Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій

Інтегрована система керування адресним простором IP-мережі підприємства

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 6.040302

Керівник курсової роботи
Старший викладач кандидат наук

Дмитро Черкасов

“____” _____

2022 р.

Виконав студент ФІ-4

Поліщук Ю. О.

“____” _____ 2022 р.

Анотація

У даній роботі розглядаються основні принципи побудови інтегрованих систем керування адресним простором (IPAM) та описується процес розробки власного IPAM-сервісу.

Вступ

Створення всесвітньої мережі Інтернет спричинило технічну революцію, яка пронеслась крізь всі соціальні та економічні сфери людського суспільства. Всесвітня павутинна стерла кордони між країнами та надала можливість співпраці для людей та компаній, що знаходяться на різних материках. Розвиток IT-технологій дозволив автоматизувати процеси, які раніше вимагали величезних людських ресурсів, до однієї програми, яка виконується на сервері в дата-центрі.

Кількість пристроїв, підключених до Інтернет, постійно зростає, причому з вибуховою швидкістю. Кожну секунду 127 нових пристроїв вперше підключаються до всесвітньої павутини.[1]

Збільшення кількості інтернет-пристроїв значно ускладнює процес адміністрування мереж усередині компанії. Раніше підмережа компанії складалася з сотні комп'ютерів в офісі, а тепер вона може мати кілька тисяч пристроїв, які до того ж можуть знаходитись в різних географічних локаціях. Такі масштаби вимагають спеціалізованих систем, які дозволятимуть вести облік пристроїв в мережі, створювати схеми мереж та їх ієрархії, надаючи доступ декільком адміністраторам, та автоматизувати рутинний процес конфігурації пристроїв та систем.

Інтегровані системи керування адресним простором (скорочено IPAM) розроблюються для вирішення такого роду задач. Вони дозволяють керувати простором IP-адрес, що наявний у компанії, розбиваючи його на підмережі, надають можливість моніторингу поточного стану мережі та інтеграції з іншими системами (DNS, DHCP, CRM, ERP, тощо). Також вони дозволяють обмежувати права доступу до різних частин мережі для кожного з адміністраторів, зменшуючи ризики повного краху мережевої системи.

У даній курсовій роботі ми розглянемо основні принципи розробки IPAM-систем, спираючись на проблеми, які вони вирішують. Ми розглянемо основні системи, що присутні сьогодні, їхні переваги й недоліки.

У кінці ми розробимо власну IPAM-систему, яка дозволить адмініструвати мережі малих та середніх розмірів. Вона дозволить нам створювати підмережі, виділяти адресний простір для них, додавати пристрої до них, отримувати інформацію про кожен елемент мережі. Система буде мати клієнт-серверну архітектуру, а отже надасть можливість адміністрування декільком особам одночасно. Усі дані будуть зберігатися в SQL базі данних. Система також матиме інтеграцію з сервісами DHCP та DNS.

Зміст

Огляд основних функцій IPAM систем

IPAM (IP Address Management) – це метод планування, відслідковування та управління адресним простором IP-мережі (IP – Internet Protocol, основний протокол комунікації в мережі інтернет). [2]

IP адресація полягає в створенні підмереж серед адресного простору підприємства та присвоєнні IP адрес всередині мереж конкретним пристроям.

Управління великою кількістю адрес (іноді їх кількість перевищує десятки тисяч) є досить складною задачою. Саме тут на допомогу приходять IPAM-системи.

IPAM-система – це комплекс програмних рішень, який дозволяє відслідковувати за станом адресного простору в зручному для користувача форматі.

Кожна з таких систем вміє зберігати дані про IP-мережі та їх адресний простір. Окрім цього такі системи вміють зберігати дані про пристрої в мережі (**MAC-адреса**, модель, виробник, серійний номер).

Також більшість з таких систем мають підтримку протоколу **DHCP**. Dynamic Host Configuration Protocol (DHCP) – це клієнт-серверний протокол, який дозволяє пристроям отримувати IP-адресу та інші пов'язану конфігураційну інформацію (маска підмережі та основний шлюз) в автоматичному режимі.[3] Це, звісно, значно спрощує процес налаштування пристроїв, проте потребує додаткової інтеграції всередині IPAM-системи, адже розподіл адрес в мережі тепер залежить не тільки від оператора системи.

Інтеграція з **DNS** – один з найважливіших функціоналів IPAM-системи. **Domain Name System (DNS) –** це ієрархічна та децентралізована

система іменування пристроїв в мережі Інтернет. У ній кожний домен отримує IP-адресу за якою клієнт звертатиметься, коли він захоче отримати сторінку в мережі Інтернет. Інтеграція з IPAM-системами спрощує процес управлінням DNS-сервером, адже вся потрібна інформація зберігається в одному місці.

Ще одна технологія, яка набула поширення в наші дні – це **VLAN**. Virtual local area network (VLAN) – це технологія в роутерах та комутаторах, яка дозволяє створювати “віртуальні” локальні мережі. Такі мережі не залежать від фізичної топології пристроїв у мережі та, натомість, створюються на логічному рівні. Ця технологія приносить багато користі підприємствам, адже дозволяє об’єднати, наприклад, бухгалтерські офіси з різних філіалів компанії в одну мережу. Проте, вона ускладнює процес адміністрування мережі, оскільки різні її частини можуть знаходитись в різних містах. IPAM-системи дозволяють вказати до якого VLAN належить підмережа. Також вони можуть зберігати інформацію про фізичне розташування пристрою, що значно спрощує роботу при усуненні неполадок.

Більшість подібних систем побудовані за клієнт-серверною архітектурою, у вигляді веб-сторінок. Це дозволяє адміністратору отримати інформацію про мережу з будь-якої точки планети. До того ж компанії зазвичай мають декількох адміністраторів, і кожен з них повинен мати доступ до єдиного джерела правди.

Розробка власної IPAM-системи

Вимоги до системи

У рамках даної курсової роботи, я розробив власну спрощену IPAM-систему. ЇЇ функціонал обмежений у порівнянні справжніми системи, проте з плюсів можна зазначити набагато простіший для розуміння інтерфейс користувача.

Як і кожна з IPAM-систем, вона дозволяє створювати мережі та підмережі й додавати до них пристрої. Система також дозволяє зберігати додаткову інформацію, яка присутня в справжніх IPAM-системах (підтримка DHCP, DNS, VLAN), проте інтеграція з сторонніми сервісами відсутня.

Основний функціонал

- Система доступна для користувача у вигляді веб-сторінки
- Створення, редагування та видалення IP-мереж
- Створення, редагування та видалення мереж
- Створення, редагування та видалення інформації про пристрої
- Перегляд інформації про IP-мережі та підмережі з можливістю сортування та пагінації на кожному з рівнів
- Перегляд інформації про всі пристрої в мережі з можливістю сортування та фільтрації
- Перегляд інформації про пристрої в мережі
- Присвоєння пристрою до мережі
- Створення, редагування та видалення інформації про DNS-сервер, який використовує мережа.
- Присвоєння домену пристрою, який знаходиться в мережі
- Збереження інформації про використання DHCP в мережі
- Створення, редагування та видалення інформації про виробника пристрою, його назву та серійний номер
- Створення, редагування та видалення інформації про розташування пристрою
- Автентифікація користувачів за JWT-токенами
- Авторизація користувачів за ролями (Адміністратор та Користувач)

- Підтримка GraphQL API, який дозволяє користувачам додавати, редагувати та видаляти дані, не використовуючи веб-сайт. У майбутньому дозволить розширити функціонал до підтримки мобільних додатків.
- Автоматична генерація документації API-запитів

Особливості застосунку

- Простий для розуміння інтерфейс користувача
- Наявність як веб-версії так і документованого API
- Просте розгортання

Архітектура системи

Система побудована за стандартною трирівневою архітектурою. Вона складається з фронт-енду, який представлений у вигляді SPA-застосунку, бек-енду та бази даних.

Фронт-енд та бек-енд комунікують один з одним за допомогою HTTP запитів.

Використання подібної архітектури дозволяє використовувати систему багатьом користувач з різних пристроїв одночасно. При цьому вся інформація про мережу зберігається в одному місці (Single source of truth). До того ж така архітектура відв’язує рівень презентації від рівня логіки та дозволяє, наприклад, створити мобільну версію застосунку в майбутньому або CLI-версію додатку.

Стек технологій

В якості основного стеку для бек-енд частини системи, я обрав .NET 6. Платформа .NET – одна з найбільш стрімко розвиваючихся, на якій можлива розробка застосувань практично любого рівня складності та направлення.[4]

Головний фреймворк для побудови серверних застосувань на платформі .NET – це ASP.NET Core. Це один з найбільш потужних та функціональних веб-фреймворків, який при цьому не вимагає великої кількості шаблонного коду і має одні з найкращих показників швидкодії.[5]

В якості фронтенд-рішення, я вирішив зупинитися на одному з SPA-фреймворків. Вони дозволяють створювати функціонально-складні сторінки відносно просто у порівнянні з традиційними серверними сторінками. До того ж, вони ідеально підходять під архітектуру, коли зв’язок з бек-ендом відбувається за допомогою API.

У якості фронтенд-фреймворка, я обрав React. Це найпопулярнійший SPA-фреймворк, розроблений компанією Facebook, з величезною кількістю бібліотек і компонентів для зручної розробки.[6]

Для швидкої розробки фронтенд-частини я обрав бібліотеку React MUI. Вона містить всі основні компоненти (форми, списки, таблиці, тощо), які стилізовані за принципами Material Design.[7], [8]

Для комунікації між фронт-ендом та бек-ендом, я вирішив використати GraphQL. GraphQL – це мова запитів, розроблена компанією

Facebook. У ній дані представлені у вигляді графу зі зв’язками.

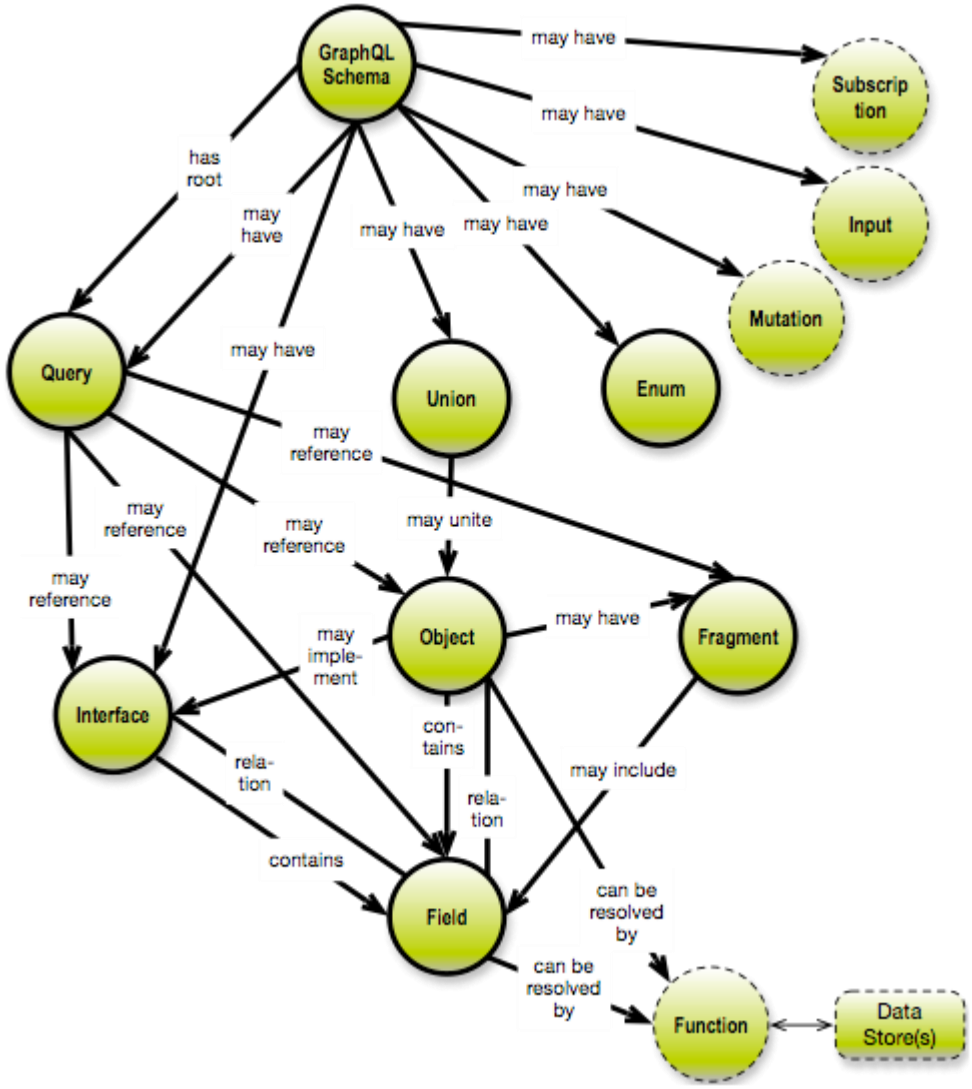

Рисунок 1 Схема даних в GraphQL Джерело: Thomas Frisendal

В GraphQL сервер публікує схему даних, в якій знаходяться типи всіх даних про які можна отримати інформацію. Приклад такої схеми, ви можете побачити нижче.

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Клієнти GraphQL, самі будують свої власні запити. Вони можуть запити інформацію про лише певні поля чи довантажити поля з об’єктів, які пов’язані з даним в одному запиті. При цьому вони здатні, фільтрувати

і сортувати дані. Присутня також підтримка пагінації. Приклад запиту проектів, що мають ім’я GraphQL, ви можете побачити нижче.

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Виконавши запит, сервер повертає лише ті дані, які потребував користувач.

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

GraphQL також дозволяє об’єднувати декілька API джерел, одночасно. Ми можемо створити власний сервіс, який бере дані з інших GraphQL сервісів та надає користувачам зручний доступ до даних.[9]

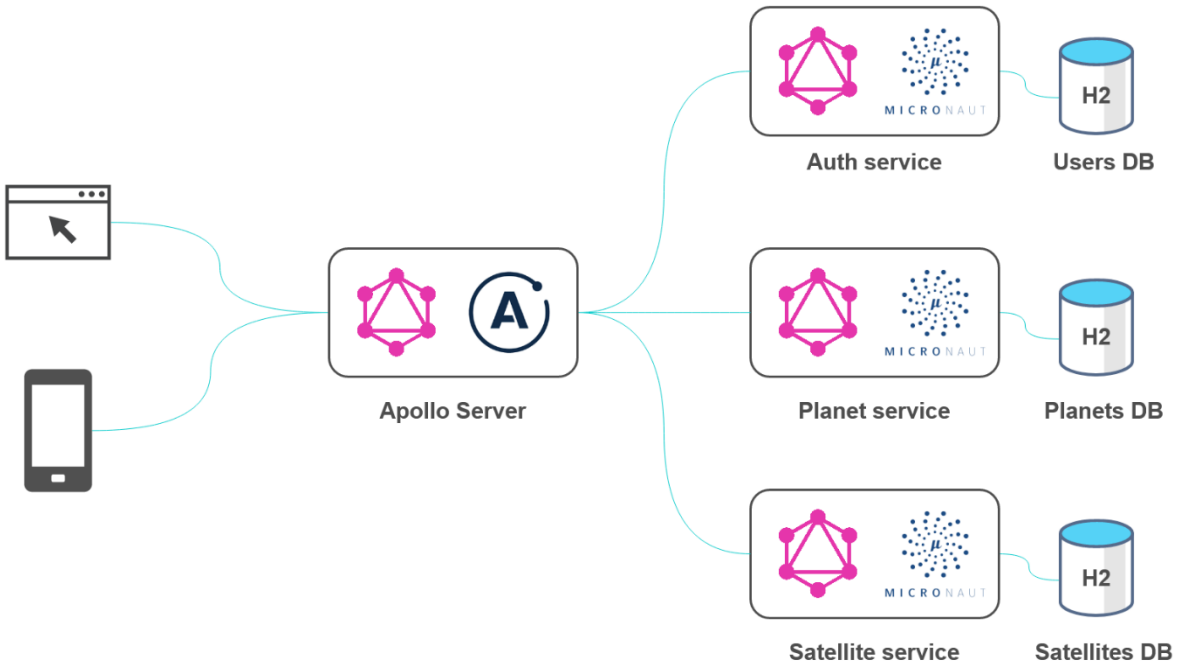

Рисунок 2 Джерело: Roman Kudryashov

Така функціональність має багато плюсів, адже фактично дозволяє користувачам системи самостійно створювати власні запити, без будь-яких дій з боку розробника бек-енду. Користувачі самі можуть створити

запит на вибірку даних за саме тими критеріями, що потрібні власне їм самим.

Розробники IPAM-системи Netbox, оцінили можливості даного інструменту і тому Netbox також має підтримку GraphQL запитів.[10]

У якості бази даних я обрав PostgreSQL. Це одна з найпопулярніших баз даних з відкритим вихідним кодом. Вона має один з найширших функціоналів серед безкоштовних СУБД. Для даної предметної області вона підходить особливо добре, адже має нативну підтримку типів даних **inet** (**IPv4** або **IPv6** адреса) та **cidr** – (**IPv4** або **IPv6** адреса з маскою у форматі **CIDR**). [11]

Для роботи з базою даних на бек-енді, я використовую Entity Framework Core. Це найпопулярніша ORM для .NET, яка розроблена Microsoft. Вона дозволяє не створювати SQL-запити вручну, а писати C# код, який вона сама транслює в SQL. У цього підходу є як і плюси, так і мінуси. З плюсів можна назвати відсутність потреби в детальному вивченні SQL. Окрім того рівень бізнес-логіки повністю відв’язаний від бази-даних. У теорії її можна замінити з PostgreSQL на MySQL, не змінюючи самого коду програми. Основним мінусом є те, що запити не завжди оптимальні за швидкодією.[12]

Для роботи з **GraphQL** на бек-енді, я використовую фреймворк **Hot Chocolate.** Це фреймворк, який інтегрується в **ASP.NET Core** і дозволяє швидко створювати **GraphQL** API. Бібліотека автоматично генерує схему даних та надає клієнт в якому користувач може створювати запити. Фреймворк надає широкі можливості з сортування, фільтрації, пагінації та кастомізації.

Одним з додаткових плюсів **Hot Chocolate** є його глубока інтеграція з **Entity Framework**. Бібліотека самостійно транслює **GraphQL** запити в

SQL, фактично обходячи потребу в будь-якій обробці даних на самому бекенді.

Для роботи з GraphQL на стороні фронт-енду, я використовую бібліотеку **Apollo**. Це найпопулярніша бібліотека для GraphQL та React, яка має широкий функціонал та вбудоване кешування.[13]

Існують інструменти, які автоматично генерують код для запиту даних на основі GraphQL схем. Один з них це **GraphQL Code Generator**. Я використовував саме його для генерації Apollo-коду запитів. [14]

Розробка застосунку

ER-Модель

База даних складається з двох схем: public та auth.

Public схема містить всі дані, що відносяться до роботи з IP адресацією. Вона містить таблиці для мереж, пристроїв та зв’язків між ними.

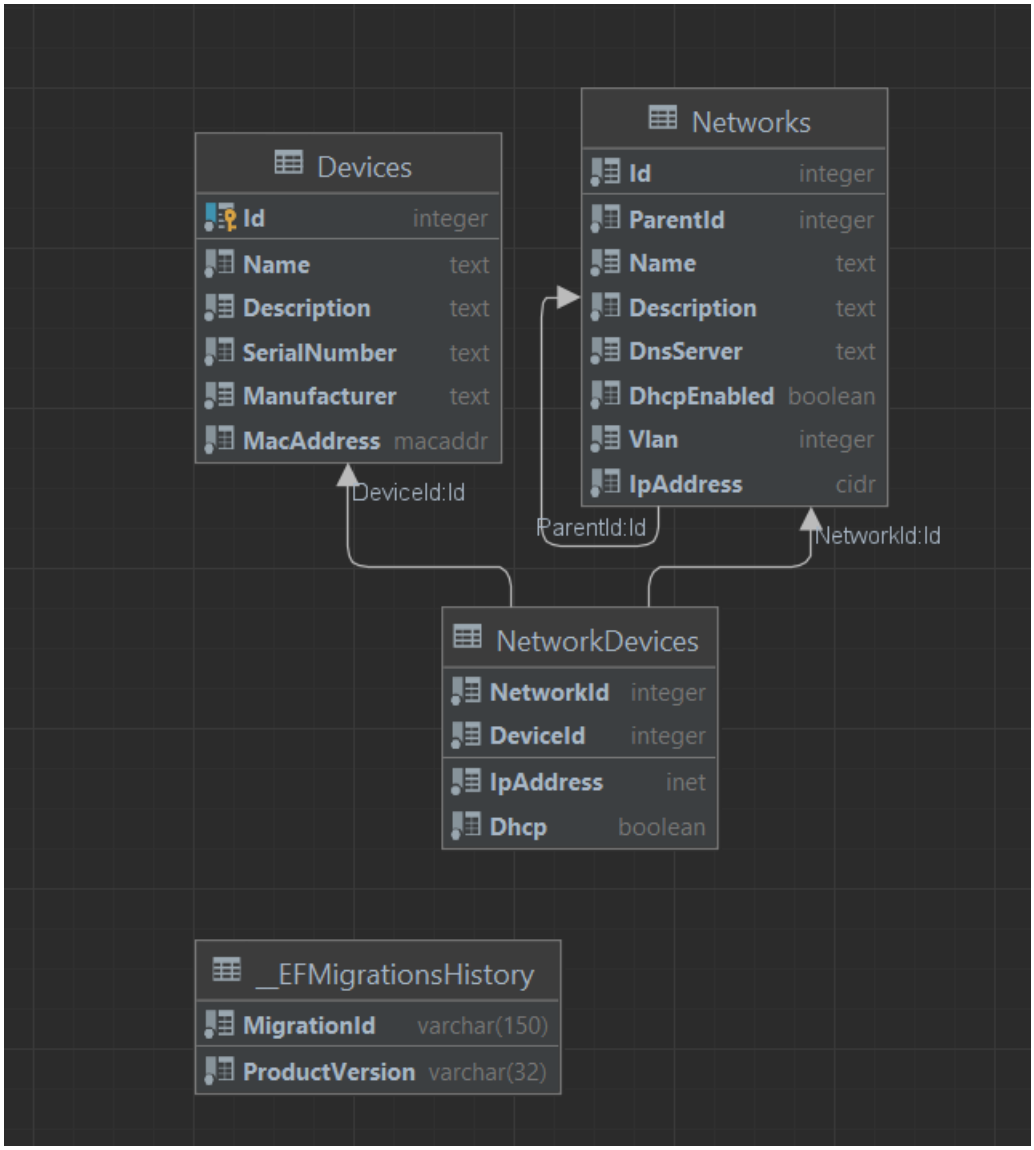

Рисунок 3 Схема public таблиць

Таблиця **Networks** – містить інформацію про IP-мережу. ЇЇ поля це:

- Внутрішній Id мережі

- Коротке ім’я мережі (офіс, квартира)
- Опис мережі (розташування, призначення, тощо)
- Адреса DNS-серверу
- Номер VLAN
- DHCP-прапор – чи підключена мережа до DHCP
- IP-Адреса мережі у форматі CIDR
- Id батьківської мережі

Таблиця Networks має рекурсивний зв’язок з собою. Таким чином вказується батьківська мережа, яка містить в собі поточну.

Таблиця Devices містить інформацію про пристрої:

- Внутрішній Id пристрою
- Коротке ім’я пристрою
- Опис пристрою (модель, зовнішній вигляд)
- Серійний номер
- Виробник
- MAC-адреса

Таблиця NetworkDevices пов’язує дві попередні таблиці. Вона являє собою пристрій, якому зарезервована адреса в мережі:

- Внутрішній Id мережі
- Внутрішній Id пристрою
- IP-адреса
- DHCP-прапор – чи отримувати IP-адресу автоматично

Перед додаванням сутностей до бази даних, проводиться перевірка на сервері щодо дотримання бізнес-вимог (Чи є мережа А підмережею В, тощо).

Auth схема містить всі таблиці, які відповідають за автентифікацію та авторизацію.

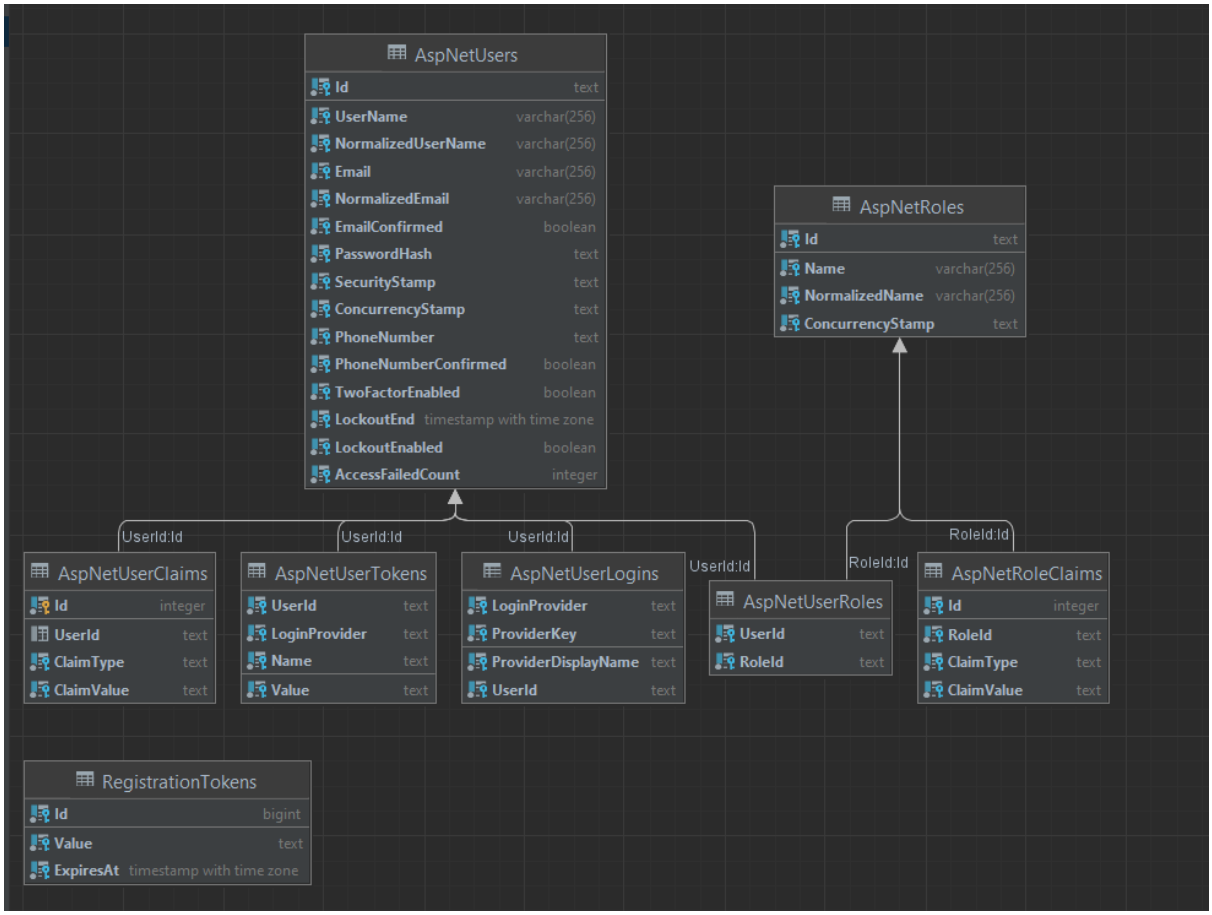

Рисунок 4 Схема auth таблиць

Ці таблиці побудовані на основі стандартних таблиць, які надаються фреймворком **ASP.NET Core Identity**. Це основний фреймворк для автентифікації та авторизації. Він надає велику кількість способів автентифікації (cookie, JWT, OAuth, тощо) та авторизації (за ролями чи за правами). У даній системі я обрав автентифікацію за JWT-токеном (вона зручніша у випадку роботи з API) та авторизацію за ролями.

Таблиця AspNetUsers містить усіх зареєстрованих користувачів системи. Основні її поля:

- Логін
- Захешований пароль

- Електронна адреса
- Номер телефона

Також вона містить нормалізовані версії багатьох полів, щоб логін не був чутливим до реєстру.

Таблиця **AspNetRoles** містить інформацію про ролі в системі. У даній системі я створив лише дві ролі – адміністратор та користувач. Користувач може переглядати та редагувати IPAM-інформацію, проте він не має доступу до персональних даних інших користувачів. Адміністратор має право доступу до абсолютно всіх даних в системі.

Таблиця **AspNetUserRoles** зв'язує користувача з певною роллю. Користувач може мати одночасно декілька ролей.

Back-end частина

Бек-енд частина складається з двох проектів. **Ipam.Data** містить інформацію про моделі даних та їх конфігурацію в **GraphQL**. Схеми даних в **GraphQL** дуже жорсткі. Ми не можемо позначити поле як null, якщо воно обов'язкове. Тому я створив окремі DTO, що використовуються при додаванні/видаленні об'єктів. Для автоматизації процесу маппінгу (копіювання полів одного об'єкту до іншого), я використовую бібліотеку Automapper. [15]

Проект **Ipam.Server** містить усю серверну логіку. Практично все API за винятком автентифікації реалізоване для **GraphQL**.

AuthController відповідає за автентифікацію користувача. Він перевіряє його логін та пароль і повертає згенерований JWT-токен.

В GraphQL існує поділення на запити даних (Query) та мутації (Mutation). Query – це запити на отримання даних, які не змінюють стан

системи. Mutations – це запити на змінні редагування даних, які змінюють стан.

Фреймворк Hot Chocolate підтримує це розділення, тому всі запити поділенні на два файли Query та Mutation.

У файлі Query.cs знаходяться запити на отримання даних. Вони отримують з контексту Entity Framework об’єкт для роботи з даними (IQueryable) та повертають його в контролері, не роблячи ніяких маніпуляцій з ними. Далі Hot Chocolate самостійно аналізує запит користувача та повертає шуканні дані.

Сама специфікація GraphQL не зазначає як сортувати чи пагінувати дані. Тому ці операції залежать від GraphQL фреймворку. У Hot Chocolate сортування, пагінація і фільтрація додаються до запитів за допомогою спеціальних атрибутів.

У файлі Mutation.cs знаходяться всі запити на створення, редагування та видалення об’єктів. Уся логіка винесена в окремі сервіси (у папці Services). Сервіси проводять маппінг, де потрібно, валідують дані, зберігають їх та повертають результат.

Під час валідації в сервісах, проводиться перевірка на дотримання інваріантів системи (чи мережа є підмережею, чи немає дублікатів IP-адрес в мережі, тощо).

Hot Chocolate використовує стандартні засоби автентифікації та авторизації **ASP.NET Core Identity**. У тім, на відміну від звичайного контролера, обмеження накладаються не на дії, а на самі дані. Ми можемо додавати атрибут Authorize, як до всього класу, так і до окремих. Як і випадку зі звичайною авторизацією в **ASP.NET Core**, ми можемо обмежувати доступ до лише автентифікованих, тих хто має певну роль, чи

тих хто має певне право (Claim). Також ми можемо комбінувати ці правила за допомогою політик (Policy).

Front-end частина

Фронтенд частина написана на мові Typescript. Це дуже зручно адже це дозволяє нам згенерувати типізований клієнт, який буде коректно використовувати API.

Схеми та операції **GraphQL** знаходяться в папці “graphql”. В самому проекті ці файли не використовуються, проте вони розміщенні для зручності розробки.

Усі файли клієнту розміщенні в папці “src”. Тут знаходиться файл “graphql.tsx”, який містить автоматично згенерований клієнт для роботи з API.

Дані JWT-токена зберігаються у localStorage. Роутер перевіряє наявність токена і перенаправляє на сторінку авторизації за його відсутності.

Усі сторінки розміщенні в папці “views”. Там містяться сторінки авторизації, створення та редагування мереж та пристроїв, списки усіх пристроїв та мереж.

Особливою є реалізація сторінки мереж. Оскільки мережі можуть мати підмережі, зручнішою для сприйняття є древовидна структура відображення. Я реалізував її за допомогою рекурсивних компоненту. Спочатку завантажуються мережі без батьків. При натисканні на кнопку більше, відображається дочірній компонент, того ж типу, що і батьківський з даними підмереж.

Також реалізовано систему асинхронних підказок (файл “Autocomplete.tsx”). При додаванні мережі, користувач може почати

вводити IP-адресу чи назву батьківської мережі і вона з’явиться серед результатів автодоповнення. Це реалізовано за допомогою спеціального запиту на пошук мереж на бек-енді. При кожній змінні вводу надсилається новий пошуковий запит. Аналогічна функціональність присутня при додаванні пристроїв до мережі.

У формах на додавання та редагування мереж і пристроїв присутня валідація. Валідація реалізована за допомогою регулярних виразів, які розпізнають як IPv4, так і IPv6 адресси. Для MAC-адрес також присутня маска вводу, яка ще більше спрощує ввід коректних даних.

Тестування та демонстрація роботи

GraphQL API Client

Перейшовши за посилання “/graphql” на бек-енді, ми побачимо спеціалізований клієнт для роботи з GraphQL API.

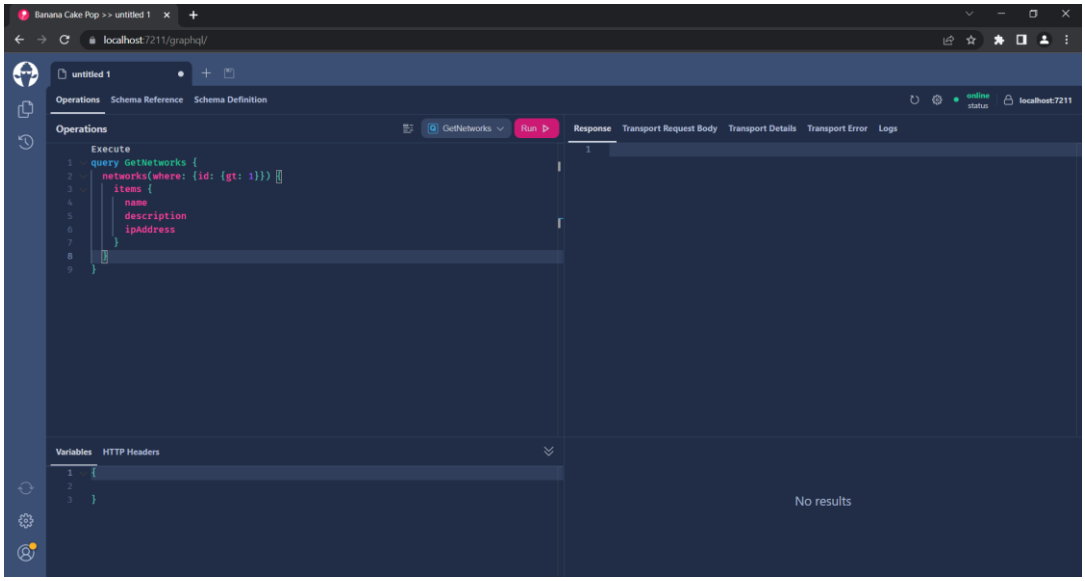

Рисунок 5 GraphQL клієнт

Тут ми можемо знайти GraphQL схему, яку ми можемо скопіювати для генерації API клієнта.

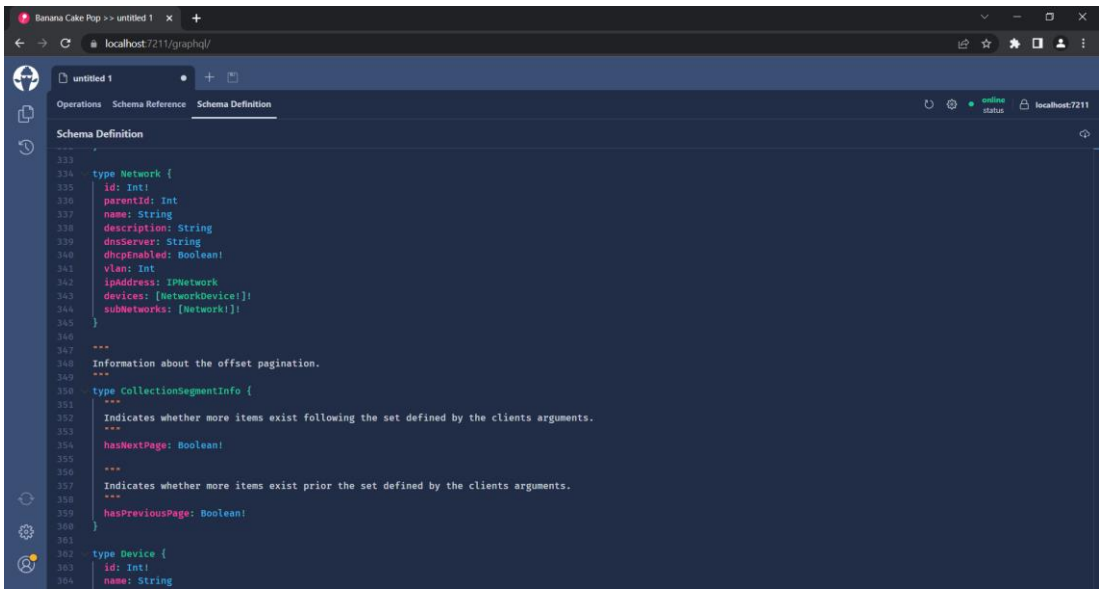

Рисунок 6 Схема IPAM-системи

На вкладці “Schema Reference” ми зможемо знайти опис усіх об’єктів даного API.

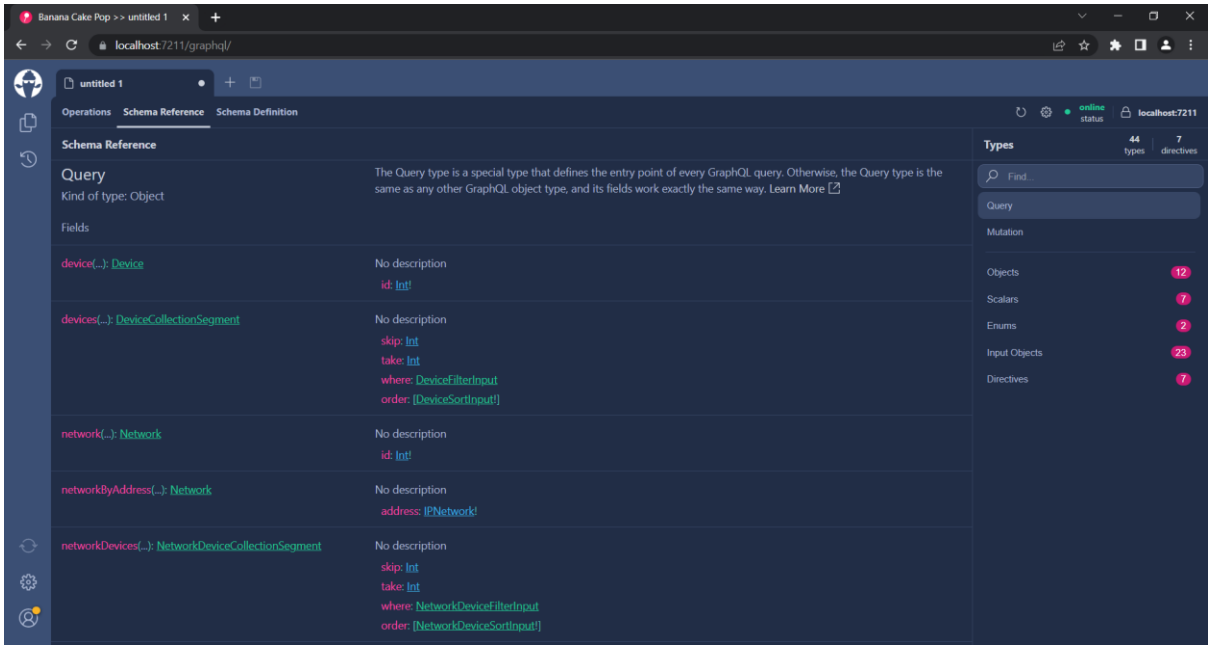

Рисунок 7 Опис усіх типів та параметрів

На вкладці “Operations” користувач може створювати свої власні запити. При цьому система буде усіляко підказувати коректні параметри.

Рисунок 8 Підказки при створенні запитів

Для виконання запитів потрібно автентифікуватись. Для цього в налаштуваннях клієнта можна передати JWT-токен.

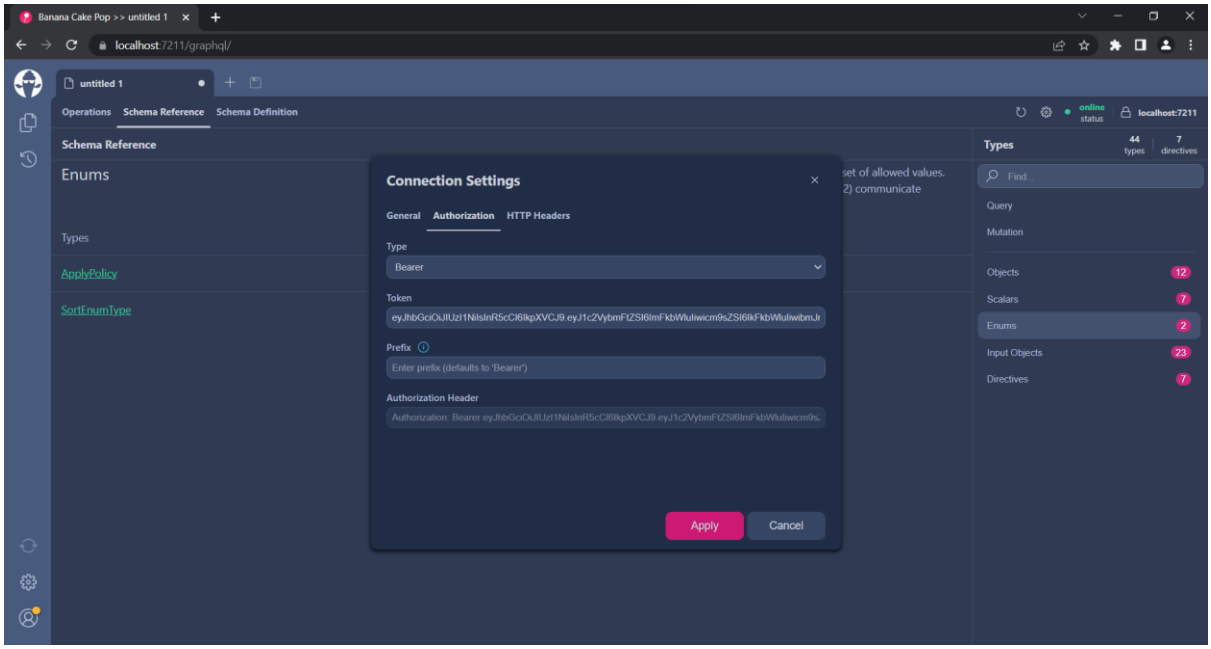

Рисунок 9 Автентифікація за допомогою JWT-токена

Веб-додаток

При першому вході користувач потрапляє на сторінку автентифікації.

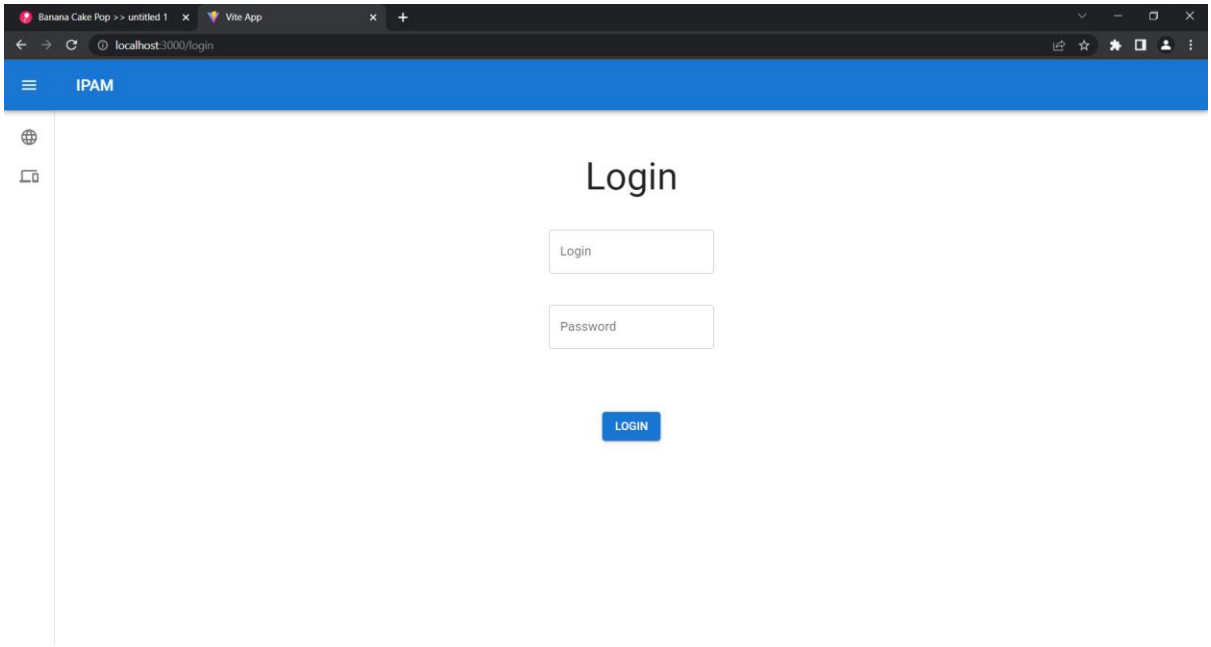

Рисунок 10 Сторінка автентифікації

Передавши коректні значення логіну та паролю (admin, Password123$), користувач перенаправляється до списку мереж.

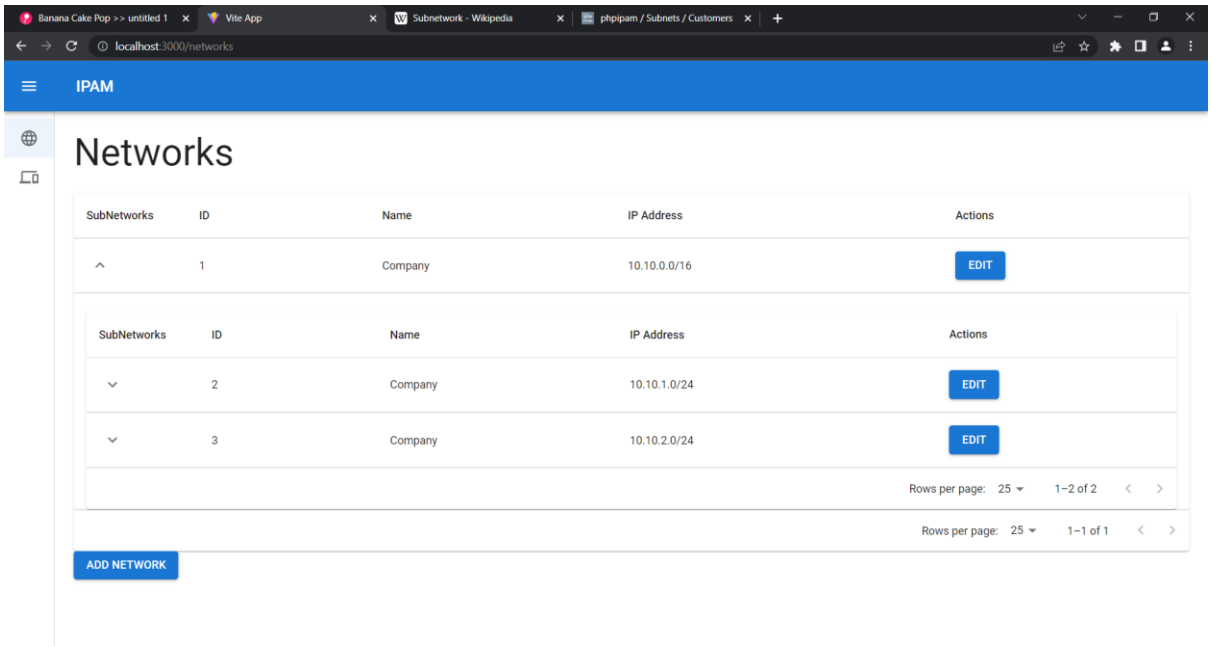

Рисунок 11 Сторінка списку мереж

Мережі можна відсортовувати, наприклад, за IP-адресою.

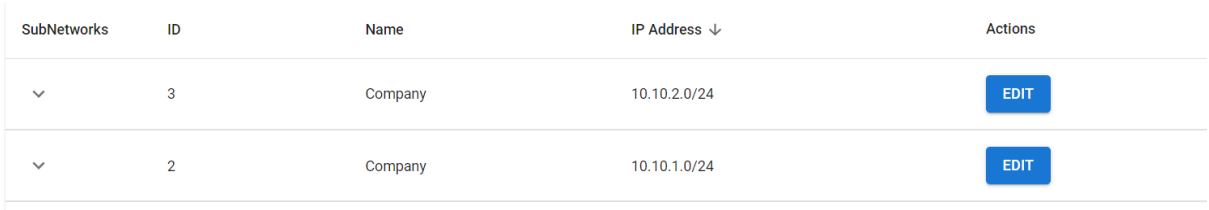

Рисунок 12 Сортування за IP-адресою

Натиснувши кнопку “Add Network”, користувач переходить на сторінку створення мережі.

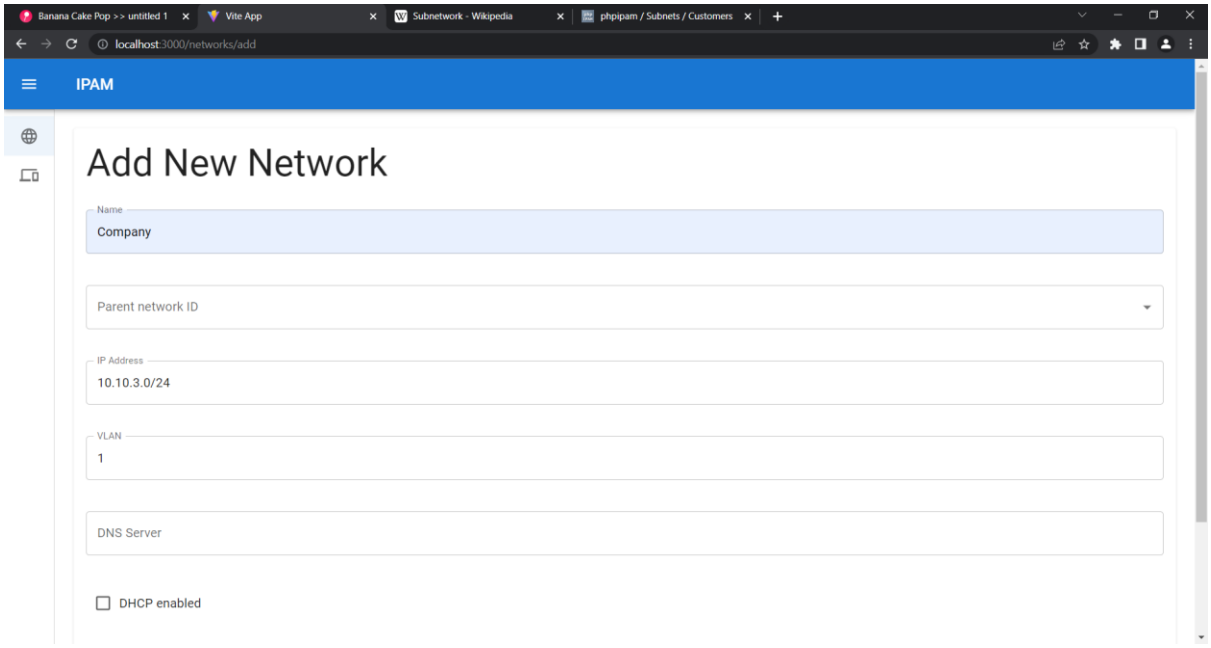

Рисунок 13 Вікно створення нової мережі

У полі “Parent network ID” працює розумне автодоповнення. Користувач починає вводити IP-адресу мережі і бачить пропозиції серед результатів.

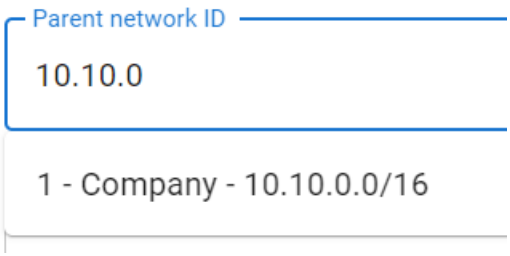

Рисунок 14 Автодоповнення батьківських мереж

При натисканні на клавішу “Save”, мережа зберігається та користувача одразу перенаправляє на сторінку її редагування, яка аналогічна за виглядом і функціоналом.

При переході на вкладку “Devices”, ми отримуємо список всіх пристроїв.

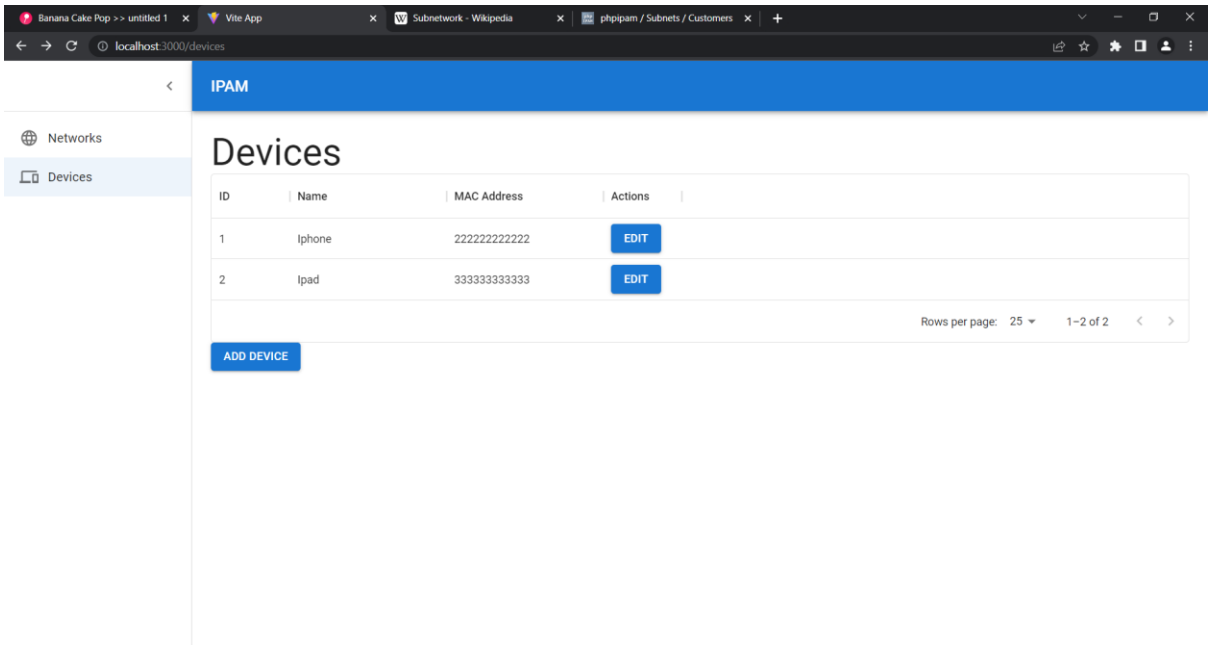

Рисунок 15 Список пристроїв

Як і з мережами, його можна відсортувати.

При натисканні кнопки “Add device” відкривається вікно додавання пристрою.

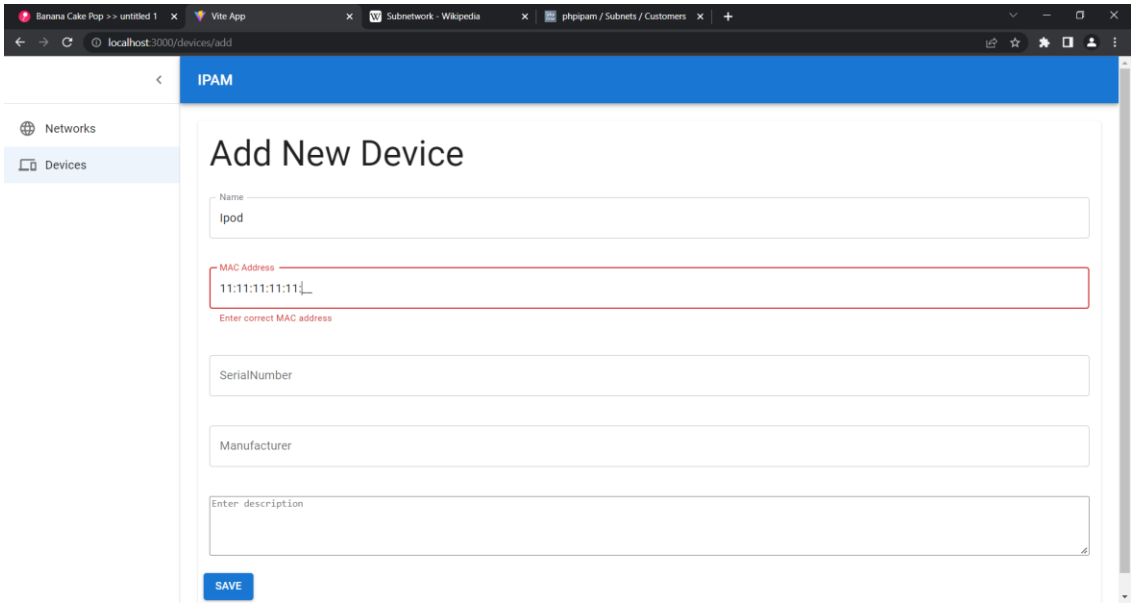

Рисунок 16 Вікно додавання пристроїв

У даній формі, аналогічно з мережами, присутня валідація полів.

Після того, як ми додали пристрій, ми можемо додати його до мережі. Для цього ми повернемось до меню редагування мережі та прокрутимо його вниз.

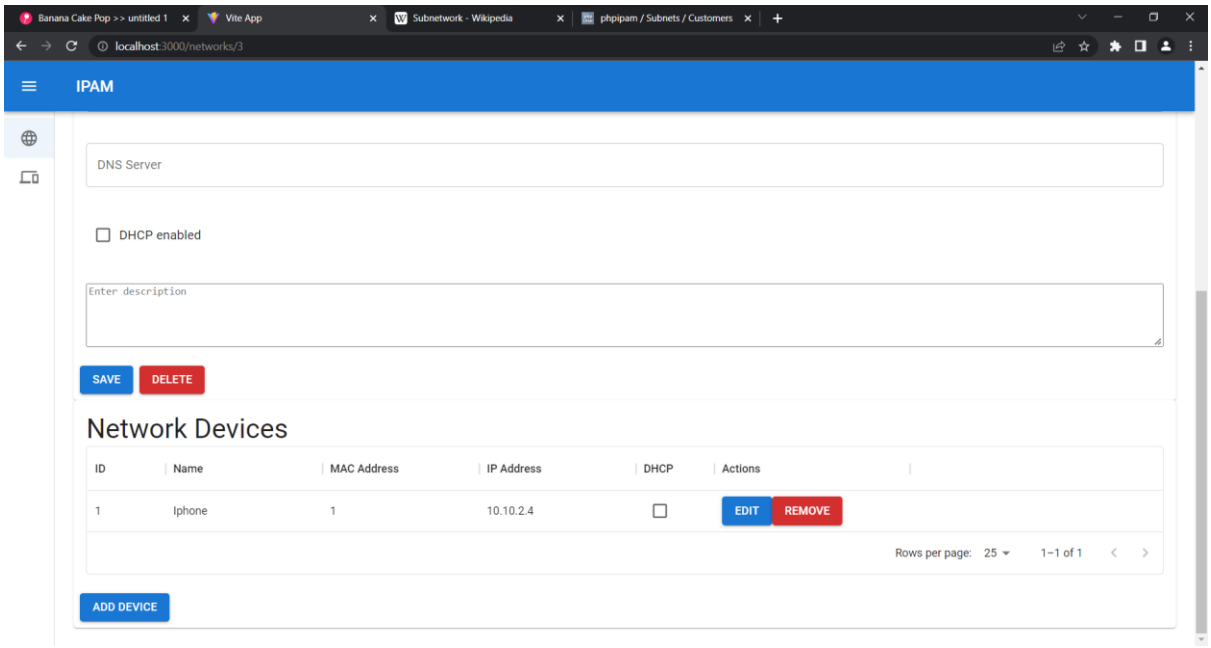

Рисунок 17 Список пристроїв в мережі

Натиснувши на кнопку “Add device” відкриється меню додавання пристрою до мережі.

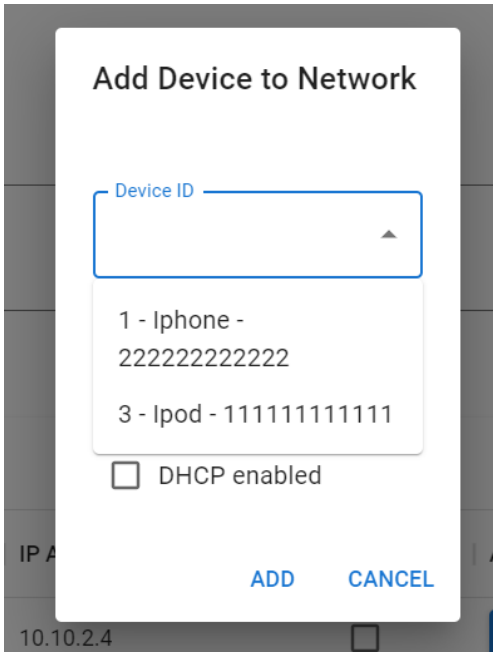

Рисунок 18 Додавання пристрою до мережі

Висновки

У рамках даної курсової роботи було описано основне призначення IPAM-систем і їх основний функціонал.

Також була створена проста система керуванням адресним простором. Її функціонал дуже обмежений, у порівнянні зі справжніми системами, проте вона має набагато простіший користувацький інтерфейс. Тож, на мою думку, її має вистачити для обліку невеликої мережі.

Список Літератури

[1] «Internet of Things statistics for 2022 - Taking Things Apart». https://dataprot.net/statistics/iot-statistics/

[2] «What is IPAM? It’s crucial for managing IP addresses». https://bluecatnetworks.com/glossary/what-is-ipam/

[3] «Dynamic Host Configuration Protocol (DHCP)». [Online]. Доступний у: https://docs.microsoft.com/en-us/windows-server/networking/technologies/dhcp/dhcp-top

[4]. *NET - Official Site*. [Online]. Доступний у: https://dotnet.microsoft.com/en-us/

[5] «Performance improvements in ASP.NET Core 6». https://devblogs.microsoft.com/dotnet/performance-improvements-in-aspnet-core-6/

[6] *React - Official Site*. [Online]. Доступний у: https://uk.reactjs.org/

[7] *MUI - Official Site*. [Online]. Доступний у: https://mui.com/

[8] *Material Design - Official Guidelines*. [Online]. Доступний у: https://material.io/design

[9] «How to GraphQL in Kotlin and Micronaut and create a single endpoint for access to microservices’ APIs». https://romankudryashov.com/blog/2020/02/how-to-graphql/

[10] «Netbox - GraphQL API Overview». [Online]. Доступний у: https://docs.netbox.dev/en/stable/graphql-api/overview/

[11] «PostgreSQL - Network Address Types». [Online]. Доступний у: https://www.postgresql.org/docs/current/datatype-net-types.html

[12] «Entity Framework Core». [Online]. Доступний у: https://docs.microsoft.com/en-us/ef/core/

[13] *Apollo GraphQL*. [Online]. Доступний у: https://www.apollographql.com/

[14] *GraphQL Code Generator*. [Online]. Доступний у: https://www.graphql-code-generator.com/

[15] «Automapper - Guide». [Online]. Доступний у: https://docs.automapper.org/en/stable/Getting-started.html

Додаток А
Приклад схеми GraphQL

```
type Network {
  id: Int!
  parentId: Int
  name: String
  description: String
  dnsServer: String
  dhcpEnabled: Boolean!
  vlan: Int
  ipAddress: IPNetwork
  devices: [NetworkDevice!]!
  subNetworks: [Network!]!
}
```

Додаток Б
Приклад запитів GraphQL

```
query GetNetworks {
  networks(where: {id: {gt: 1}}) {
    items {
      name
      description
      ipAddress
    }
  }
}

query GetDevices {
  devices {
    items{
      id
    }
  }
}
```

Додаток В
Сервіс NetworkService

```
using AutoMapper;
using HotChocolate.Execution;
using Ipam.Data.Dto;
using Ipam.Data.Models;
using Ipam.Extensions;
using Microsoft.EntityFrameworkCore;

namespace Ipam.Services;

public interface INetworkService
{
    Task<Network> AddNetwork(NetworkInfo network);
    Task<Network> UpdateNetwork(int id, NetworkInfo network);
    IQueryable<Network> SearchNetworks(string searchQuery);
    Task<Network> DeleteNetwork(int id);
}

public class NetworkService : INetworkService
{
    private readonly IpamContext _context;
    private readonly IMapper _mapper;

    public NetworkService(IpamContext context, IMapper mapper)
    {
        _context = context;
        _mapper = mapper;
    }

    public async Task<Network> AddNetwork(NetworkInfo network)
    {
        var entity = _mapper.Map<NetworkInfo, Network>(network);
        await ValidateNetwork(entity);
        var result = await _context.Networks.AddAsync(entity);
        await _context.SaveChangesAsync();
        return result.Entity;
    }

    public async Task<Network> UpdateNetwork(int id, NetworkInfo network)
    {
        var entity = _mapper.Map<NetworkInfo, Network>(network);
        entity.Id = id;
        await ValidateNetwork(entity);
        var result = _context.Networks.Update(entity);
        await _context.SaveChangesAsync();
        return result.Entity;
    }

    public async Task<Network> DeleteNetwork(int id)
    {
```

```
        var network = await _context.Networks.FirstAsync(d => d.Id == id);
        _context.Networks.Remove(network);
        await _context.SaveChangesAsync();
        return network;
    }

    public IQueryable<Network> SearchNetworks(string searchQuery)
    {
        var search = _context.Networks.FromSqlRaw(@"
        select * from ""Networks""
        where ""Name"" ilike {0}
        or text(""Id"") ilike {0}
        or text(""IpAddress"") ilike {0}
        ", $"%{searchQuery}%");
        return search.Take(50);
    }

    private async Task ValidateNetwork(Network network)
    {
        if (network.ParentId != null)
        {
            var parent = await _context.Networks.FirstAsync(n => n.Id ==
network.ParentId);
            if (!parent.IpAddress.Contains(network.IpAddress))
                throw new QueryException(ErrorBuilder.New()
                    .BadInput("Parent doesn't contain Subnetwork")
                    .Build());
        }

        var neighbours = await _context.Networks
            .Where(n => n.ParentId == network.ParentId && n.Id !=
network.Id)
            .Select(n => n.IpAddress)
            .ToListAsync();

        if (neighbours.Any(ipNetwork =>
ipNetwork.Overlap(network.IpAddress)))
            throw new QueryException(ErrorBuilder.New()
                .BadInput("Networks overlap")
                .Build());
    }
}
```

Додаток Г
Компонент списків мереж

```
import React, {useState} from "react";
import { GridSortModel} from "@mui/x-data-grid/models/gridSortModel";
import { useChildNetworksQuery} from "../../graphql";
import {
    Box,
    Button,
    Collapse, IconButton, Paper,
    Table, TableBody,
    TableCell,
    TableContainer,
    TableHead, TablePagination, TableRow,
    TableSortLabel,
    Typography
} from "@mui/material";
import KeyboardArrowDownIcon from '@mui/icons-
material/KeyboardArrowDown';
import KeyboardArrowUpIcon from '@mui/icons-
material/KeyboardArrowUp';
import Loading from "../../components/Loading";
import {getOrder} from "../../utils";
import {Link} from "react-router-dom";

type NetworksProps = {
    parentId?: number
};

interface HeadCell {
    id: string;
    label: string;
    width?: number;
}

const headCells: readonly HeadCell[] = [
    {
        id: 'subNetworks',
        label: 'SubNetworks',
        width: 100
    },
    {
        id: 'id',
        label: 'ID',
    },
    {
        id: 'name',
```

```
        label: 'Name'
    },
    {
        id: 'ipAddress',
        label: 'IP Address'
    }
];

export default function Networks({parentId}: NetworksProps) {
    const [page, setPage] = useState(0);
    const [pageSize, setPageSize] = useState(25);
    const [orderBy, setOrderBy] = useState<GridSortModel>([]);

    const result = useChildNetworksQuery({variables: {parentId,
order: getOrder(orderBy)}});
    const data = result.data?.networks?.items;

    const [isOpenRows, setOpenRows] = useState(data?.map(() => false)
?? []);

    const onSortUpdate =
        (property: string) => (event: React.MouseEvent<unknown>) => {
            if (!orderBy[0] || orderBy[0].field !== property) {
                setOrderBy([{field: property, sort: 'asc'}])
            } else if (orderBy[0].sort === 'desc') {
                setOrderBy([])
            } else {
                setOrderBy([{field: property, sort: 'desc'}])
            }
        };

    return <Box component={Paper} mt={3}><TableContainer>
        <Table>
            <TableHead>
                <TableRow>
                    {headCells.map(headCell =>
                        <TableCell
                            key={headCell.id}
                            sortDirection={orderBy[0]?.field ===
headCell.id ? orderBy[0].sort! : false}
                            width={headCell.width}
                        >
                            <TableSortLabel
                                active={orderBy[0]?.field ===
headCell.id}

                                direction={orderBy[0]?.sort!}
                                onClick={onSortUpdate(headCell.id)}
                            >
```

```
                                {headCell.label}
                            </TableSortLabel>
                        </TableCell>)}
                    <TableCell key="actions">Actions</TableCell>
                </TableRow>
            </TableHead>
            <TableBody>
                {data ? data.map((i, index) => {
                    const isOpen = isOpenRows[index];
                    const setOpen = () => {
                        const openRows = [...isOpenRows];
                        openRows[index] = !openRows[index];
                        setOpenRows(openRows);
                    }
                    return <>
                        <TableRow key={i.id}>
                            <TableCell key="subnetworks"><IconButton
                                aria-label="expand row"
                                size="small"
                                onClick={() => setOpen()}
                            >
                                {isOpen ? <KeyboardArrowUpIcon/> :
<KeyboardArrowDownIcon/>}
                            </IconButton></TableCell>
                            <TableCell key="id">{i.id}</TableCell>
                            <TableCell
key="name">{i.name}</TableCell>
                            <TableCell
key="ipAddress">{i.ipAddress}</TableCell>
                            <TableCell key="actions"><Button
variant="contained" color="primary" component={Link}
                                                                to={`/ne
tworks/${i.id}`}>Edit</Button></TableCell>
                        </TableRow>
                        <TableRow key={-i.id}><TableCell colSpan={50}
                                                         style={{padd
ingBottom: 0, paddingTop: 0}}><ChildNetworks
                            isOpen={isOpen}
                            parentId={i.id}/></TableCell></TableRow>
                    </>
                }) : <TableRow key="loading"><TableCell
colSpan={50}><Loading/></TableCell></TableRow>}
            </TableBody>
        </Table>
    </TableContainer>
        <TablePagination
            rowsPerPageOptions={[10, 25, 50]}
            component="div"
```

```
            count={data?.length ?? 0}
            rowsPerPage={pageSize}
            page={page}
            onPageChange={(_, page) => setPage(page)}
            onRowsPerPageChange={event =>
setPageSize(parseInt(event.target.value, 10))}
        />
    </Box>
}

function ChildNetworks({isOpen, ...props}: NetworksProps & { isOpen:
boolean }) {
    return <Collapse in={isOpen} timeout={"auto"} mountOnEnter>
        <Networks {...props} />
    </Collapse>
}
```

Додаток Д

Форма додавання пристрою

```
import React from 'react';
import {Stack, TextareaAutosize, TextField} from "@mui/material";
import {TextValidator} from "react-material-ui-form-validator";
import Loading from "../../components/Loading";
import {DeviceInfoInput} from "../../graphql";
import ReactInputMask from 'react-input-mask';
import {css} from "@emotion/react";

type DeviceFormProps = {
    device?: DeviceInfoInput,
    setName: (_: string) => void,
    setSerial: (_: string) => void,
    setManufacturer: (_: string) => void,
    setDescription: (_: string) => void,
    setMacAddress: (_: string) => void,
}

const macRegex = "^[0-9A-Fa-f]{1,2}([\\.:-])(?:[0-9A-Fa-
f]{1,2}\\1){4}[0-9A-Fa-f]{1,2}$";
const maskChars = {
  'A': '[0-9A-Fa-f]'
};

export default function DeviceForm({device, setName, setDescription,
setMacAddress, setSerial, setManufacturer}: DeviceFormProps) {
    if (device) {

        return <Stack spacing={5}>
            <TextField label="Name" id="name" value={device.name}
                       onChange={e =>
setName(e.currentTarget.value)}></TextField>
            <ReactInputMask
                mask="AA:AA:AA:AA:AA:AA"
                formatChars={maskChars}
                value={device.macAddress}
                onChange={e => setMacAddress((e.target as
any).value)}
                // className={this.props.classes.textField}
            >{() => <TextValidator name={"mac"} label="MAC Address"
id="mac" value={device.macAddress}
                                   validators={['required',
`matchRegexp:${macRegex}`]} css={css`width: 100%`}
                                   errorMessages={['This field is required',
'Enter correct MAC address']}
```

```
            ></TextValidator>) as any}
            </ReactInputMask>
            <TextField label="SerialNumber" id="name"
value={device.serialNumber}
                       onChange={e =>
setSerial(e.currentTarget.value)}></TextField>
            <TextField label="Manufacturer" id="name"
value={device.manufacturer}
                       onChange={e =>
setManufacturer(e.currentTarget.value)}></TextField>
            <TextareaAutosize placeholder="Enter description"
minRows={5} id="description"
                              value={device.description ?? ""}
                              onChange={e =>
setDescription(e.currentTarget.value)}/>

        </Stack>;
    } else return <Loading/>;
}
```