність теми. Час це невід'ємна частина світу, де ми живемо. Ми можемо спостерігати за його плином кожної миті незалежно від того де ми перебуваємо. Неважливо чим ми займаємося: чи працюємо над виробництвом; чи здійснюємо дослідження, які можуть кардинально змінити спосіб життя всього населення; чи просто відпочиваємо дивлячись як відроджується природа після довгої зими, як зеленіють листочки на гілках деревах, що змінює природу і дає нам новий крок еволюції.

Це визначення появилося разом з появою людства. Воно дозволяє нам розташувати різноманітні події та об'єкти в хронологічному ряді. Ми часто зберігаємо згадки про них в різноманітних формах: текстові, зображеннях, піснях, чи просто в спогадах. В подальшому ми їх порівнюємо, проводимо аналіз, робимо висновки, які допоможуть нам в майбутньому, які дозволять нам краще зрозуміти світи, допомогти у вирішенні задач, які ставить перед собою людство або його частина.

Оглядаючись назад, у минуле, в людей часто виникає здивування - з якою швидкістю проходить час. Це пов'язано з тим як кардинально відрізняється минуле і сучасність у різноманітних сферах нашого життя. Особливо це відчувається в сфері технологій.

Ще кілька десятиліть тому, доступ до комп'ютерних мереж мали лише працівники науково-дослідних лабораторій та великих корпорацій, а зараз більша частка населення Землі, яка постійно збільшується має цей доступ [1]. Тому виникає потреба спрощення існуючих технологій, щоб кожна людина могла користуватися досягненням технічного прогресу без попередньої підготовки.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є:

– Проаналізувати необхідність DHCP-серверів в сучасних мережах.

– Проаналізувати технології, які найкраще підійдуть для створення DHCP-сервера.

– Створити DHCP-сервер, який допоможе людям без досвіду налаштовувати складні мережі.

Практичне значення одержаних результатів. На практиці отриманий результат спрощує налаштування маршрутизаторів користувачів, дозволить мережевим адміністраторам зручніше аналізувати кількість підключених клієнтів, скільки часу вони користуються мережею.

РОЗДІЛ 1. ПРОТОКОЛ DHCP ТА ПРОЕКТУВАННЯ DHCP-СЕРВЕРА

1.1 DHCP-протокол

DHCP — це протокол керування мережею, який використовують в мережах TCP/IP для автоматичного призначення IP-адрес та інших параметрів, таких як: адресу DNS-сервера, маску підмережі користувачам, які підключеним до мережі [2]. Цей протокол працює за схемою архітектури клієнт-сервер.

Ця технологія позбавляє необхідності індивідуального налаштування мережевих інтерфейсів на пристроях користувачів вручну і складається з двох мережевих компонентів, централізовано встановленого мережевого DHCP-сервера та клієнтів, що працюються на кожному пристрої. При підключені до мережі і періодично під час роботи клієнта з мережею клієнт запитує набір параметрів від DHCP-сервера, використовуючи протокол DHCP [3].

DHCP може бути впроваджений в мережах, як у великих мережах підприємств, домашніх мережах, а також в регіональних мережах провайдера. Багато маршрутизаторів та побутових шлюзів мають можливість DHCP-сервера [4]. Більшість мережевих маршрутизаторів отримують унікальну IP-адресу в мережі ISP. У локальній мережі сервер DHCP призначає локальну IP-адресу для кожного клієнта.

1.2 Виникнення протоколу DHCP

Першим протоколом, що дозволяє вирішувати проблему перетворення фізичної адреси в IP-адресу був RARP, визначений в RFC 903. Цей протокол дозволяє пристроям динамічно отримувати відповідну IP-адресу. Він працює на третьому рівні моделі OSI - мережевому рівні, це значно ускладнювало реалізацію на багатьох платформах, а також вимагає присутності сервера на кожному мережевому каналі [5]. Протокол RARP згодом був замінений протоколом BOOTP, який задокументований в RFC 951. В цьому протоколі

добавили агента ретрансляції, який дозволяє переадресовувати пакети BOOTP через підмережі, дозволяючи одному серверу BOOTP обслуговувати хости у підмережах [6].

DHCP базується на протоколі BOOTP, проте може динамічно розподіляти IP-адреси з пулу адрес та повертати їх назад в нього, коли вони більше не потрібні клієнту [7]. Також може бути використаний для надання конфігурації IP-клієнтам. Стандартом вважається RFC 2131.

1.3 Причини використання DHCP

Кожний пристрій в мережі на основі TCP/IP повинен мати унікальний IP-адрес для доступу в мережу. Без DHCP IP-адрес нових комп'ютерів або комп'ютерів переміщених з однієї підмережі в іншу, потрібно вручну налаштовувати. IP-адреси для пристроїв відключених від мережі, потрібно вручну звільняти [8].

При використанні DHCP весь процес автоматизований і керується централізовано. DHCP-сервер підтримує пул IP-адрес і дозволяє клієнту з підтримкою DHCP орендувати довільну вільну адресу з пулу адрес. Так як IP-адреси є динамічними, а не статичними, адреса, яка більше не використовується, автоматично повертається в пул для перерозподілу.

Мережевий адміністратор встановлює DHCP-сервери, які зберігають інформацію про конфігурацію TCP/IP і надають клієнтам з підтримкою DHCP в формі продовження оренди [9]. DHCP-сервер зберігає відомості про конфігурацію у базі даних, яка включає в себе:

- Доступні параметри конфігурації TCP/IP для всіх клієнтів в мережі.
- Доступні IP-адреси, доступні в пулі для призначення клієнтам, а також вилучені адреси з пулу.
- Зарезервовані IP-адреси, пов'язані з конкретним клієнтом, що забезпечує отримання тієї самої IP-адреси, одному клієнту.

- Термін дії оренди або період часу, протягом якого IP-адрес може бути використаний для продовження терміну оренди.
- Клієнт з підтримкою DHCP при прийнятті оренди отримує:
- Доступний IP-адрес для підмережі, до якої підключається.
- Запрошені параметри DHCP, які налаштовуються на сервері, такі як: шлюз по замовчуванні, DNS-сервери, доменне ім’я DNS.

Протокол DHCP робить життя мережевого інженера більш простим і приємною, правда цьому інженеру доведеться зрозуміти як використовувати цей протокол [10].

1.4 Принцип роботи DHCP-протоколу

DHCP використовує сервісну модель без підключень, використовуючи протокол UDP. Використовує два номери портів UDP для своїх операції. UDP порт 67 призначений для сервера, а порт 68 призначений для клієнтів [11].

Операції DHCP поділяються на 4 фази: виявлення сервера, пропозиція оренди IP, запит на оренду IP та підтвердження оренди IP. Графічно фази можна побачити на рисунку 1.1.

Операція DHCP починається з того, що клієнт робить широкомовний запит. Коли клієнт і DHCP-сервер знаходяться в різних мережах, то використовується агент ретрансляції DHCP.

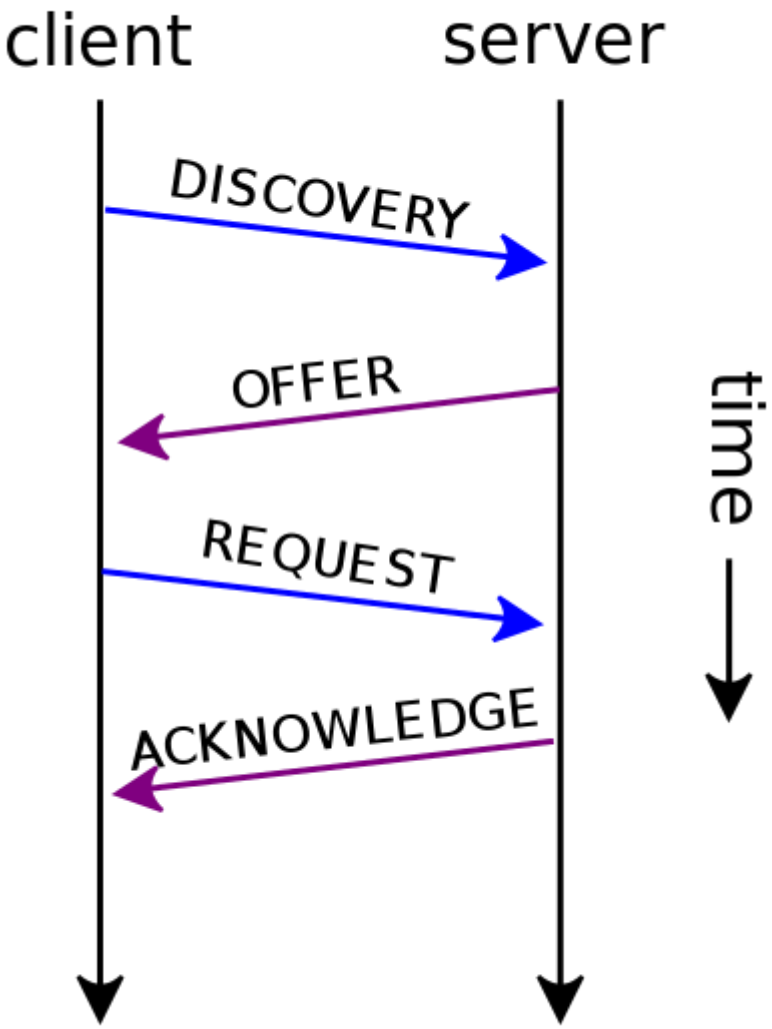

Рисунок 1.1 — Операції DHCP

1.4.1 Discovery

Клієнт DHCP робить широкомовний запит з повідомленням DHCP Discover у підмережі до якої підключений, використовуючи адресу призначення 255.255.255.255. Клієнт може запросити останню IP-адресу якою він користувався, сервер може задовольнити цей запит, якщо ця IP-адреса в конкретний момент доступна в пулі IP-адрес, а також налаштування самого сервера дозволяють виконати таку функцію. На рис. 1.2 зображений discovery

запит клієнта з MAC-адресою 0c:e5:20:52:89:00. Цей пакет містить ім’я хоста, а також його MAC-адресу.

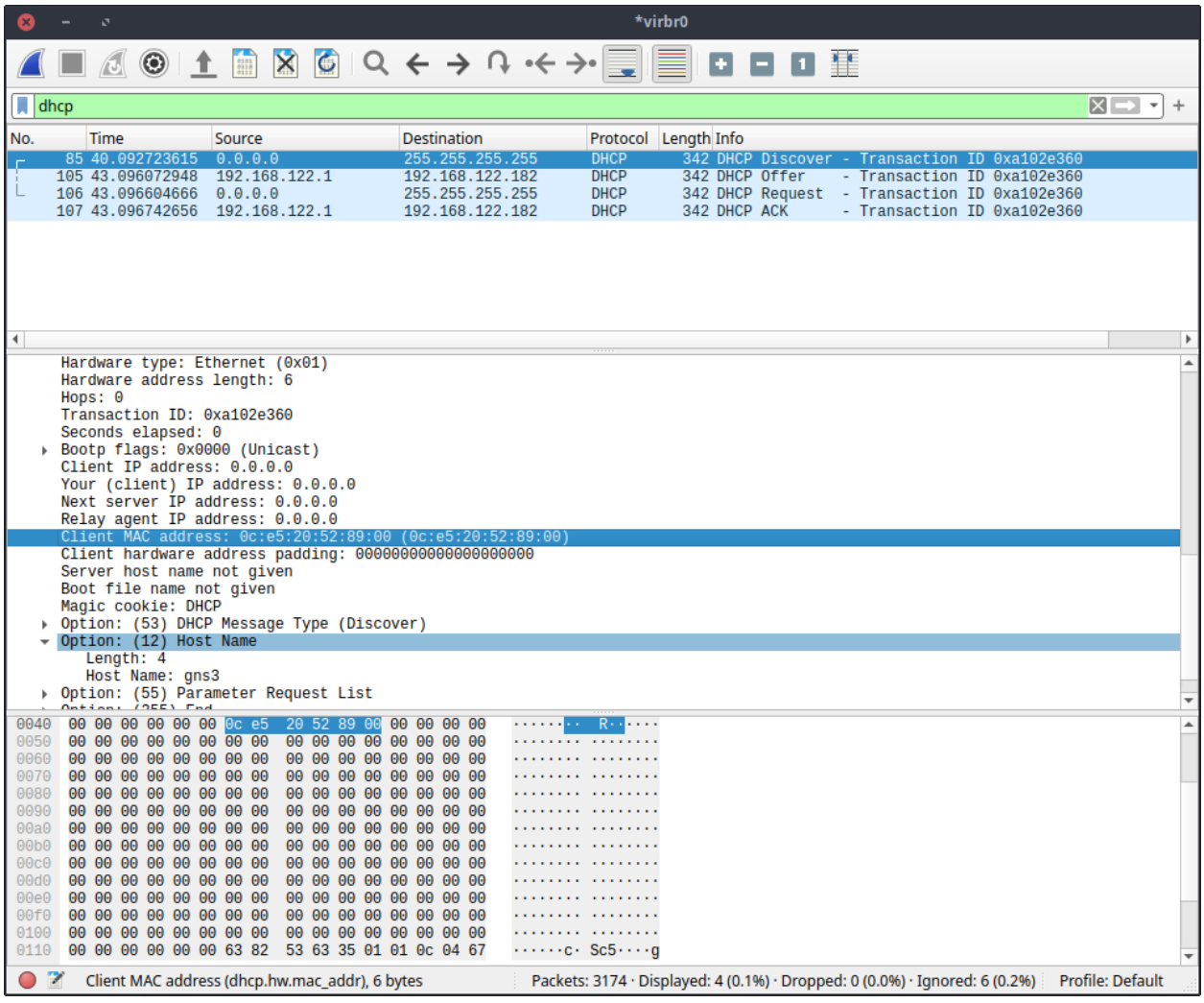

Рисунок 1.2 — запит DHCP Discover

Основне призначення цього пакету можна інтерпретувати як: “Я шукаю DHCP-сервер, який може надати оренду IP-адреси”.

1.4.2 Offer

Коли DHCP-сервер отримує повідомлення Discover від клієнта, що є запитом на оренду IP-адреси, DHCP-сервер резервує IP-адресу для цього клієнта і пропонує йому оренду відповідаючи повідомлення DHCP Offer. Це повідомлення містить ідентифікатор клієнта — MAC-адресу, IP-адресу, яку

пропонує сервер, маску підмережі, IP-адресу DHCP-сервера, тривалість оренди та адреси DNS-серверів. Розглянути DHCP Offer запит можна на рисунку 1.3.

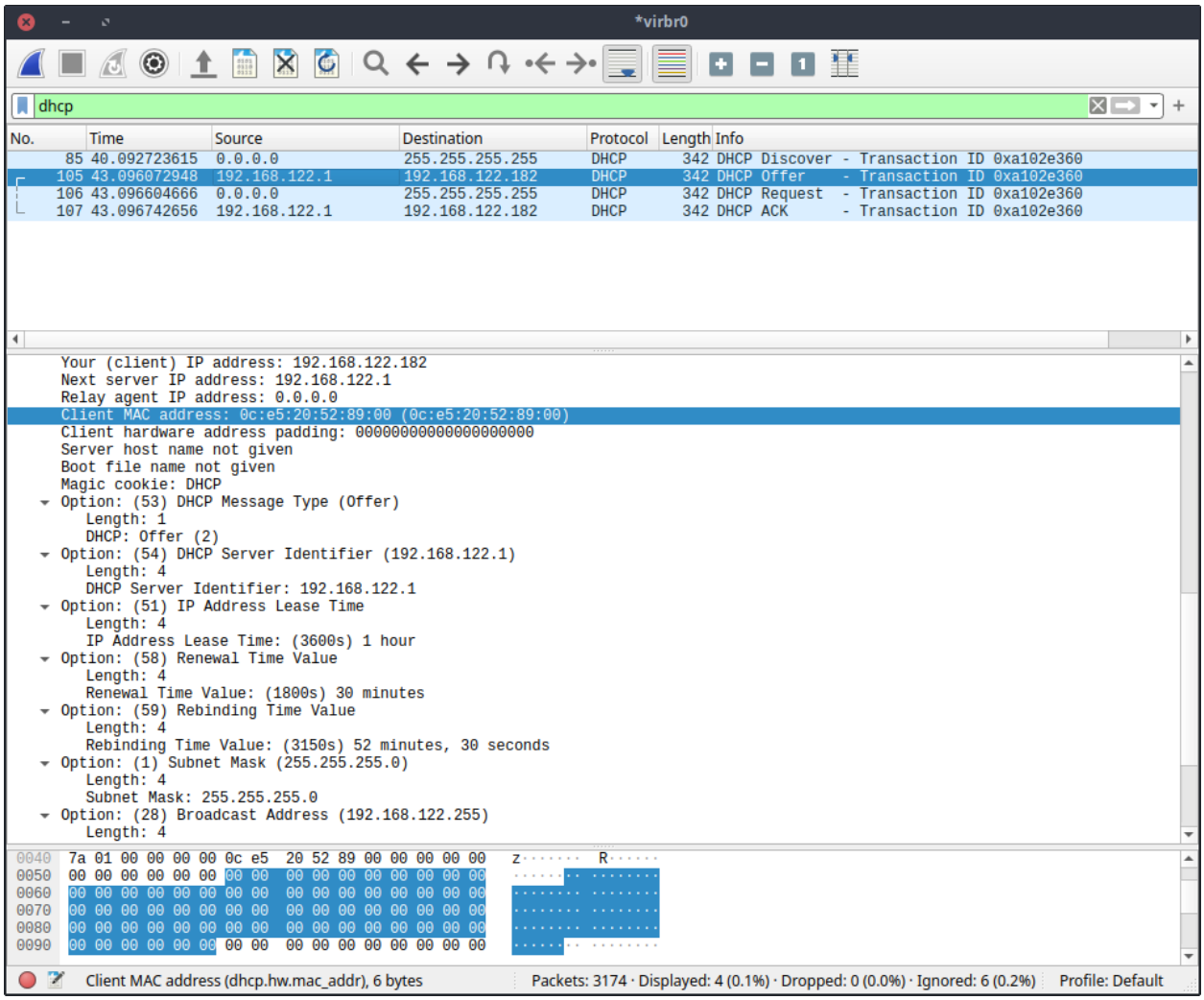

Рисунок 1.3 — запит DHCP Offer

На рисунку 1.3 можемо спостерігати, що DHCP-сервер пропонує клієнту з MAC-адресою 0c:e5:20:52:89:00 наступну IP-адресу 192.168.122.182, час оренди 1 год, маску підмережі - 255.255.255.0.

1.4.3 Request

У відповідь на пропозицію оренди, DHCP клієнт відповідає повідомленням DHCP Request, яка відправляється широкомовним запитом на сервер, запитуючи запропоновану раніше адресу. Клієнт може отримати

пропозиції від кількох серверів, але приймає лише одну. Клієнт робить ARP запит, щоб з'ясувати, чи є інший хост із запропонованими даними, якщо інший хост не відповів, то в мережі немає іншого хоста з цією конфігурацією і клієнт робить запит, що приймає запропоновану IP-адресу. Цей запит отримують всі DHCP-сервери в підмережі і всі сервери від яких хост проігнорував пропозицію повертають запропоновані ними IP-адреси до свого пулу доступних адрес в даний момент часу.

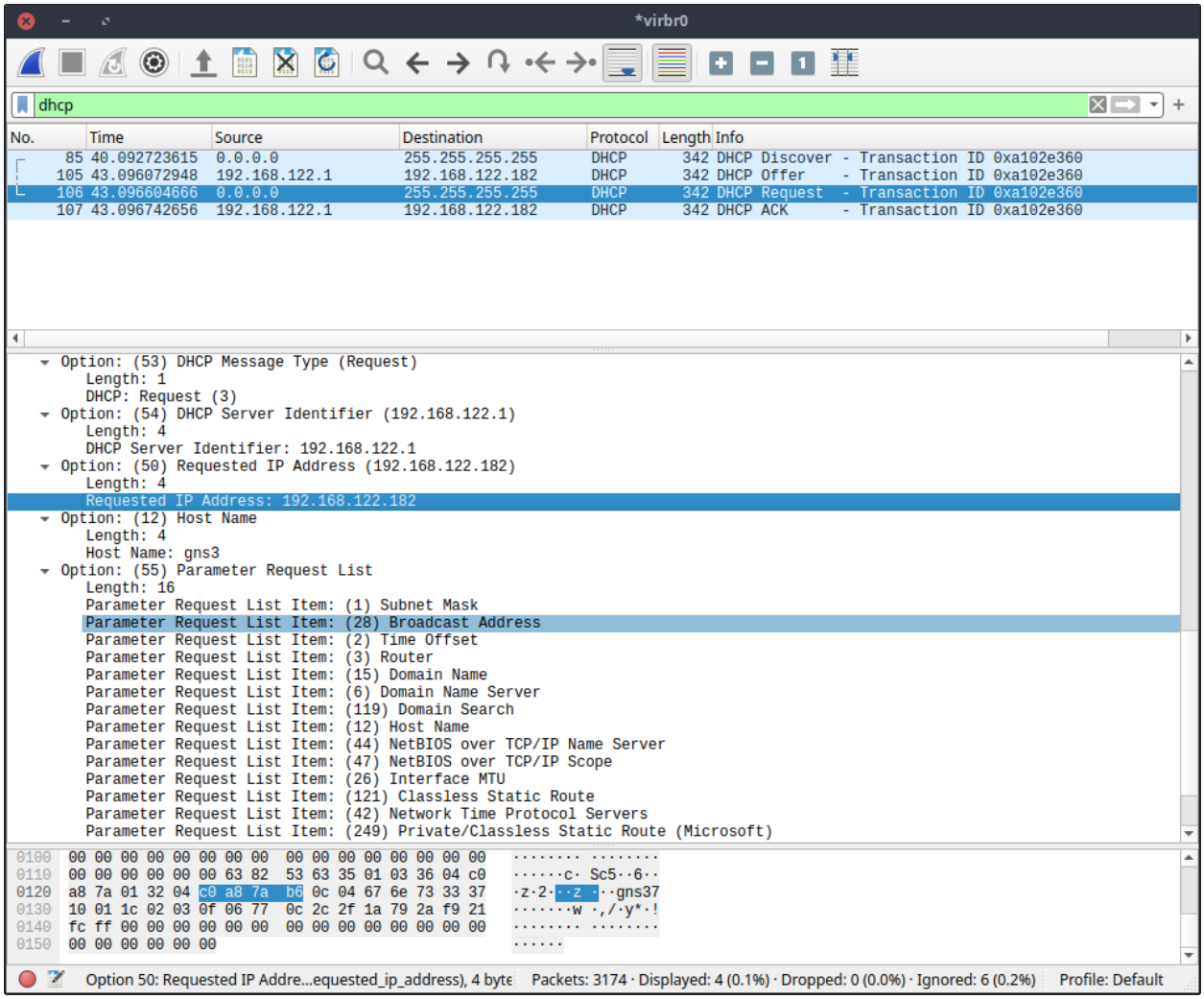

Рисунок 1.4 — Запит DHCP Request

На рисунку 1.4 ми можемо бачити, як клієнт відправляє DHCP Request запит в якому запитує раніше запропоновану клієнту IP-адресу 192.168.122.182.

1.4.4 Acknowledgement

Коли DHCP-сервер отримує повідомлення DHCP Request від клієнта, процес конфігурації переходить на останню стадію. Тобто етап підтвердження передбачає надіслання клієнту пакету DHCP Pack. Цей пакет містить інформацію про тривалість оренди, та іншу інформацію про конфігурацію, яку може запросити клієнт.

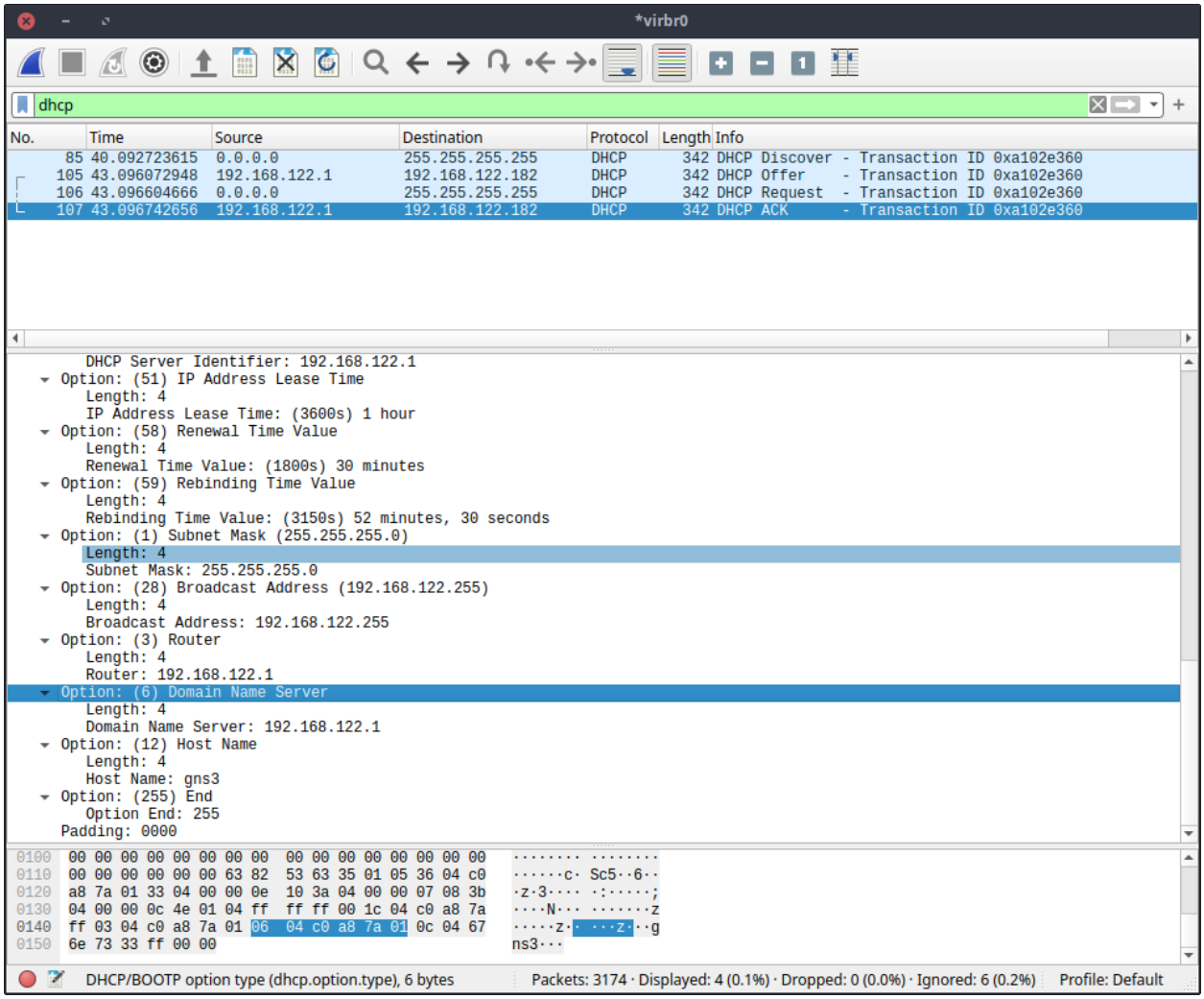

Рисунок 1.5 — Запит DHCP ACK

Як ми бачимо з рисунка 1.5 DHCP-сервер надіслав остаточну інформацію про конфігурацію, яку повинен використовувати для своєї роботи. Протоколом передбачено, що клієнти налаштують свої мережеві інтерфейси узгоджено до DHCP-сервера.

1.5 Пошук актантів та варіантів використання

Будь-яке програмне забезпечення працює в контексті, що визначає зовнішнє оточення системи. Таке оточення формують актори DHCP-сервер, якими можуть слугувати пристрої, які ним будуть користуватися, а також адміністратори. Кожен з акторів взаємодіє з DHCP-сервером за власною схемою та очікує від нього певну поведінку. Схему цією взаємодії називають варіантами використання.

В DHCP-сервері, який розробляється в цій кваліфікаційній роботі фігурують наступні актори:

- клієнти;
- адміністратор.

Таким чином діаграма прецедентів має вигляд (див Рисунок 1.6).

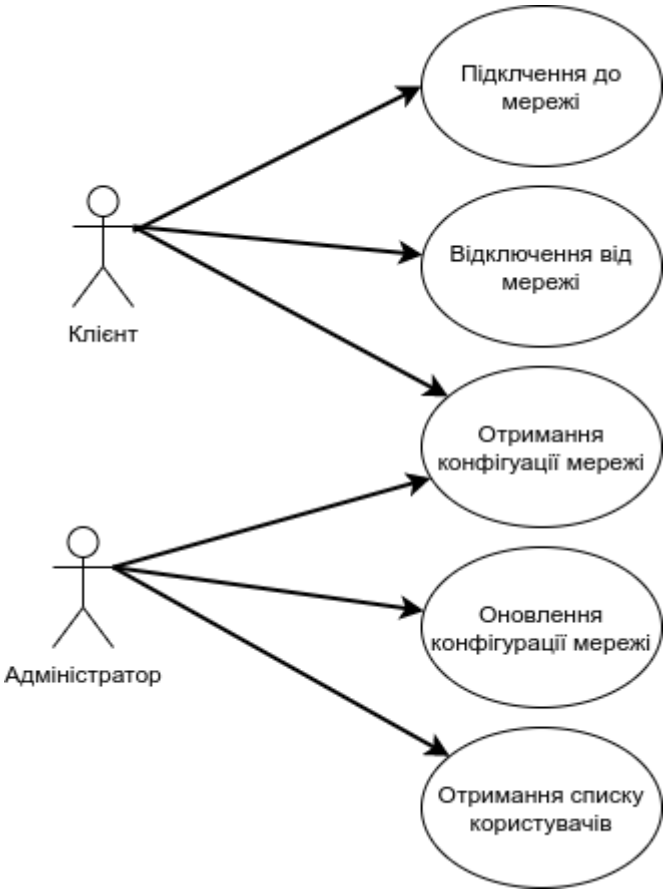

Рисунок 1.6 — Діаграма прецедентів

Розроблений DHCP-сервер повинен в повній мірі забезпечувати виконання функцій по допомозі в підключенні клієнтів до мережі, а також взаємодію адміністраторів для змінення наявних налаштувань, при потребі. Тому на основі предметної області та основних обов’язків актора можна сформулювати ключові варіанти використання. Опис яких наведено в таблиці 1.1.

Таблиця 1.1 - Таблиця варіантів використання

Актор	**Найменування**	**Формулювання**
Клієнт	Підключення до мережі	Дозволяє отримати IP-адресу і маску мережі
	Отримання конфігурації мережі	Дозволяє отримати інформацію про мережу, яка цікавить клієнта
Адміністратор	Отримання конфігурації DHCP-сервера	Дозволяє отримати інформацію про конфігурацію даного DHCP-сервера
	Зміна параметрів DHCP-сервера	Дозволяє змінити налаштування DHCP-сервера
	Отримання списку клієнтів	Дозволяє отримати інформацію про клієнтів, які користуватися розробленим DHCP-сервером

В таблиці 1.1 наведено основі варіанти використання, які потрібно буде реалізувати.

1.6 Вибір засобів розробки

Реалізація спроектованого DHCP-сервера відбувалася такими засобами, як Go, React. Розглянемо їх детальніше.

1.6.1 Вибір Go

Go — це сучасна компільована мова програмування розроблена в Google. Публічно було представлено в листопаді 2009 року, а версія 1.0 випущена в березні 2012 р. [12]. Основною причиною розробки нової мови програмування, було бажання підвищити продуктивність роботи з багатоядерними процесорами, великою кодовою базою та мережевим обладнанням. Під час розробки розробники поєднали простоту написання коду і ефективність [13].

Вони взяли всі переваги з інших мов, а саме:

- З C/C++ взяли статичну типізацію та ефективність виконання.
- Юзабіліті та читабельність вихідного коду, на рівні з Python.
- Висока продуктивність у роботі з мережевим обладнанням.

Дана мова програмування дотримується простої філософії: “Одна проблема повинна мати одне рішення”. Завдяки такому підходу Go є інтуїтивно зрозумілою мовою, що демонструє високу ефективність обробки даних [14].

Оскільки Go це мова з відкритим кодом, то будь-хто може внести свій внесок в її розвиток і використовувати відповідно до своїх вимог; однак найкращим рішенням є програмування на стороні сервера, програмування комп’ютерних мереж і веб-розробка. Завдяки своїм паралельним функціям обробка декількох запитів та одночасна взаємодія з багатьма користувачами обробляються швидше і з меншими витратами процесорного часу [15].

Згідно із результатами опитування розробників Go 2019, топ сфери використання Go - це веб розробка, бази даних та мережі [16]. З результатами опитування можна ознайомитися на рисунку 1.7.

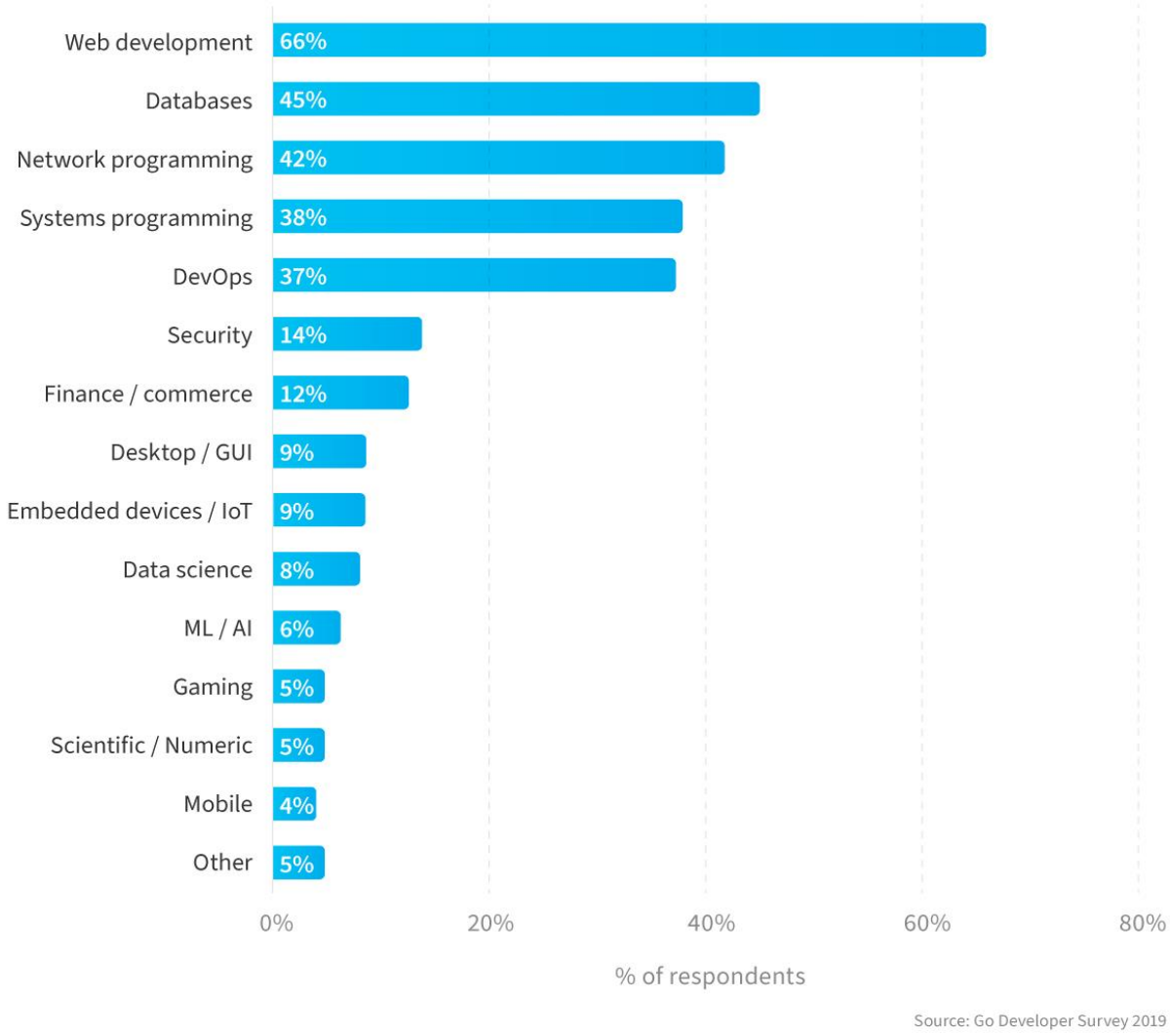

Рисунок 1.7 — Дослідження сфер використання Go

Також до переваг можна віднести наступні пункти:

- Мова програмування може похвалитися своєю швидкістю компіляції. Це досягається завдяки перетворенню коду безпосередньо в команди машинного рівня.
- Має потужний механізм тестування, який може повідомити розробника, чи працює їх код перед компіляцією. Ця функціональність дозволяє економити час на пошуку помилок.
- Компілятор може запускати програми на машинах з різними операційними системами, використовуючи той самий компілятор.

1.6.2 Вибір React

Facebook в 2011 році створив React для власного використання [17]. Як відомо, Facebook є одним з найбільших веб-сайтів соціальних мереж у світі.

В 2012 році почали використовувати в Instagram, який є дочірньою компанією Facebook.

З розвитком цієї бібліотеки Facebook зробив її відкритою в 2013 році. Спільнота розробників спочатку відхилила її, Оскільки запропонований підхід був не зрозумілий більші кількості розробників. Але все більше людей експериментували з цим, вони почали застосовувати компонентно-орієнтований підхід для відокремлення конкретних завдань [18].

React надзвичайно гнучкий. Його можна використовувати на різноманітних платформах для створення якісних користувацьких інтерфейсів [19]. Оскільки React — це бібліотека, а не фреймворк. І саме бібліотечний підхід дозволив перетворити React на чудовий інструмент.

React був створений з метою: створювати компоненти для веб-додатків. React компонентом може бути що небудь у веб-додатку наприклад: кнопки, текст, зображення, списки та інше.

1.7 Висновок до першого розділу

В першому розділі кваліфікаційної роботи було розглянуто історію виникнення комп’ютерних мереж, їх розвиток. Було з'ясовано, які протоколи використовувалися для автоматичного конфігурування мережевих інтерфейсів користувачів, та яку роль вони відіграють. Розглянуто принципи роботи DHCP протоколу, і його фази. В цьому розділі також були обрані засоби розробки.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ DHCP-СЕРВЕРА

2.1 Реалізація DHCP-сервера

Дане програмне забезпечення складається з трьох основних частин: самого DHCP-сервера, WebApi керування DHCP-сервером і графічного інтерфейсу. Графічно архітектуру можна побачити на рисунку 2.1.

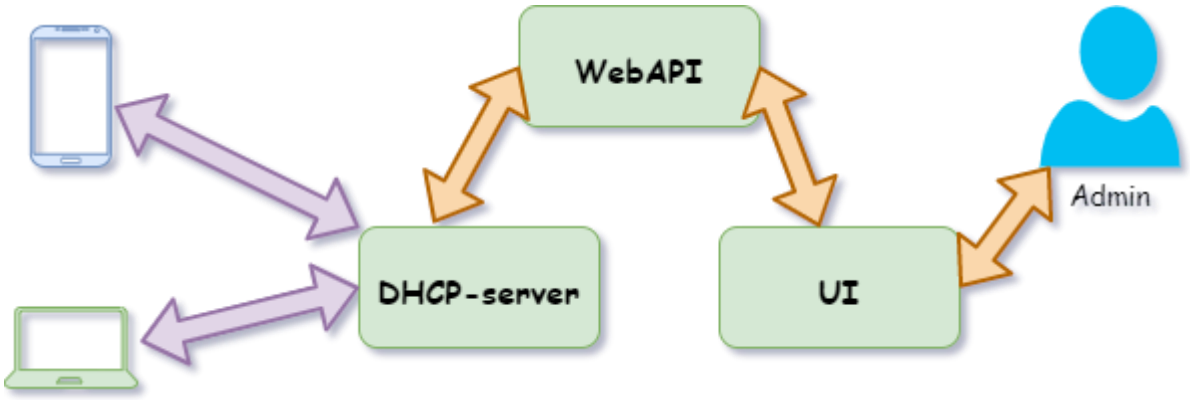

Рисунок 2.1 - Архітектура

Відповідно до рисунка вище, адміністратор використовує даний продукт за допомогою графічного інтерфейсу, який в свою чергу використовуючи WebApi може взаємодіяти з DHCP-сервером.

Оскільки DHCP-сервер повинен працювати з мережею, розробку варто розпочати з лістенера який, дозволить відкрити UDP порт, а також приймати пакети які будуть приходити. Для цього створимо функцію ListenAndServe з реалізацією можна ознайомитися на лістингу 2.1.

Лістинг 2.1 — Реалізація функції ListenAndServe

```
func ListenAndServe(handler Handler) error {
     listener, err := net.ListenPacket("udp", ":67")

     if err != nil {
          return err
     }
```

```
    defer listener.Close()

    return Serve(listener, handler)
}
```

За допомогою функції ListenPacket з пакету net відкрили порт 67 на протоколі UDP, після чого перевірили наявність помилок при відкриті порта. Якщо в змінні err нічого не знаходиться ми можемо перейти до обробки пакетів які будуть надходити на наш інтерфейс. Після чого інтерфейс ServeConn для роботи з з'єднанням. Оскільки нам потрібно читати і писати байти в з'єднання, то інтерфейс буде мати наступний вигляд.

Лістинг 2.2 – Реалізація інтерфейсу ServeConn

```
type ServeConn interface {
    ReadFrom(Packet) (Size, net.Addr, error)
    WriteTo(Packet, net.Addr) (Size, err error)
}
```

Завдяки раніше створеному інтерфейсу, ми забезпечили що метод ReadFrom може приймати на вхід лише слайси байтів, а повертати їхню кількість, адресу звідки прийшли, і якщо сталася якась помилка, то і її. Метод WriteTo дозволить нам відправляти пакети у вигляді байтів.

Для обробки всіх з'єднань створимо функцію ServeDHCP яка на вхід приймає лістенера створеного раніше, а також handler який дозволить обробляти запити по DHCP протоколі. В функції створюється буфер з байтів в який будуть записувати отримані пакети, після чого буфер перевіряється на валідність пакету, а також отримується опції з пакету а також тип dhcp повідомлення, які можуть бути DHCPDISCOVER, DHCPREQUEST, DHCPRELEASE і DHCPDECLINE останні два сигналізують про можливість звільнення оренди з IP адреси. З фрагментом функції ServeDHCP можна ознайомитися на лістингу 2.3. Графічно роботу цієї функції можна побачити на рисунку 2.2.

Лістинг 2.3 — Фрагмент функції ServeDHCP

```
if res := handler.ServeDHCP(req, reqType, options); res != nil {
```

```
        ip, port, err := net.SplitHostPort(addr.String())
        if err != nil {
            return err
        }

        if net.ParseIP(ip).Equal(net.IPv4zero) || req.Broadcast() {
            port, _ := strconv.Atoi(port)
            addr = &net.UDPAddr{
                IP:   net.IPv4bcast,
                Port: port,
            }
        }
        if _, err := conn.WriteTo(res, addr); err != nil {
            return err
        }
}
```

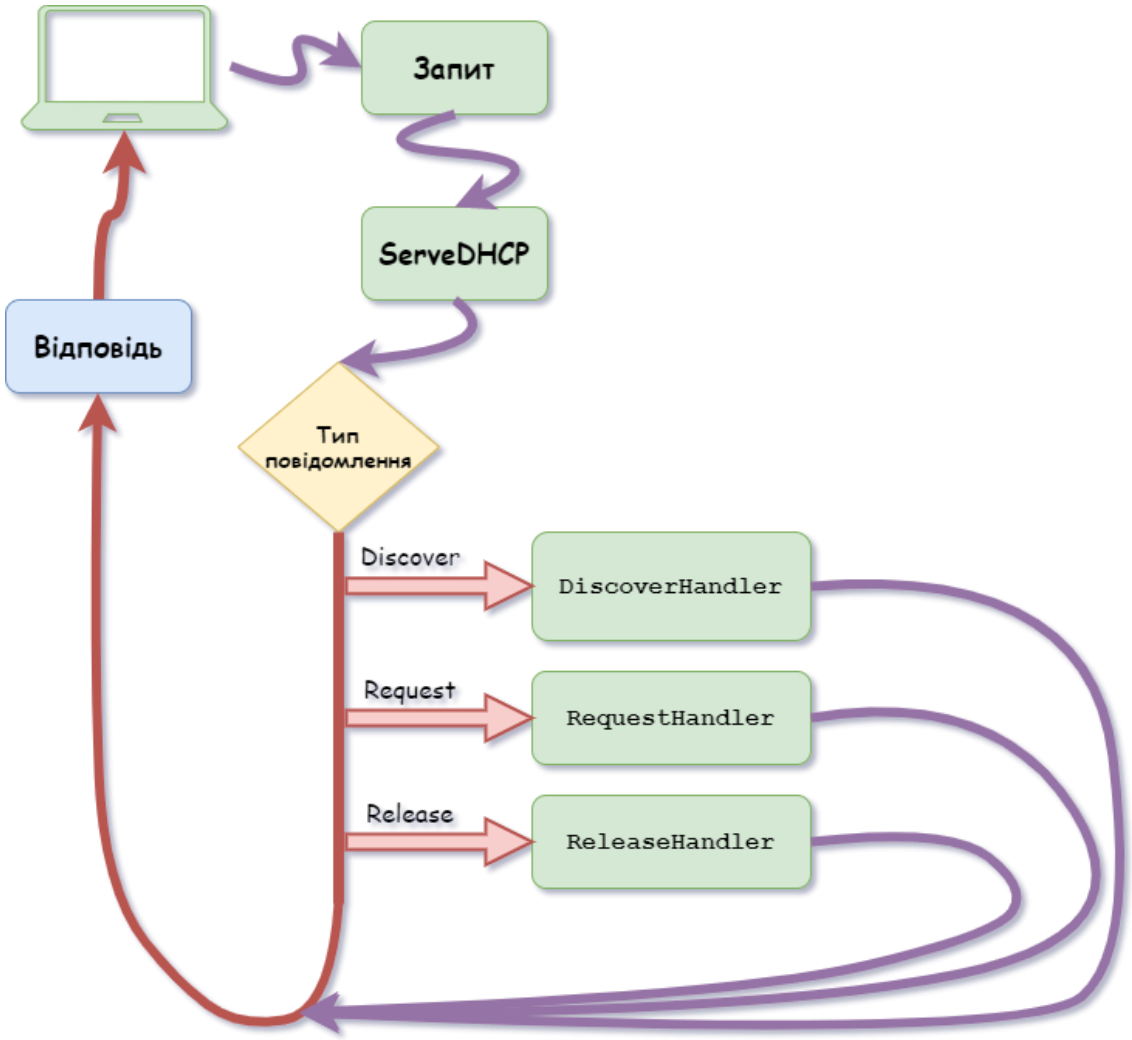

Рисунок 2.2 – метод ServeDHCP

В цьому фрагменті отримується результат метода ServeDHCP, за умови що результат не nil (аналогія з undefined з іншими мовами програмування)

отримується IP адреса клієнта і перевіряється її приналежність до стандарту IPv4, коли клієнт хоче продовжити свою оренду, Після чого клієнту відправляється пакет.

Метод ServeDHCP це метод який реалізує інтерфейс Handler з реалізацію цього інтерфейсу можна ознайомитися нижче.

```
type Handler interface {
     ServeDHCP(Packet, MessageType, Options) Packet
}
```

Як бачимо із сигнатури інтерфейсу, інтерфейс Handler містить лише один метод ServeDHCP, який приймає на вхід Packet який що є слайсом байтів, MessageType який я байтом а також опції з типом Options можна ознайомитися нижче на лістингу 2.4.

Лістинг 2.4 — Визначення типу Options

```
type OptionCode byte
type Option struct {
     Code  OptionCode
     Value []byte
}
```

Метод ServeDHCP дозволяє перенаправити процес виконання до потрібного обробника на основі типу повідомлення, яке прийшло. Оскільки для різних типів повідомлень очікується поведінка, котра може суттєво відрізнятися. Так як при запиті типу DHCPDISCOVER необхідно зарезервувати адресу для подальшої оренди, а при DHCPREQUEST запропонувати клієнту конфігурацію. З реалізацією метода ServeDHCP можна ознайомитися на лістингу 2.5.

Лістинг 2.5 — Реалізація метода ServeDHCP

```
func (dhcpHandler *DHCPHandler) ServeDHCP(packet Packet,
messageType MessageType, options Options) Packet {
     switch messageType {
     case Discover:
          return dhcpHandler.DiscoverHandler(packet, options)
     case Request:
          return dhcpHandler.RequestHandler(packet, options)
```

```
     case Release, Decline:
          return dhcpHandler.ReleaseHandler()
     }
     return nil
}
```

Як можемо спостерігати з лістингу 2.5 за допомогою оператора switch відбувається розгалуження для різних типів пакетів, які повинні обробити запит. При отриманні пакету з типом Discover викликається метод DiscoverHandler з реалізацією метода можна ознайомитися на лістингу 2.6.

Лістинг 2.6 - Реалізація DiscoverHandler

```
func (dhcpHandler *DHCPHandler) DiscoverHandler(packet Packet,
options Options) (d Packet) {
     free, nic := -1, dhcpHandler.CHAddr().String()
     for i, v := range dhcpHandler.leases {
          if v.Nic == nic {
               free = i
               goto reply
          }
     }
     if free = dhcpHandler.freeLease(); free == -1 {
          return
     }
reply:
     return ReplyPacket(packet, Offer, dhcpHandler.ip,
IPAdd(dhcpHandler.start, free), dhcpHandler.leaseDuration,

     dhcpHandler.options.SelectOrderOrAll(options[OptionParameterRe
questList]))
}
```

В цьому хендлері спочатку дістається апаратна адреса клієнта за допомогою метода CHAddr, після чого вона перевіряється на наявність в діючих орендах. Якщо оренда раніше зроблена, то виконується перестрибування до мітки reply за допомогою оператора goto, цим DHCP-сервер дозволяє використовувати попередню IP-адресу клієнтом. Але якщо це новий пристрій, який ніколи не користувався цим DHCP-сервером, то викликається метод freeLease, який дозволяє орендувати доступну IP-адресу в пулі адрес. З реалізацією метода freeLease можна ознайомитися на лістингу 2.7.

Лістинг 2.7 — Реалізація метода freeLease

```
func (dhcpHandler *DHCPHandler) freeLease() int {
     timeNow := time.Now()
     seed := rand.Intn(s.leaseRange)
     for _, v := range [][]int{{seed, s.leaseRange}, {0, seed}} {
          for i := v[0]; i < v[1]; i++ {
               if lease, ok := dhcpHandler.leases[i]; !ok ||
lease.Expiry.Before(timeNow) {
                    return i
               }
          }
     }
     return -1
}
```

В цьому метода спочатку отримується час в момент виклику функції Now пакету time, після чого генерується ціле невід'ємне число в межах встановлених лімітом на видачу IP-адрес і резервується одна із адрес для подальшого використання.

По аналогії з повідомленнями типу Discover повідомлення з типом Request обробляються в хендлері RequestHandler з реалізацією якого можна ознайомитися на лістингу 2.8.

Лістинг 2.8 — Реалізація метода RequestHandler

```
func (dhcpHandler *DHCPHandler) RequestHandler(packet Packet,
options Options) Packet {
     if server, ok := options[OptionServerIdentifier]; ok &&
!net.IP(server).Equal(dhcpHandler.ip) {
          return nil
     }
     if reqIP := net.IP(options[OptionRequestedIPAddress]);
len(reqIP) == 4 {
          if leaseNum := IPRange(dhcpHandler.start, reqIP) - 1;
leaseNum >= 0 && leaseNum < dhcpHandler.leaseRange {
               if lease, exists := dhcpHandler.leases[leaseNum];
!exists || lease.Nic == dhcpHandler.CHAddr().String() {
                    IP := net.IP(options[OptionRequestedIPAddress])

                    dhcpHandler.leases[leaseNum] = lease{
                         Nic:    dhc.CHAddr().String(),
                         Expiry: time.Now().Add(s.leaseDuration),
                         IP:     IP.String(),
                    }
```

```
                    return ReplyPacket(packet, ACK, dhcpHandler.ip,
IP, dhcpHandler.leaseDuration,

     dhcpHandler.options.SelectOrderOrAll(options[OptionParameterRe
questList]))
                }
           }
     }
     return ReplyPacket(packet, NAK, dhcpHandler.ip, nil, 0, nil)
}
```

В цьому метода спочатку перевіряється, чи пакет насправді адресований цьому DHCP-серверу, після чого розпочинається процес виконання оренди, в хеш-таблицю Leases вноситься фізична адреса клієнта, а також час завершення оренди. Якщо клієнт бажає отримати IP-адресу, котра не належить пулу адрес, або вже зайнята іншим клієнтом, то у відповідь цей клієнт отримує відповідь пакет DHCPNAK, котрий заставить розпочати процес перепідключення заново.

Останнім хендлером в ServeDHCP є ReleaseHandler, який скасовує діючу оренду передчасно, це може статися, коли існує інший клієнт з запропонованою конфігурацією, або клієнт сам відмовляється від наявної IP-адреси. З реалізацією цього хендлера можна ознайомитися на лістингу 2.9.

Лістинг 2.9 — Реалізація ReleaseHandler

```
func (dhcpHandler *DHCPHandler) ReleaseHandler(packet Packet) {
     nic := packet.CHAddr().String()

     for i, v := range dhcpHandler.leases {
          if v.Nic == nic {
               delete(dhcpHandler.leases, i)
               return
          }
     }
}
```

В цьому хендлері дістається фізична адреса клієнта, котрий хоче відкликати раніше надану йому IP-адресу. Після чого за допомогою for range здійснюється прохід по всій хеш-таблиці оренд, з метою знайти оренду видану

на потрібну фізичну адресу. Коли потрібна оренда знаходиться, то вона видаляється з хеш-таблиці за допомогою функції delete.

2.2 Розробка модуля для керування DHCP-сервером

Згідно з варіантами використання, які були визначенні у попередньому розділі, одним з варіантів використання DHCP-сервера є можливість змінювати налаштування і моніторити користувачів. Для цього потрібно розробити модуль api, який розробляється на основі інтерфейсу WebApi. WebApi — це тип прикладного програмного інтерфейсу, який використовує архітектуру та мережеві протоколи Web, одним з яких є HTTP, для зв'язку між програмами, які розташовані на окремих пристроях в мережі [20].

WebApi може приймати різні форми, такі як виклик віддалених процедур, передача поточного стану, та інші [21]. Особливістю, яка відрізняє WebApi від традиційної взаємодії в Інтернеті, є те, що користувач інтерфейсів - це не людина, яка безпосередньо користується веб-браузером, а програма.

У реалізації WebApi найбільш поширеним стилем архітектури є REST, де виклик служби інтерфейсу з параметрами кодується в унікальний ідентифікатор ресурсів, а відповідь зазвичай кодується за допомогою JSON, хоча допускаються і інші варіанти наприклад звичайний текст [22].

На стороні сервера важливими аспектами є кінцеві точки, оскільки вони вказують, де лежать ресурси, до яких програмне забезпечення намагається отримати доступ.

Для розробки WebApi в цьому проекті було обрано стандартну бібліотеку для Go net.http [23]. Спочатку нам необхідно створити об'єкту Router, який дозволить маршрутизувати всі запити, котрі будуть надходити до сервера з api. Наступним кроком буде створення кінцевих точок, які будуть знаходитися на різних унікальних ідентифікаторів ресурсів, що дозволяє організувати спілкування. З створенням кінцевих точок можна ознайомитися на лістингу 2.10.

Лістинг 2.10 — Створення кінцевих точок

```
// auth
router.Use(authMiddleware)
router.HandleFunc("/api/auth/login", login).Methods("POST")
router.HandleFunc("/api/auth/logout", logout).Methods("DELETE")
router.HandleFunc("/api/auth/user", user)

// control dhcp
router.HandleFunc("/api/config", getConfig).Methods("GET")
router.HandleFunc("/api/config", setConfig).Methods("POST")
router.HandleFunc("/api/leases", getLeases).Methods("GET")
router.HandleFunc("/api/shutdown", shutdownDHCP(mDHCP))
router.HandleFunc(("/api/start"), startDHCP(mDHCP))
```

Графічно маршрути можна зобразити так, дивитися рисунок 2.3.

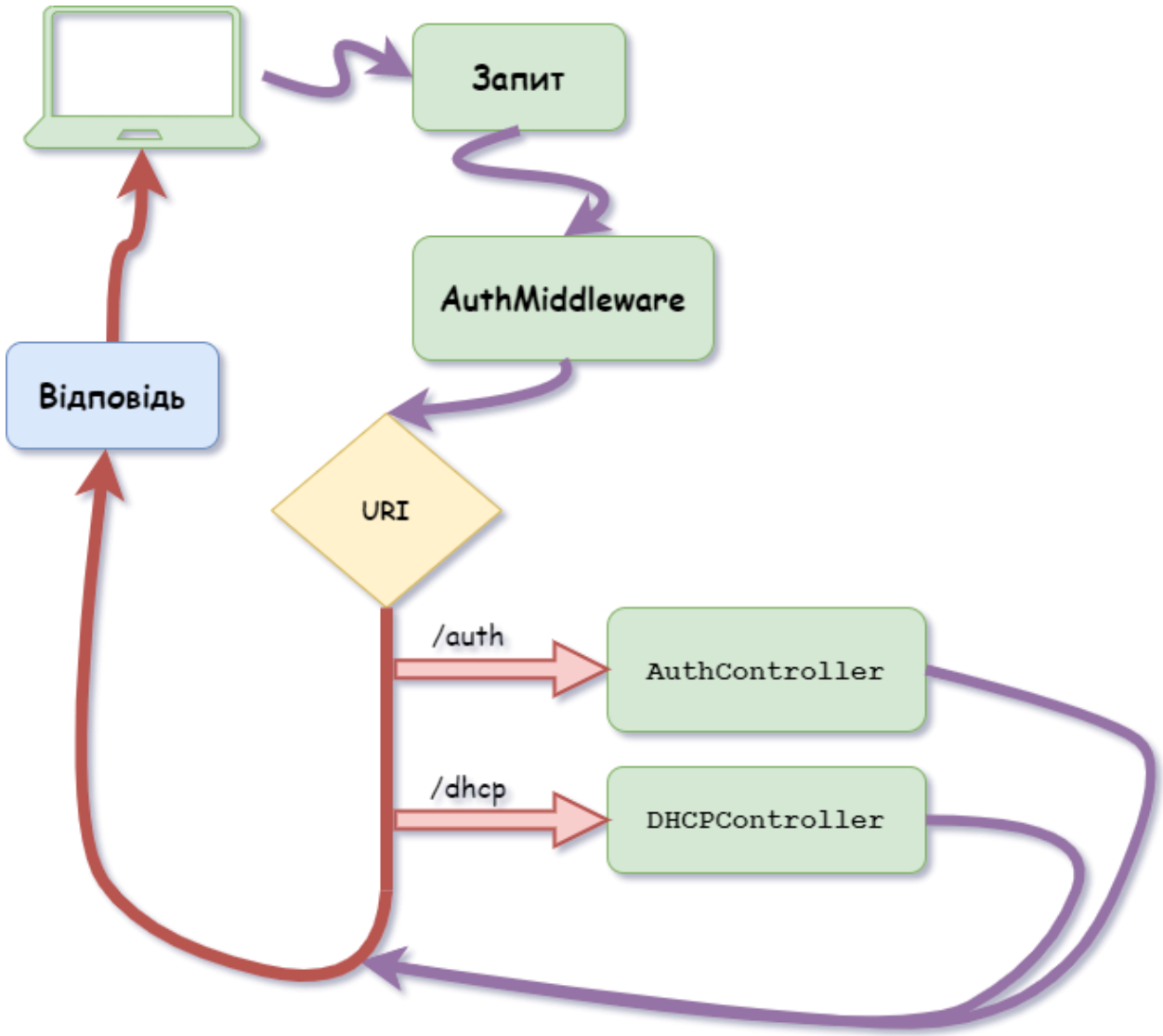

Рисунок 2.3 – Життєвий цикл запиту в WebAPI

Кінцеві точки можна умовно розділити на дві групи, ті які відповідають за авторизацію сюди можна віднести: login, logout, user - AuthController, а також ті що контролюють стан dhcp сервера відповідно сюди належать getConfig, setConfig, getLeases, shutdownDHCP і startDHCP - DHCPController. Авторизація потрібна, щоб лише адміністратор міг попасти в панель керування. Кінцеві точки створюються за допомогою метода HandlerFunc який першим аргументом шлях, другим - функцію, яка повинна обробити запит за вказаним раніше шляхом. Розглянемо обробника на прикладі функції login реалізація якої на лістингу 2.11.

Лістинг 2.11 — Реалізація обробника login

```
func login(w http.ResponseWriter, r *http.Request) {
     var ctx, cancel = context.WithTimeout(context.Background(),
100*time.Second)
     var user models.User
     var u models.User

     err := json.NewDecoder(r.Body).Decode(&user)
     if err != nil {
          w.WriteHeader(http.StatusBadRequest)
          w.Write([]byte(err.Error()))
     }
     err = db.UsersCollection().FindOne(ctx, bson.M{
          "username": user.Username,
     }).Decode(&foundUser)
     defer cancel()
     if err != nil {
          w.WriteHeader(http.StatusBadRequest)
          w.Write([]byte(err.Error()))
          return
     }
     passwordValid, err := utils.VerifyPassword(user.Password,
u.Password)
     defer cancel()

     if !passwordValid {
          w.Write([]byte(err.Error()))
          return
     }

     bytes, _ := json.Marshal(u)

     w.Write([]byte(bytes))
}
```

Функція приймає два аргументи: перший це інтерфейс http.ResponseWriter в який записується відповідь, яку потрібно повернути користувачеві, другий — це вказівник на запит, обробкою якого займаємося в конкретний момент часу. В обробнику спочатку створюється контекст, який використовується при отриманні користувача з бази даних. Створення контексту в результаті виконання функції WithTimeout, де другим аргументом вказується часовий ліміт виконання. А також створюється два екземпляри типу User. У перший записується інформація з тіла обробляючого запиту, другий для отримання користувача з бази даних. Після чого перевіряється пароль на правильність.

Інші обробники побудовані за схожою стратегією, обробник setConfig отримує в тілі запиту нову конфігурацію, котру записує у файл, з якого під час запуску DHCP-сервер вичитує власні налаштування.

2.3 Розробка графічного інтерфейсу

Графічний інтерфейс — це невід’ємна частина будь-якого програмного забезпечення [25]. Саме графічний інтерфейс дозволяє зручніше працювати з раніше розробленим WebApi. Для розробки графічного інтерфейсу ми раніше вибрали бібліотеку React, причини використання були описані в першому розділі кваліфікаційної роботи. Для початкової генерації проекту використаємо утиліту create-react-app, з процесом генерування проекту можна ознайомитися на рисунку 2.4.

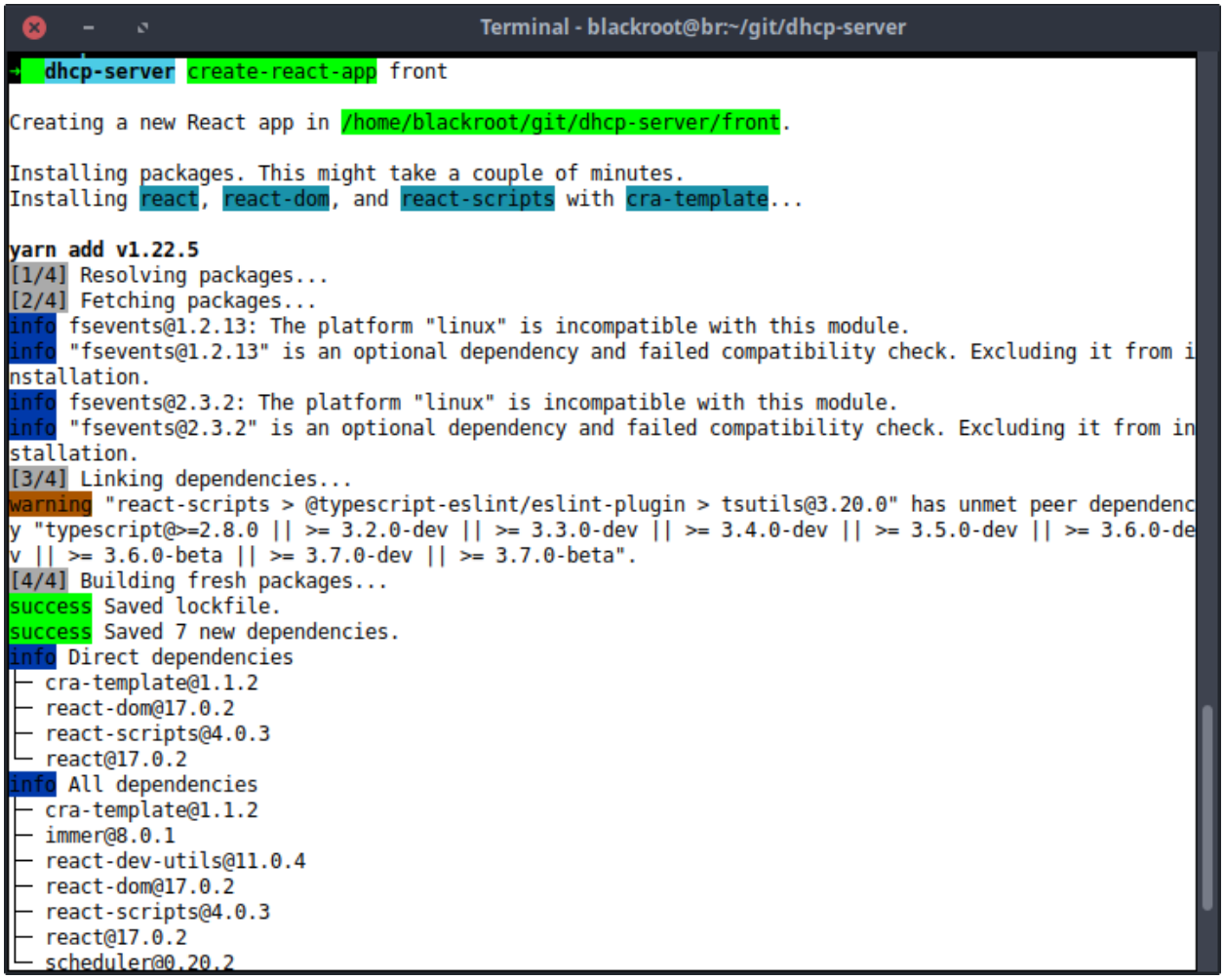

Рисунок 2.4 — Генерація проекту для графічного інтерфейсу

Наступними етапами буде створення сторінок, які дозволяють користувачу залогінитися в розробленому додатку, переглянути конфігурацію DHCP-сервера, і за потреби її змінити, а також переглянути список діючих оренд IP-адрес.

Розглянемо створення сторінки на прикладі розробки сторінки, на якій зображується таблиця з виданими IP-адресами, їх фізичними адресами і датою закінчення оренди. Реалізацію цієї сторінки можна побачити на лістингу 2.12.

Лістинг 2.12 — Реалізація сторінки з таблицею оренд

```
const Home = () => {
  const [clients, setClients] = useState(null)
  useEffect(async () => {
    const buffer = []
    const clients = await fetcher('/api/leases')
    for (const client of Object.values(clients)) {
```

```
      buffer.push(client)
    }
    setClients(buffer)
  }, [])
  return (
    <TableContainer component={Paper}>
      <Table className={classes.table}>
        <TableHead>
          <TableRow>
            <TableCell> IP </TableCell>
            <TableCell> Фізична адреса </TableCell>
            <TableCell> Закінчення оренди </TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {clients.map(client => (
            <TableRow key={client.nic}>
              <TableCell>{client.ip}</TableCell>
              <TableCell>{client.nic}</TableCell>
              <TableCell>{client.expiry}</TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
  )
}
```

Спочатку створюється стейт, в якому зберігається інформація про всі оренди, за допомогою функції useState, параметром якої передається значення для початкової ініціалізації. Далі за допомогою хука useEffect викликається асинхронна функція, в якій відбувається запит до WebApi після чого отриманні дані візуалізуються в браузері користувача. З виглядом цієї таблиці можна ознайомитися на рисунку 2.5.

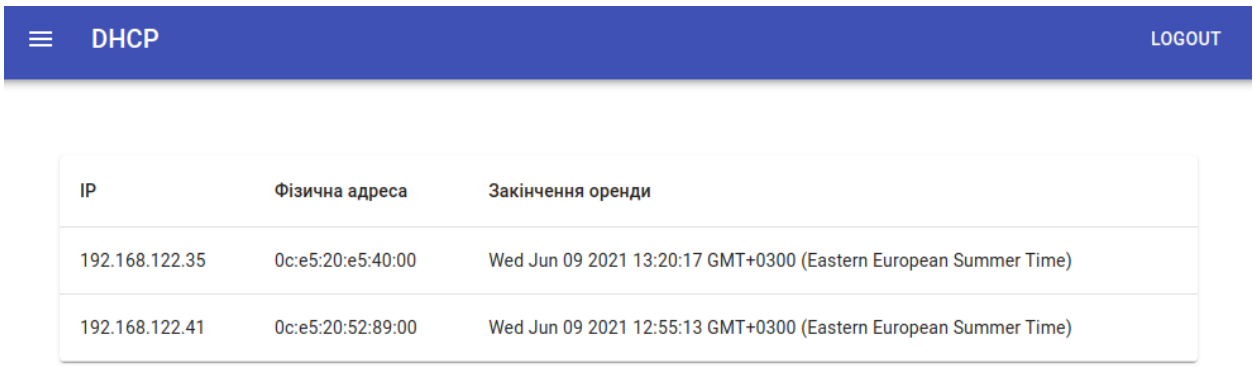

Рисунок 2.5 — Вигляд таблиці оренд

Реалізований швидше WebApi вимагає від користувача авторизації, тому слід добавити авторизацію в користувацьких інтерфейс. Для цього потрібно створити провайдер — UserProvider. Провайдер — це об’єкт, з якого дочірні компоненти можуть діставати дані. Даними можуть бути об’єкти, функції. Методи які містяться в провайдері зазвичай змінюють стан даних, з якими розміщуються вони. З реалізацією UserProvider можна ознайомитися на лістингу 2.13.

Лістинг 2.13 — Реалізація UserProvider

```
const UserProvider = ({ children }) => {
  const [user, setUser] = useState(undefined)
  const history = useHistory()
  const [methods] = useState({
    login ({ username, password }) {
      return fetcher('/api/auth/login', {
        method: 'POST',
        body: JSON.stringify({ username, password })
      }).then(() => methods.sync())
    },
    sync () {
      return fetcher('/api/auth/user')
        .then((user) => setUser(user))
        .catch(() => {
          setUser(null)
          history.push('/login')
        })
    },
    logout () {
      return fetcher('/api/auth/logout', { method: 'DELETE' })
```

```
        .then(() => setUser(null))
        .then(() => methods.sync())
    }
  })

  useEffect(() => {
    methods.sync()
  }, [methods])

  return (
    <UserContext.Provider value={[user, methods]}>
      {children}
    </UserContext.Provider>
  )
}
```

В провайдері спочатку створюється стейт для збереження користувача, а також функція для його зміни за допомогою функції useState. За аналогією створюємо методи для авторизації серед яких логін, синхронізація стану і вихід. За допомогою метода login виконується POST запит до WebApi, в тілі якого знаходяться авторизаційні дані користувача. Як можемо спостерігати в провайдер передаємо масив з двох елементів, де перший — інформація про користувача, другий — авторизаційні методи. Метод синхронізації при кожному оновлені сторінки, переході на іншу сторінку дозволяє перевірити, чи дійсно даний користувач залогінений.

2.4 Тестування розробленого DHCP-сервера

Тестування розробленого DHCP-сервера розпочнемо з його компіляції. Це можна зробити за допомогою команди go build. Процес компіляції можна побачити на рисунку 2.6.

```
→  back git:(master) x go build
→  back git:(master) x ls
api  back  config  db  dhcp  go.mod  go.sum  helper  main.go  models  utils
→  back git:(master) x
```

Рисунок 2.6 — процес компіляції DHCP-сервера

На рисунку ми бачимо, що процес пройшов без помилок і поточному каталозі з'явився файл back, який має права на виконання. Після чого спробуємо запустити цей бінарний файл. Процес запуску і перевірка відкритості UDP порта зображені на рисунку 2.7.

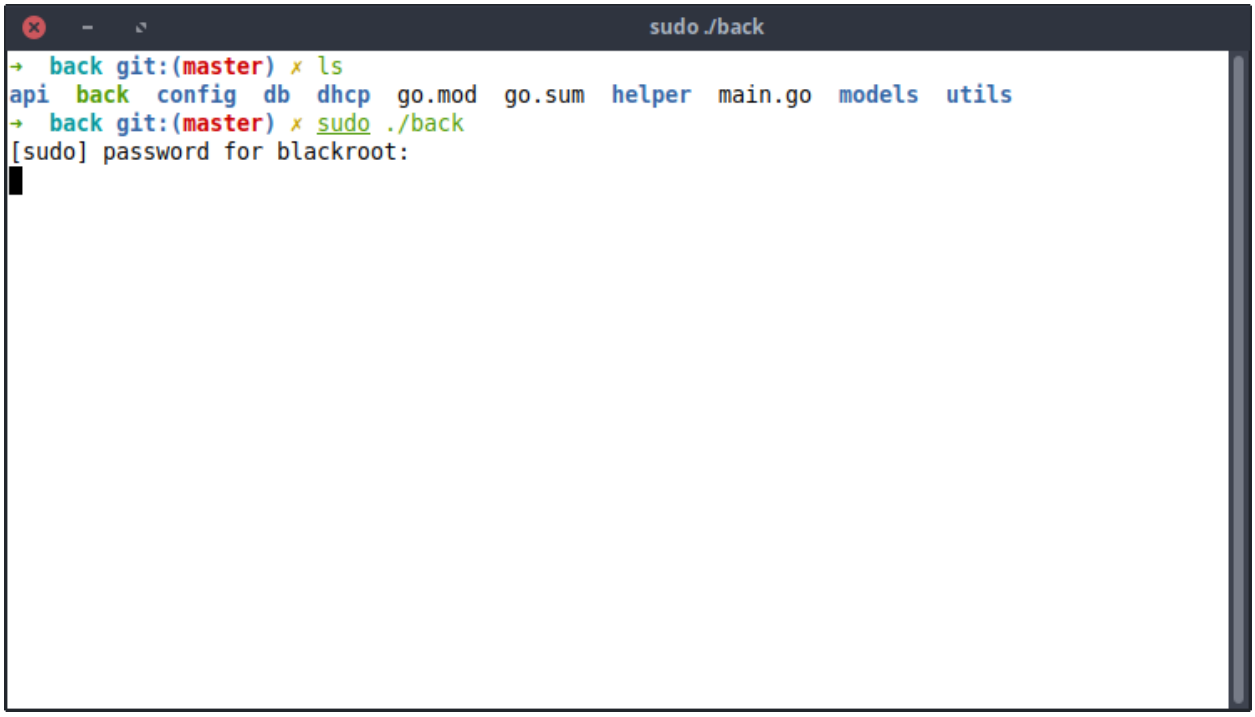

Рисунок 2.7 — Запуск DHCP-сервера

Наступним кроком створимо віртуальну мережу, яка буде складатися з двох комп'ютерів, хаба, котрий буде з'єднувати ці комп'ютери. Для моделювання скористаюся емулятором GNS3. GNS3 — це емулятор програмного забезпечення мережі, випущений в 2008 році. Він дозволяє поєднувати віртуальні та реальні пристрої, що використовуються для імітації мереж. З топологією спроектованої мережі можна ознайомитися на рисунку 2.8.

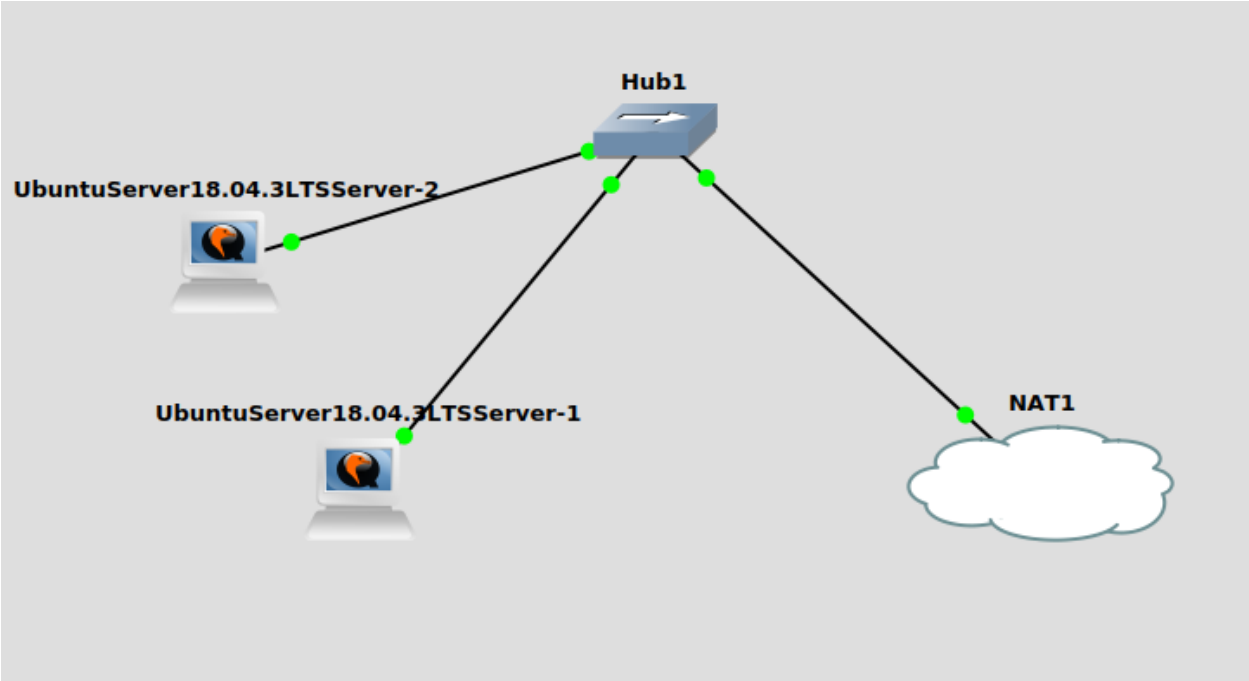

Рисунок 2.8 — Топологія змодельованої мережі

Після чого ми можемо підключитися до кожного з створених комп’ютерів. На рисунку 2.9, ми можемо спостерігати відповідність конфігурації на клієнті і у веб-інтерфейсі для першого комп’ютера.

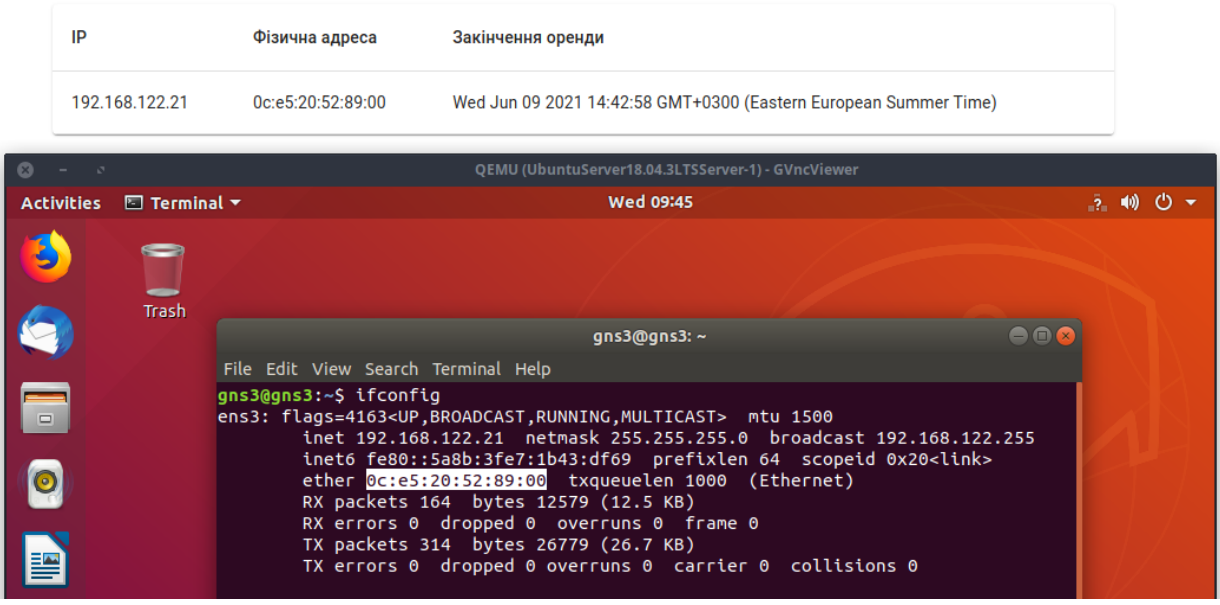

Рисунок 2.9 — Конфігурація інтерфейса ens3 на першому комп’ютері

За допомогою утиліти ifconfig дізналися IP-адресу, яку видали для цього клієнта і його MAC-адресу. Можемо спостерігати що таблична інформація збігається з фактичною. Тепер повторимо це саме для другого комп'ютера. На рисунку 2.10, бачимо відповідність налаштувань в графічному інтерфейсі і другому комп'ютері.

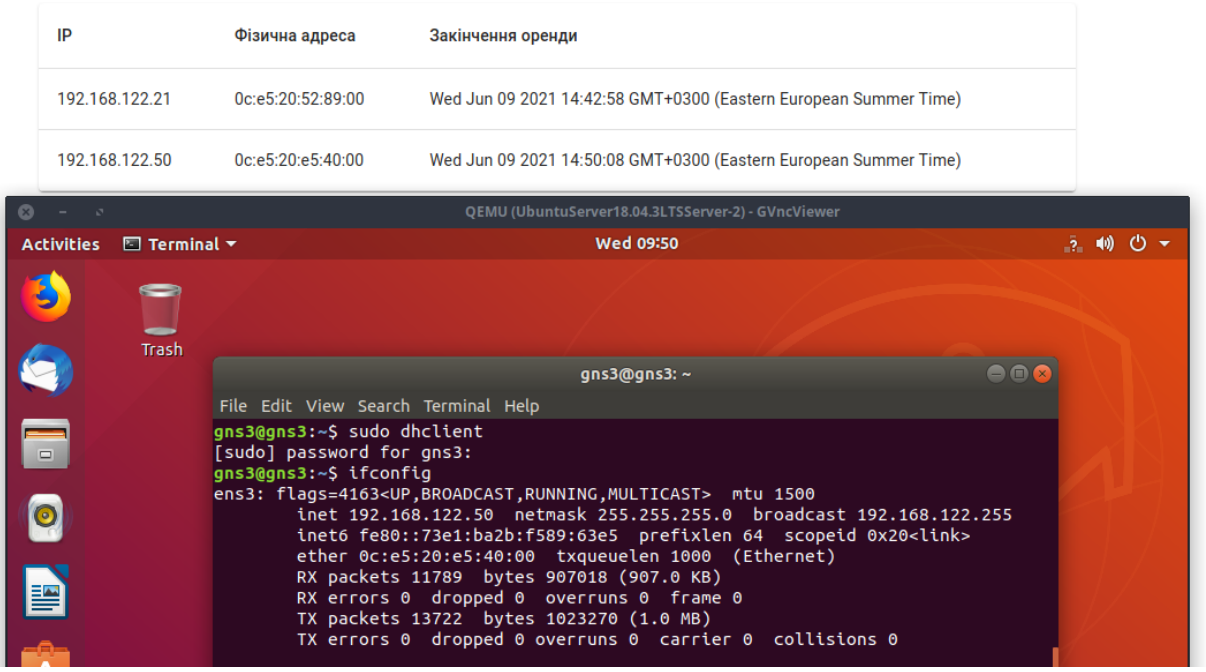

Рисунок 2.10 — Конфігурація інтерфейса ens3 на другому комп'ютері

Другий клієнт також налаштував, свій інтерфейс, запропонованим DHCP-сервером способом.

Далі змінимо маску підмережі з 24 на 25, тобто на 255.255.255.128, що дозволить початкову мережу розбити на дві підмережі, останній хост при 25-тій масці повинен мати IP-адресу, що закінчується на 127 і загальна кількість в цій підмережі може становити лише 126 пристроїв. Нові параметри DHCP-сервера можна бачити на рисунку 2.11.

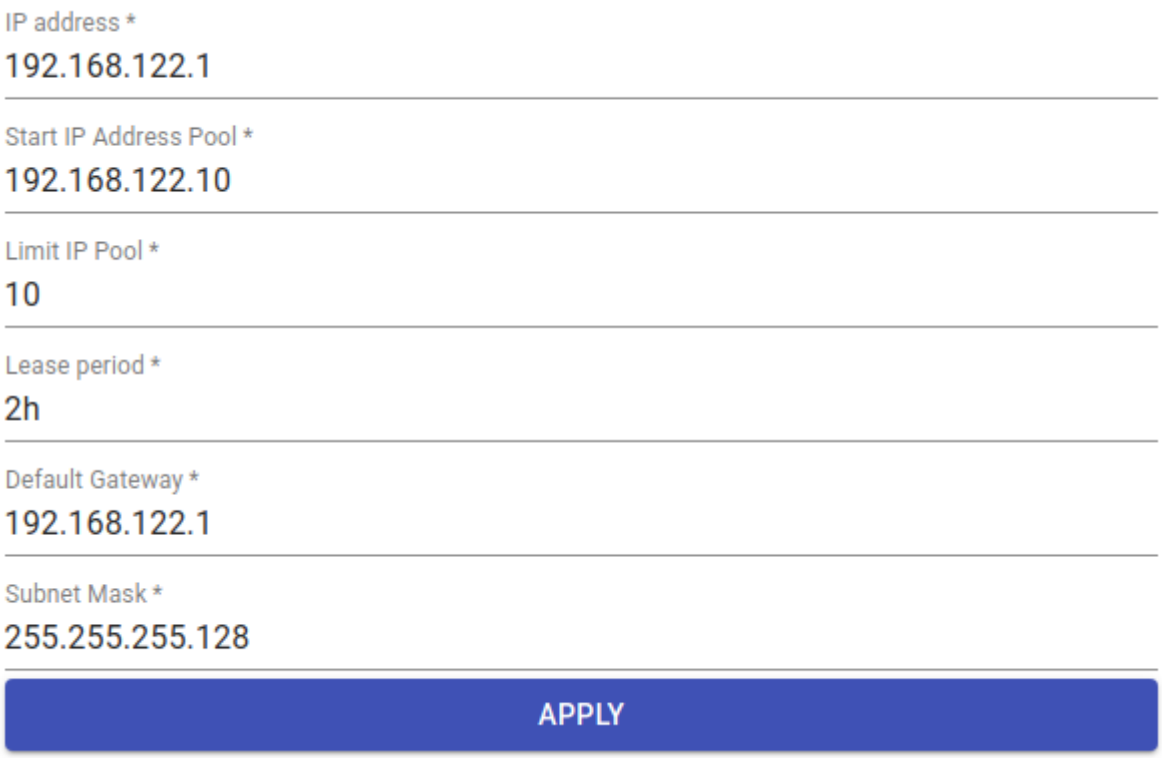

Рисунок 2.11 — Налаштування DHCP-сервера

Після чого натискаємо на кнопку Apply для застосування налаштувань. В процесі DHCP-сервер перезапуститься. Після чого оновимо налаштування на одному із комп'ютерів за допомогою утиліти dhclient.

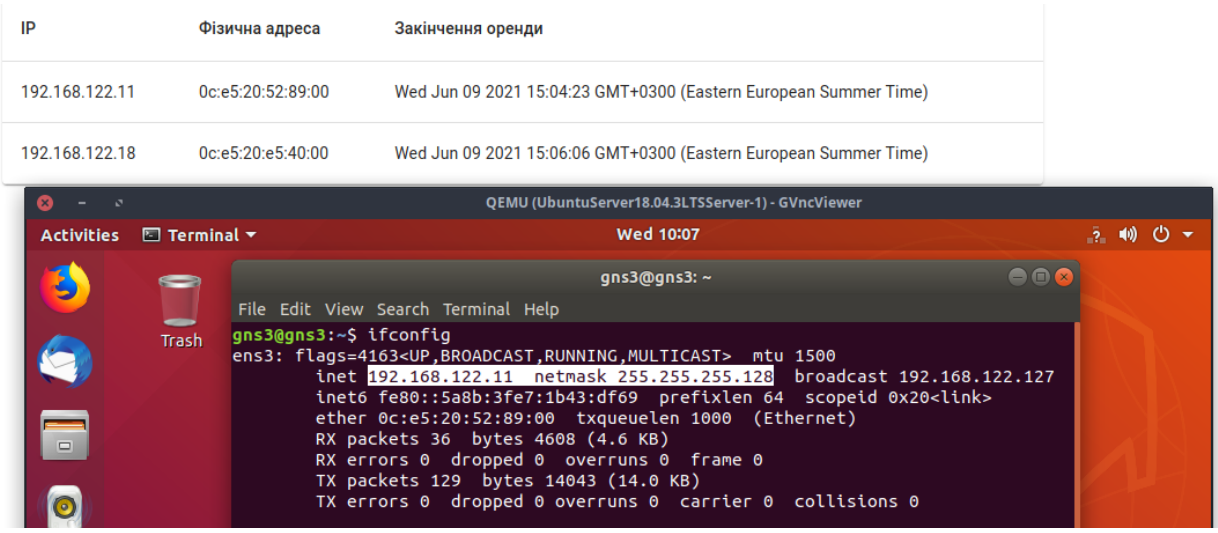

Рисунок 2.12 — Конфігурація інтерфейса ens3 на першому комп'ютері

Після перезавантаження і оновлення оренди, на першому комп’ютері оновилася конфігурація, відповідно до налаштованої на рисунку 2.12.

2.5 Висновки до другого розділу

В другому розділі кваліфікаційної роботи проводиться розробка і тестування DHCP-сервера. DHCP-сервер складається з трьох складових частин, а саме:

- самого DHCP-сервера.
- WebApi, яке дозволяє адмініструвати DHCP-сервер.
- Графічного інтерфейсу, яке дозволяє здійснювати адміністрування.

Також в цьому розділі проводиться ознайомлення з базовими конструкціями мов програмування, які були обрані в першому розділі кваліфікаційної роботи. А також описується методологія розробки програмного на практичному прикладі.

Реалізація DHCP-сервера здійснюється лише можливостями, які надають розробники мови програмування Go, без використання сторонніх модулів і бібліотек.

Для тестування розробленого DHCP-сервера використовується вільний емулятор для симуляції мереж — GNS3.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1Характеристика життєдіяльності людини у системі „людина-машина–середовище існування”

Протягом тривалого часу функції людини стосовно техніки залишалися в основному енергетичними, тобто для керування технікою людина користувалася своєю м’язовою силою. Цю працю можна охарактеризувати, як складні рухові процеси, які вимагають значної фізичної сили, спритності, точної координації рухів. Узгодження людини з технікою зводилося лише до врахування анатомічних та фізіологічних особливостей.

З появою нових видів діяльності виникла потреба врахування психологічних можливостей людини, таких як швидкість реакції, особливості пам’яті та уваги, емоційний стан.

Так як технологічний прогрес не стоїться на місці, виникає потреба в створенні нових програмних продуктів. Комп’ютеризація, з одного боку, розширює можливості людини, а з іншого — значним чином змінило вимоги до її діяльності. Вже не потрібна примітивна праця з виконанням монотонних фізичних операцій, з шаблонною розумовою діяльністю. Збільшилася потреба у творчій висококваліфікований праці. Ускладнилась проблема узгодження умов праці, конструкції машин і робочих місць з психологічним та фізіологічними можливостями людини. Так як людина є невід’ємною і найважливішою складовою частиною системи людина-машина-середовище існування.

Для того, щоб керувати технологічним процесом, людині необхідні дані, які характеризують хід процесу. При виконанні роботи людині доводиться обробляти великий обсяг інформації. При цьому вона зазнає нервового перенапруження. Для розв’язку проблем психологічного характеру конструктори намагаються пристосувати машину до людини, щоб забезпечити найсприятливіший режим роботи.

3.2Аналіз потенційних шкідливостей на робочому місці. Заходи щодо їх зниження

Робоче приміщення, у якому проходило дослідження та розробка DHCP-сервера, має відповідати вимогам щодо охорони праці при організації роботи з візуальними дисплейними терміналами обчислювальних машин.

Дане приміщення має 2 робочих місця. Розглянемо відповідність характеристик робочого місця нормативним. Для цього зведемо основні вимоги до організації робочого місця і відповідні фактичні значення для робочого місця, за яким виконується робота, у табл. 3.1

Робоче приміщення та місце відповідає вимогам щодо охорони праці при організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин

Таблиця 3.1 — Характеристики робочого місця

Параметр	**Позначення**	**Величина**
Довжина, м	A	5
Ширина, м	B	4
Висота, м	H	3
Кількість робочих місць	N	2
Площа, $м^2$	S	20
Об’єм, $м^3$	V	60

Відповідно до ДСН 3.3.6.042-99 роботи, що виконуються користувачами ЕОМ, відносяться до легких фізичних робіт – категорії Іа. У виробничих приміщеннях на робочих місцях з візуальними дисплейними терміналами мають забезпечуватись оптимальні параметри мікроклімату.

Згідно з ДБН В.2.5-28:2018 приміщення, що розглядається, повинне мати природне і штучне освітлення.

Денне (природне) освітлення приміщення відбувається за системою однобічного бічного освітлення. Природне освітлення проникає у приміщення через три світлові прорізи (віконні отвори), які мають регулювальні пристрої для відкривання. Також наявні штори (жалюзі), з можливістю захисту працюючих від прямого попадання сонячних променів і регулювання рівня освітленості в приміщенні. Вікна приміщення орієнтовані на північний-схід. Оскільки будинок розташований у відносній віддаленості від прилеглих будівель, то які небуть перешкоди природному освітленню розглянутого приміщення відсутні.

Всередині приміщення стіни обклеєні світлими шпалерами, стеля побілена, у якості покриття підлоги використаний лінолеум світло-жовтого кольору.

Наявність постійного шуму в робочій зоні призводить до розладу центральної нервової системи і до таких захворювань як неврози, однак фактичний обмірюваний рівень шуму в робочій зоні склав 45 дБА, що задовольняє нормативному рівню шуму (не повинен перевищувати 50 дБА), тому додаткових заходів по поліпшенню цього фактору не потрібно.[26]

Проаналізуємо стан електробезпеки на робочому місці:

- всі прилади на робочому місці використовують напругу 230 В;
- електропроводка захована і ізольована від працівників спеціальним коробом;
- всі робочі місця з ПЕОМ використовують розетки на 230 В;
- споживачі електроенергії - 2 ПЕОМ у вигляді ноутбуків;
- відносна вологість повітря - 60%,
- температура +22 °С - +24 °С;
- підлога: ізолююча – лінолеум.

Проаналізувавши наведене вище, можна сказати, що дані робочі місця відноситься до приміщень без підвищеної електронебезпеки.

У професійних операторів частіше зустрічаються порушення органів зору, опорно-рухового апарату, центральної нервової, імунної, серцево-судинної та статевої систем, захворювання шкіри. Також зафіксована значна кількість скарг операторського персоналу на загальне недомагання, передчасне стомлювання, головний біль, порушення функцій органів зору, які здійснювали несприятливий психофізіологічний вплив на самопочуття та працездатність операторів.

Комп'ютерний зоровий синдром — комплекс порушень здоров'я, який може виникати у користувачів персональних комп'ютерів [27]. Діагноз ставлять, якщо людина, що працює з комп'ютером протягом двох годин, висловлює хоча б дві з десяти скарг:

- головний біль;
- сльозотеча;
- туман;
- двоїння;
- свербіж;
- важкість в очах;
- фотофобія;
- миготіння знаків на екрані.

Синдром розвивається при умові, що робоче місце організовано неправильно — у користувача незручне крісло, відсутні підставки для ніг та кистей рук, неправильно встановлена висота і нахил монітора відносно очей, відстань від очей до екрану. За таких умов тіло людини при роботі займає вимушене положення; спина напружена, шия витягнута, плечі жорстко фіксовані. Напружені м'язи погіршують кровообіг у сонних артеріях, а недостатнє кровозабезпечення головного мозку веде до втрати можливості нормально мислити, появи головного болю. а фоні шийного остеохондрозу з'являється відчуття випирання очних яблук, туману в очах, мушок та райдужних кіл у полі зору. Розвитку комп'ютерного зорового синдрому сприяє поганий мікроклімат приміщення, значна загальна іонізація та мікробне забруднення, а також куріння.

Робота за комп'ютером характеризується також тим, що постійний напружений погляд на екран монітора зменшує частоту моргання. При цьому погіршується зволоження поверхні очного яблука сльозовою рідиною, яка захищає рогівку ока від висихання, пилу та інших забруднень. Це може призвести до виникнення так званого синдрому Сікка: рогівка висихає і мутніє, і як наслідок розвивається сліпота.

3.3Висновки до третього розділу

В даному розділі було проаналізовано основні проблеми безпеки життєдіяльності та охорони праці, які можуть виникнути у працівника під час роботи. Були охарактеризовані відношення людини в системі “людина—машина—середовище існування”, а також були виділені основні вимоги до приміщення, освітлення та ергономічних характеристик.

У приміщенні застосовується бокове природне освітлення та штучне (два ряди світильників Л201Б 4х40-0.3, у кожному з яких знаходиться по чотири лампи типу ЛБ-40). Встановлено, що температура повітря у приміщенні становить 24 С. Зазначено, що приміщення за групою електронебезпечності відноситься до приміщень без підвищеної небезпеки ураження струмом.

Також окремо було розглянуто фактори, що впливають на стан користувачів комп’ютера. Зокрема, було детально розглянуто зоровий дискомфорт, його прояви та причини.

ВИСНОВКИ

Основним завданням кваліфікаційної роботи, було створення DHCP-сервера для зручного керування локальними мережами. Що дозволяє спростити життя організаціям, окремим людям з всього світу, при налаштуванні мереж. Це буде залишатися актуальним, доки комп'ютерні мережі будуть використовуватися. В наш час складно уявити життя, без миттєвої передачі інформації на великі відстані, у цифровому вигляді. Ця зручність дозволяє ввести бізнес і спілкуватися з друзями на великих відстанях.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр» подано інформацію про DHCP-протокол, його розвиток, та причини його використання. Також в цьому розділі обґрунтовано вибір інструментів розробки.

В другому розділі кваліфікаційної роботи було розроблено DHCP-сервер, який дозволяє спростити процес підключення користувачів до локальної мережі.

В розділі «Безпека життєдіяльності, основи охорони праці» охарактеризована життєдіяльність людини у системі „людина-машина–середовище існування". Бо саме баланс цієї системи є надзвичайно важливий в житті людини, оскільки їй нерідко доводиться обробляти великий обсяг інформації, від якого будуть залежати.

ПЕРЕЛІК ДЖЕРЕЛ

1. International Telecommunication Union Measuring the Information Society Report Volume 1 URL: https://www.itu.int/en/ITU-D/Statistics/Documents/publications/misr2018/MISR-2018-Vol-1-E.pdf (дата звернення 15.04.2021).

2. Фейт С. TCP / IP. Архитектура. Протоколы. Реализация / Сідні Фейт. – Сан-Франциско: Лори, 2016. – 424 с.

3. Гант К. TCP/IP — Сетевое администрирование / Крейг Гант. – Санкт-петербург: O'REILLY, 2018. – 813 с

4. Чеппел Л. TCP/IP. Учебный курс / Л. Чеппел, Т. Ед. – Санкт-Петербург: БХВ-Петербург, 2019. – 960 с.

5. RFC903 [Електронний ресурс]. – 1984. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc903.

6. RFC2131 [Електронний ресурс]. – 1996. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/RFC2131.

7. Dynamic Host Configuration Protocol [Електронний ресурс] // Network Working Group. – 1997. – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc2131.

8. Таненбаум Е. Комп'ютерні мережі / Е. Таненбаум, Д. Ветералл. – Амстердам: Brill, 2018. – 960 с. – (6). – (Классика Computer Science).

9. Робачевский А. Интернет изнутри. Экосистема глобальной сети / Андрей Робачевский. – Москва: Альпина Паблишер, 2019. – 224 с.

10. Sportack M. IP Routing Fundamentals / Mark Sportack. – Indianapolis: Cisco Press, 2020. – 328 с.

11. Венделл О. Официальное руководство Cisco по подготовке к сертификационным экзаменам CCNA ICND2 200-105 / Одом Венделл. – Індіанаполіс: Cisco Press, 2019. – 1008 с.

12. Hey! Ho! Let's Go! [Електронний ресурс] / [Р. Гріземер, Р. Пайк, К. Томпсон та ін.] // Google Open Source Blog. – 2019. – Режим доступу до ресурсу: https://opensource.googleblog.com/2009/11/hey-ho-lets-go.html.

13. Кейлеб Д. Introducing Go: Build Reliable, Scalable Programs / Доксі Кейлеб. – Ньютон: O'REILLY, 2021. – 124 с.

14. Саммерфілд М. Programming in Go: Creating Applications for the 21st Century / Марк Саммерфілд. – Boston: Addison–Wesley, 2018. – 493 с.

15. Кокс-Будай К. Concurrency in Go: Tools and Techniques for Developers / Кетрін Кокс-Будай. – Ньютон: O'REILLY, 2017. – 238 с.

16. Go Developer Survey 2019 Results [Електронний ресурс] // The Go Blog. – 2020. – Режим доступу до ресурсу: https://blog.golang.org/survey2019-results.

17. Вірух Р. The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js / Робін Вірух. – Ньютон: O'REILLY, 2017. – 208 с.

18. Бенкс А. Learning React: Functional Web Development with React and Redux / А. Бенкс, Е. Порчелло. – Ньютон: O'REILLY, 2017. – 350 с.

19. Мардан А. React Quickly: Painless web apps with React, JSX, Redux, and GraphQL / Азат Мардан. – Ньютон: O'REILLY, 2018. – 528 с.

20. Гурлі Д. HTTP: The Definitive Guide / Д. Гурлі, Б. Тотті. – Ньютон: O'REILLY, 2019. – 596 с.

21. Лоре А. Проектирование веб-API / Арно Лоре. – Ньютон: O'REILLY, 2020. – 440 с.

22. Маррс Т. JSON at Work: Practical Data Integration for the Web / Том Маррс. – Ньютон: O'REILLY, 2017. – 374 с.

23. Net.Http [Електронний ресурс] // Go documentation – Режим доступу до ресурсу: https://golang.org/pkg/net/http/.

24. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» - К.: Міністерство охорони здоров’я України, 1999. – 12 с.

25. ДБН В.2.5-28-2008 «Природне і штучне освітлення» – К.: Мінрегіон України, 2018. – 157 с..

26. Наказ МОЗ України від 22.02.2019 року № 463 “Про затвердження Державних санітарних норм допустимих рівнів шуму в приміщеннях житлових та громадських будинків і на території житлової забудови”, зареєстрований в Мін'юсті України 20.03.2019 року за № 281/33252.

27. Ульяницька, Н. Я., О. Я. Андрійчук, and Н. Б. Грейда. "Особливості відновлення зорової втоми у комп’ютерних користувачів старшого шкільного віку.».

ДОДАТКИ

Додаток А

Лістинг файлів програмного коду сервера

Файл Server.go

```
package dhcp

import (
     "net"
     "strconv"
)

type Handler interface {
     ServeDHCP(req Packet, msgType MessageType, options Options)
Packet
}

type Size int

type ServeConn interface {
     ReadFrom(Packet) (Size, net.Addr, error)
     WriteTo(Packet, net.Addr) (Size, err error)
}

func Serve(conn ServeConn, handler Handler) error {

     buffer := make([]byte, 1500)
     for {
          n, addr, err := conn.ReadFrom(buffer)
          if err != nil {
               return err
          }
          if n < 240 { // Packet too small to be DHCP
               continue
          }
          req := Packet(buffer[:n])
          if req.HLen() > 16 { // Invalid size
               continue
          }
          options := req.ParseOptions()
          var reqType MessageType
          if t := options[OptionDHCPMessageType]; len(t) != 1 {
               continue
          } else {
               reqType = MessageType(t[0])
               if reqType < Discover || reqType > Inform {
                    continue
               }
          }
          if res := handler.ServeDHCP(req, reqType, options); res
!= nil {
```

```
                ip, port, err := net.SplitHostPort(addr.String())
                if err != nil {
                     return err
                }

                if net.ParseIP(ip).Equal(net.IPv4zero) ||
req.Broadcast() {
                     port, _ := strconv.Atoi(port)
                     addr = &net.UDPAddr{
                          IP:   net.IPv4bcast,
                          Port: port,
                     }
                }
                if _, e := conn.WriteTo(res, addr); e != nil {
                     return e
                }
           }

     }
}

func ListenAndServe(handler Handler) error {
     l, err := net.ListenPacket("udp", ":67")

     if err != nil {
          return err
     }
     defer l.Close()

     return Serve(l, handler)
}
```

Файл dhcp.go

```
package dhcp

import (
     "context"
     "encoding/json"
     "log"
     "math/rand"
     "net"
     "time"

     "gitlab.com/volodya_melnyk/dhcp-server/back/config"
)

type lease struct {
     Nic    string    `json:"nic"`
     Expiry time.Time `json:"expiry"`
}
```

```
var listener net.PacketConn = nil
var cancel context.CancelFunc
var Status byte
var leases map[int]lease

const (
     RunCode      = 1
     ShutdownCode = 2
)

type DHCPHandler struct {
     ip            net.IP        // Server IP to use
     options       Options       // Options to send to DHCP Clients
     start         net.IP        // Start of IP range to distribute
     leaseRange    int           // Number of IPs to distribute
(starting from start)
     leaseDuration time.Duration // Lease period
     leases        map[int]lease // Map to keep track of leases
}

func GetLeases() []byte {
     bytes, _ := json.Marshal(leases)

     return bytes
}

func createDHCPHandler() *DHCPHandler {
     serverIP := config.GetServerIP()

     handler := &DHCPHandler{
          ip:            serverIP,
          leaseDuration: 2 * time.Hour,
          start:         config.GetStartIP(),
          leaseRange:    50,
          leases:        make(map[int]lease, 10),
          options: Options{
               OptionSubnetMask:       []byte{255, 255, 255, 0},
               OptionRouter:           []byte(serverIP),
               OptionDomainNameServer: []byte(serverIP),
          },
     }

     return handler
}

func Shudown() {
     cancel()
     Status = ShutdownCode
}

func Run() (*DHCPHandler, context.CancelFunc, error) {
     networkInterface := "virbr0"
     ctx := context.Background()
```

```
     var handler *DHCPHandler

     ctx, cancel = context.WithCancel(context.Background())
     var err error

     go func() {
          listener, err = NewUDP4BoundListener(networkInterface,
":67")
          if err != nil {
               log.Println(err)
               panic(err)
          }

          go func() {
               <-ctx.Done()
               listener.Close()
          }()

          handler = createDHCPHandler()
          Status = RunCode

          Serve(listener, handler)
     }()

     return handler, cancel, nil
}
func (s *DHCPHandler) ServeDHCP(p Packet, msgType MessageType,
options Options) (d Packet) {
     log.Println(s.leases)
     leases = s.leases

     switch msgType {
     case Discover:
          free, nic := -1, p.CHAddr().String()
          for i, v := range s.leases { // Find previous lease
               if v.Nic == nic {
                    free = i
                    goto reply
               }
          }
          if free = s.freeLease(); free == -1 {
               return
          }
     reply:
          return ReplyPacket(p, Offer, s.ip, IPAdd(s.start, free),
s.leaseDuration,

     s.options.SelectOrderOrAll(options[OptionParameterRequestList]
))
     case Request:
          if server, ok := options[OptionServerIdentifier]; ok &&
!net.IP(server).Equal(s.ip) {
               return nil // Message not for this dhcp server
```

```
          }
          if reqIP := net.IP(options[OptionRequestedIPAddress]);
len(reqIP) == 4 {
               if leaseNum := IPRange(s.start, reqIP) - 1; leaseNum
>= 0 && leaseNum < s.leaseRange {
                    if l, exists := s.leases[leaseNum]; !exists ||
l.Nic == p.CHAddr().String() {
                         s.leases[leaseNum] = lease{Nic:
p.CHAddr().String(), Expiry: time.Now().Add(s.leaseDuration)}
                         return ReplyPacket(p, ACK, s.ip,
net.IP(options[OptionRequestedIPAddress]), s.leaseDuration,

     s.options.SelectOrderOrAll(options[OptionParameterRequestList]
))
                    }
               }
          }
          return ReplyPacket(p, NAK, s.ip, nil, 0, nil)
     case Release, Decline:
          nic := p.CHAddr().String()
          for i, v := range s.leases {
               if v.Nic == nic {
                    delete(s.leases, i)
                    break
               }
          }
     }
     return nil
}

func (s *DHCPHandler) freeLease() int {
     now := time.Now()
     b := rand.Intn(s.leaseRange) // Try random first
     for _, v := range [][]int{{b, s.leaseRange}, {0, b}} {
          for i := v[0]; i < v[1]; i++ {
               if l, ok := s.leases[i]; !ok || l.Expiry.Before(now)
{
                    return i
               }
          }
     }
     return -1
}
```

Додаток Б

Лістинг файлів програмного коду API

Файл api.go

```
package api

import (
	"fmt"
	"log"
	"net/http"
	"time"

	"github.com/gorilla/mux"
	"gitlab.com/volodya_melnyk/dhcp-server/back/config"
	"gitlab.com/volodya_melnyk/dhcp-server/back/dhcp"
)

func shutdownDHCP(mDHCP chan byte) http.HandlerFunc {
	return func(w http.ResponseWriter, r *http.Request) {
		fmt.Fprintf(w, "shutdown")

		mDHCP <- dhcp.ShutdownCode
	}
}

func startDHCP(mDHCP chan byte) http.HandlerFunc {
	return func(w http.ResponseWriter, r *http.Request) {
		fmt.Fprintf(w, "start dhcp server")

		mDHCP <- dhcp.RunCode
	}
}

func Create(mDHCP chan byte) *http.Server {
	router := mux.NewRouter()

	router.HandleFunc("/api/leases", func(w http.ResponseWriter, r
*http.Request) {
		log.Println(dhcp.GetLeases())

		w.Header().Set("Content-Type", "application/json")
		w.Write(dhcp.GetLeases())
	})

	// auth
	// router.Use(authMiddleware)
	router.HandleFunc("/api/auth/login", login).Methods("POST")
	router.HandleFunc("/api/auth/signup", signup).Methods("POST")
	router.HandleFunc("/api/auth/logout",
logout).Methods("DELETE")
```

```
	router.HandleFunc("/api/auth/user", user)

	// controll dhcp
	router.HandleFunc("/api/shutdown", shutdownDHCP(mDHCP))
	router.HandleFunc(("/api/start"), startDHCP(mDHCP))

	srv := &http.Server{
		Handler:      router,
		Addr:         config.GetAddrForApi(),
		WriteTimeout: time.Second * 15,
		ReadTimeout:  time.Second * 15,
		IdleTimeout:  time.Second * 60,
	}

	return srv
}
```

Файл auth.go

```
package api

import (
	"context"
	"encoding/json"
	"fmt"
	"net/http"
	"time"

	"gitlab.com/volodya_melnyk/dhcp-server/back/db"
	"gitlab.com/volodya_melnyk/dhcp-server/back/helper"
	"gitlab.com/volodya_melnyk/dhcp-server/back/models"
	"gitlab.com/volodya_melnyk/dhcp-server/back/utils"
	"go.mongodb.org/mongo-driver/bson"
	"go.mongodb.org/mongo-driver/bson/primitive"
)

type ContextKey string

const ContextUserKey ContextKey = "user"

func authMiddleware(next http.Handler) http.Handler {
	return http.HandlerFunc(func(w http.ResponseWriter, r
*http.Request) {

		if r.RequestURI == "/api/auth/login" {
			next.ServeHTTP(w, r)
		} else {
			cookie, err := r.Cookie("token")
			if err != nil {
				w.WriteHeader(http.StatusForbidden)
				return
			}
```

```
                clientToken := cookie.Value

                _, err = helper.ValidateToken(clientToken)
                if err != nil {
                     w.WriteHeader(http.StatusForbidden)
                     return
                }

                cookie, err = r.Cookie("user_id")
                if err != nil {
                     w.WriteHeader(http.StatusForbidden)
                     return
                }
                userId := cookie.Value

                var user models.User

                ctx, cancel :=
context.WithTimeout(context.Background(), 100*time.Second)
                db.UsersCollection().FindOne(ctx, bson.M{"user_id":
userId}).Decode(&user)
                defer cancel()

                userMarshal, _ := json.Marshal(user)

                ctx = context.WithValue(r.Context(), ContextUserKey,
userMarshal)

                next.ServeHTTP(w, r.WithContext(ctx))
           }
     })
}

func login(w http.ResponseWriter, r *http.Request) {
     var ctx, cancel = context.WithTimeout(context.Background(),
100*time.Second)
     var user models.User
     var foundUser models.User

     err := json.NewDecoder(r.Body).Decode(&user)
     if err != nil {
          w.WriteHeader(http.StatusBadRequest)
          w.Write([]byte(err.Error()))
     }

     err = db.UsersCollection().FindOne(ctx, bson.M{"username":
user.Username}).Decode(&foundUser)
     defer cancel()
     if err != nil {
          w.WriteHeader(http.StatusBadRequest)
          w.Write([]byte(err.Error()))
```

```
          return
     }

     passwordIsValid, msg := utils.VerifyPassword(user.Password,
foundUser.Password)
     defer cancel()

     if !passwordIsValid {
          response := fmt.Sprintf("{error: %s}", msg)

          w.Write([]byte(response))
          return
     }
     token, refreshToken, _ := helper.GenerateAllTokens(user.Email,
user.Username, user.User_id)

     helper.UpdateAllTokens(token, refreshToken, foundUser.User_id)

     jsonBytes, _ := json.Marshal(foundUser)

     cookie := http.Cookie{
          Name:     "token",
          Value:    *foundUser.Token,
          Path:     "/api",
          HttpOnly: true,
     }
     http.SetCookie(w, &cookie)

     cookie = http.Cookie{
          Name:     "refresh_token",
          Value:    *foundUser.Refresh_token,
          Path:     "/api",
          HttpOnly: true,
     }
     http.SetCookie(w, &cookie)

     cookie = http.Cookie{
          Name:     "user_id",
          Value:    foundUser.User_id,
          Path:     "/api",
          HttpOnly: true,
     }
     http.SetCookie(w, &cookie)

     w.Write([]byte(jsonBytes))
}

func signup(w http.ResponseWriter, r *http.Request) {
     ctx, cancel := context.WithTimeout(context.Background(),
100*time.Second)
     var user models.User

     err := json.NewDecoder(r.Body).Decode(&user)
```

```
     if err != nil {
          w.WriteHeader(http.StatusBadRequest)
          w.Write([]byte(err.Error()))

     }

     count, err := db.UsersCollection().CountDocuments(ctx,
bson.M{"username": user.Username})
     defer cancel()
     if err != nil {
          w.WriteHeader(http.StatusInternalServerError)
          w.Write([]byte("error occured while checking for the
email"))
     }

     password := utils.HashPassword(user.Password)
     user.Password = password

     if count > 0 {
          w.WriteHeader(http.StatusInternalServerError)
          w.Write([]byte("this email number already exists"))
          return
     }

     user.Created_at, _ = time.Parse(time.RFC3339,
time.Now().Format(time.RFC3339))
     user.Updated_at, _ = time.Parse(time.RFC3339,
time.Now().Format(time.RFC3339))
     user.ID = primitive.NewObjectID()
     user.User_id = user.ID.Hex()

     token, refreshToken, _ := helper.GenerateAllTokens(user.Email,
user.Username, user.User_id)

     user.Token = &token
     user.Refresh_token = &refreshToken

     resultInsertionNumber, insertError :=
db.UsersCollection().InsertOne(ctx, user)
     if insertError != nil {
          w.WriteHeader(http.StatusConflict)
          w.Write([]byte("error: User item was not created"))
     }
     defer cancel()

     bytesResponse, _ := json.Marshal(resultInsertionNumber)

     cookie := http.Cookie{
          Name:     "token",
          Value:    *user.Token,
          HttpOnly: true,
          Path:     "/api",
     }
```

```
	http.SetCookie(w, &cookie)

	cookie = http.Cookie{
		Name:     "refresh_token",
		Value:    *user.Refresh_token,
		HttpOnly: true,
		Path:     "/api",
	}
	http.SetCookie(w, &cookie)

	cookie = http.Cookie{
		Name:     "user_id",
		Value:    user.User_id,
		Path:     "/api",
		HttpOnly: true,
	}
	http.SetCookie(w, &cookie)

	// w.WriteHeader(http.StatusOK)
	w.Write(bytesResponse)
}

func userFromContext(ctx context.Context) models.User {
	cUser := ctx.Value(ContextUserKey).([]byte)

	var user models.User

	json.Unmarshal(cUser, &user)

	return user
}

func user(w http.ResponseWriter, r *http.Request) {
	user := userFromContext(r.Context())
	responseString, _ := json.Marshal(&user)

	w.Header().Set("Content-Type", "application/json")

	w.WriteHeader(http.StatusOK)
	w.Write(responseString)
}

func logout(w http.ResponseWriter, r *http.Request) {
	cookie := http.Cookie{
		Name:     "token",
		Value:    "",
		HttpOnly: true,
		Path:     "/api",
	}
	http.SetCookie(w, &cookie)

	cookie = http.Cookie{
		Name:     "refresh_token",
```

```
            Value:    "",
            HttpOnly: true,
            Path:     "/api",
        }
        http.SetCookie(w, &cookie)

        cookie = http.Cookie{
            Name:     "user_id",
            Value:    "",
            Path:     "/api",
            HttpOnly: true,
        }
        http.SetCookie(w, &cookie)

        w.WriteHeader(http.StatusOK)
}
```