КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від ___ червня 2024 року, протокол № ___.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____ Ігор АНІСІМОВ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
на тему:
«АНАЛІЗАТОР WIFI-МЕРЕЖ»

Виконав:
студент 4-го курсу
денної форми навчання
спеціальності 172 - Телекомунікації та радіотехніка
ОПП «Інформаційна безпека телекомунікаційних систем і мереж»
Мостовий Кирил Леонідович _____

Науковий керівник:
кандидат фіз.-мат. наук, асистент
Котов Михайло Миколайович _____

Рецензент:
кандидат фіз.-мат. наук, доцент
Загороднюк Сергій Петрович _____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент Кирил МОСТОВИЙ _____

Київ – 2024

РЕФЕРАТ

Дипломна робота: 99 стор., 28 рис., 1 додаток, 13 джерел.

АНАЛІЗАТОР, WI-FI МЕРЕЖІ, МОНІТОРИНГ, СКАНУВАННЯ МЕРЕЖ, MAUI, .NET CORE, C#, WINDOWS

Об'єкт розробки: програмний аналізатор для діагностики та аналізування стану WiFi мереж.

Мета роботи: створення зручного інструменту для огляду та аналізу параметрів безпровідних мереж, який поєднує найкращі характеристики існуючих рішень та вдосконалює їх, додаючи новий функціонал.

Розроблено додаток для операційних систем Windows з графічним інтерфейсом, що дозволяє користувачам сканувати наявні WiFi мережі, переглядати їх параметри та рівні сигналу, а також проводити базовий аналіз безпеки мереж.

Створений аналізатор є зручним інструментом як для домашніх користувачів для оптимізації домашньої WiFi мережі, так і для системних адміністраторів компаній для діагностики корпоративних безпровідних мереж.

ЗМІСТ

ВСТУП

З розвитком технологій бездротового зв'язку WiFi-мережі стали невід'ємною частиною повсякденного життя, забезпечуючи зручний доступ до Інтернету в будинках, офісах, громадських місцях та навіть на вулиці. Зі зростанням популярності WiFi збільшується кількість користувачів, що створює необхідність у постійному моніторингу та аналізі характеристик цих мереж.

У зв'язку з актуальністю та важливістю цієї теми, дана робота присвячена розробці додатку для аналізу WiFi-мереж. Основною метою роботи є створення інструменту, який дозволить користувачам зручно та ефективно моніторити (відстежувати) WiFi-мережі. Цей додаток буде корисним для адміністраторів мереж, технічних спеціалістів, а також для звичайних користувачів.

Розробка буде зосереджена на врахуванні кращих характеристик існуючих рішень для аналізу WiFi-мереж, водночас вдосконалюючи їх та додаючи нові функціональні можливості. У рамках цієї роботи буде проведено огляд існуючих додатків, визначено їхні переваги та недоліки, спроектовано архітектуру та інтерфейс користувача, реалізовано основні функціональні можливості додатку та проведено тестування з аналізом отриманих результатів. Даний додаток, базуючись на кращих характеристиках існуючих рішень, буде простим у використанні, доступним для Windows, безкоштовним та відкритим, а також включатиме такі функції, як відображення приблизної відстані до мережі, фільтрування за частотою 6 ГГц та вимірювання швидкості завантаження в даний момент часу, яких немає в подібних додатках.

1. ОСНОВИ WIFI-МЕРЕЖ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ ЇХ АНАЛІЗУ

Цей розділ є оглядом основ WiFi-мереж, які є основою бездротового зв'язку, та досліджує основні поняття і принципи функціонування WiFi-мереж. Він описує основні характеристики бездротового зв'язку, такі як частотний діапазон, типи мереж та їх характеристики, пояснює роль різних стандартів WiFi та їх вплив на швидкість і надійність зв'язку. Також розглядаються існуючі інструменти для аналізу WiFi-мереж, проводиться порівняльний аналіз їх функціональних можливостей, визначаються переваги та недоліки, що допомагає у виборі найбільш відповідного інструменту для конкретних завдань з аналізу та оптимізації WiFi-мереж.

1. 1. Типи WiFi-мереж та їх характеристики

WiFi мережа - це форма бездротової локальної мережі (WLAN), що базується на стандартах IEEE 802.11 і використовує радіосигнали для з'єднання пристроїв. Завдяки WiFi мережам, пристрої, такі як комп'ютери, смартфони, планшети та інші, можуть з'єднуватися з Інтернетом або між собою без необхідності фізичного дротового з'єднання.

WiFi мережі можуть бути відкритими, де будь-який пристрій може приєднатися без пароля, або захищеними паролем. Крім того, вони можуть використовувати різні методи шифрування, такі як WEP, WPA, WPA2 або WPA3, для захисту передачі інформації.

WiFi мережі застосовуються в різних областях, включаючи домашнє використання, офіси, громадські місця, такі як кафе та бібліотеки, а також навіть у міських середовищах для надання доступу до Інтернету.

WiFi-мережі поділяються на кілька типів за різними характеристиками, такими як:

- Тип шифрування:

- WEP: Старий, не безпечний тип шифрування WiFi, використовує ключі 40 або 104 біти [1].
- WPA: З'явився в 2003 році як перехідний варіант до WPA2, використовує TKIP для шифрування [2].
- WPA2: Найбезпечніший тип шифрування WiFi, з використанням AES для захисту даних [3].
- WPA3: Найновіший стандарт безпеки, введений у 2018 році, з удосконаленнями для захисту паролів та інших функцій [4].
- TKIP: Використовується разом із WPA для заміни WEP, але вже не вважається безпечним через подібність до WEP [5].
- AES: Стандарт шифрування, розроблений у 2001 році, використовує симетричний алгоритм з одним ключем для шифрування та розшифрування даних [6].

- Тип аутентифікації:
  - Open(Відкритий): Найменш безпечний тип аутентифікації, не потребує пароля для підключення до мережі.
  - Shared Key: Використовує один спільний пароль для всіх користувачів мережі, базується на протоколі WEP.
  - WPA/WPA2-PSK: Використовує попередньо спільний ключ для всіх користувачів мережі, забезпечує безпеку за допомогою стандартів шифрування.
  - WPA3: Найновіший стандарт безпеки, використовує SAE для більш сильного захисту від спроб вгадування паролів, має режими WPA3-Personal та WPA3-Enterprise [7].
  - EAP: Протокол аутентифікації для встановлення взаємної аутентифікації та безпечного обміну облікових даних [8].
  - WPA3-OWE: Надає безпечне підключення для клієнтів без введення паролю, подібно до SAE.
- Діапазон:

- 2.4 ГГц: Часто використовується в домашніх та офісних мережах. Хоча має ширший діапазон, він може стикатися з перешкодами від інших пристроїв, таких як мікрохвильові печі та бездротові телефони.
- 5 ГГц: Менш схильний до перешкод і часто використовується для вимогливих мереж, які потребують великої пропускної здатності, наприклад, для потокового відео та ігор.
- 6 ГГц: Новий діапазон з широким спектром каналів і меншими перешкодами. Ідеальний для високошвидкісних мереж, таких як потокове відео та ігри.

- Стандарти бездротового зв'язку [9]:
  - WiFi 1 (IEEE 802.11a): Перший стандарт на частоті 5 ГГц, зі швидкістю до 54 Мбіт/с.
  - WiFi 2 (IEEE 802.11b): Популярний на частоті 2,4 ГГц, до 11 Мбіт/с.
  - WiFi 3 (IEEE 802.11g): Та ж сама частота, що й b, але до 54 Мбіт/с.
  - WiFi 4 (IEEE 802.11n): Підтримує обидві частоти, до 600 Мбіт/с.
  - WiFi 5 (IEEE 802.11ac): Сучасний стандарт на обох частотах, до 1,3 Гбіт/с.
  - WiFi 6 (IEEE 802.11ax): Нова версія, що збільшує пропускну здатність до 10-12 Гбіт/с.
  - WiFi 7 (IEEE 802.11be): Майбутній стандарт із передачею даних до 40 Гбіт/с та розширеним діапазоном для підвищення ефективності та надійності мережі.

1. 2. Основні параметри WiFi-мереж

Основні параметри WiFi-мереж включають:

- SSID: Унікальне ім'я мережі для її ідентифікації, може бути налаштоване власником.

- Пароль: Захищає мережу від несанкціонованого доступу, рекомендується складний та надійний.
- Протокол: Визначає, як дані передаються по мережі, наприклад, стандарти IEEE 802.11.
- Тип безпеки: Використовується для захисту даних, наприклад, WEP, WPA, WPA2, WPA3.
- Діапазон мережі: Частота роботи мережі, така як 2,4 ГГц, 5 ГГц та 6 ГГц.
- Канал: Конкретна частота, на якій передаються дані в мережі.
- IP-адреса та DNS: Визначають, як пристрій підключається до Інтернету через WiFi.
- MAC-адреса: Унікальний ідентифікатор WiFi-адаптера пристрою.
- Швидкість передачі даних: Максимальна швидкість передачі даних по мережі.
- Сила сигналу WiFi: Впливає на якість з'єднання і вимірюється в дБм.

1.3. Огляд існуючих інструментів для аналізу WiFi-мереж

Існує багато інструментів для аналізу WiFi-мереж, які допомагають користувачам дізнатися більше про свої WiFi-мережі, ідентифікувати та вирішувати проблеми, а також оптимізувати продуктивність. Ось два з найпопулярніших десктопних додатків:

1. **WiFi Analyzer** [10]: Цей безкоштовний та відкритий код інструмент доступний для Windows, macOS та Linux. Він пропонує широкий спектр функцій, включаючи:

- Сканування та відображення всіх доступних WiFi-мереж
- Перегляд інформації про мережу, такі як SSID, BSSID, канал, частота, тип шифрування, рівень сигналу, тощо
- Виявлення та аналіз мережевих проблем, таких як перешкоди та конфлікти каналів
- Створення звітів про мережу

- Моніторинг продуктивності WiFi-мережі

2. **inSSIDer** [11]: Цей безкоштовний інструмент доступний для Windows та macOS. Він пропонує більш просунуті функції, ніж WiFi Analyzer, включаючи:

- Детальне відображення спектра WiFi

1.4. Порівняльний аналіз функціональних можливостей

WiFi Analyzer та inSSIDer - це два популярних інструменти для аналізу WiFi-мереж. Ось порівняльний аналіз їхніх функцій:

- Платформа: WiFi Analyzer підтримує Windows, macOS та Linux, тоді як inSSIDer працює на Windows та macOS.
- Ціна: Обидва інструменти є безкоштовними.
- Сканування WiFi-мереж: Обидва інструменти можуть сканувати WiFi-мережі.
- Відображення інформації про мережу: Обидва інструменти можуть відображати інформацію про мережу.
- Виявлення та аналіз мережевих проблем: Обидва інструменти можуть виявляти та аналізувати мережеві проблеми.
- Створення звітів про мережу: Обидва інструменти можуть створювати звіти про мережу.
- Моніторинг продуктивності WiFi-мережі: Обидва інструменти підтримують моніторинг продуктивності WiFi-мережі.
- Детальне відображення спектра WiFi: Тільки inSSIDer може надавати більш детальне відображення спектра WiFi.

Деякі приклади користувацького інтерфейсу для додатку WiFi Analyzer зображено на рисунках 1.1, 1.2 та 1.3, для додатку inSSIDer на рисунках 1.4 та 1.5.

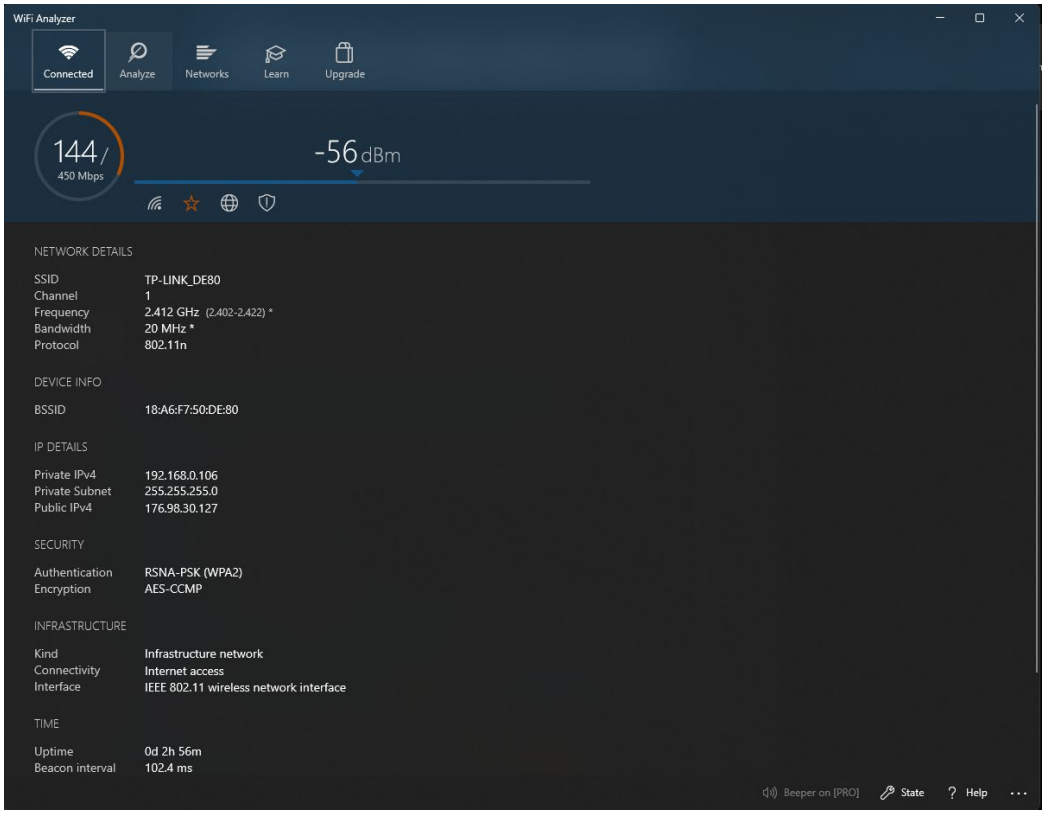

Рисунок 1.1 – Головна (Connected) сторінка WiFi Analyzer додатку.

Рисунок 1.2 – Analyze сторінка WiFi Analyzer додатку.

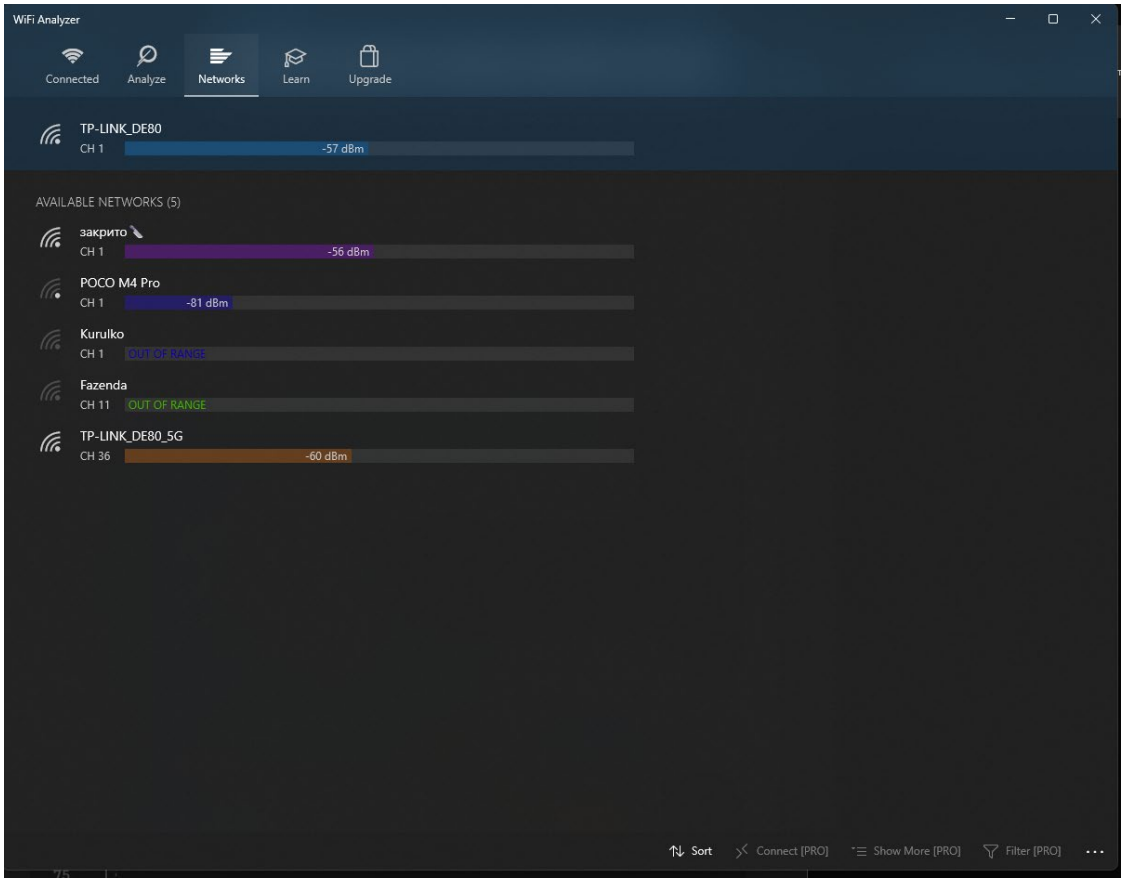

Рисунок 1.3 – Networks сторінка WiFi Analyzer додатку.

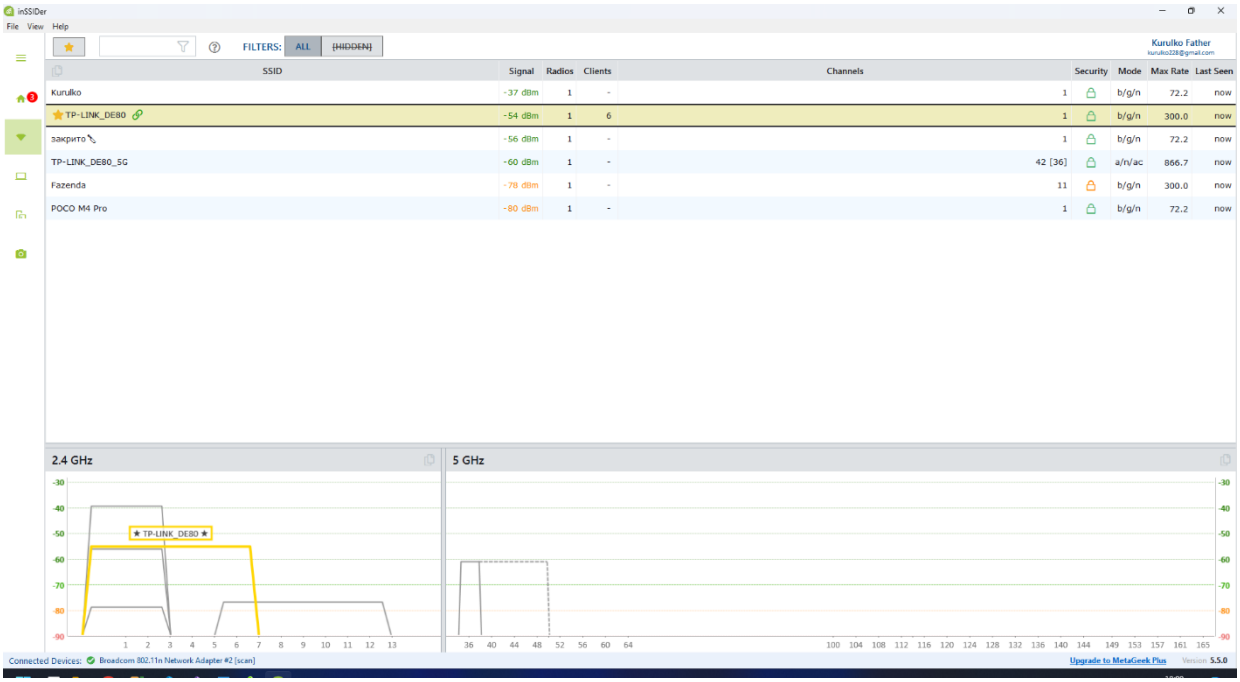

Рисунок 1.4 – Головна сторінка inSSIDer додатку.

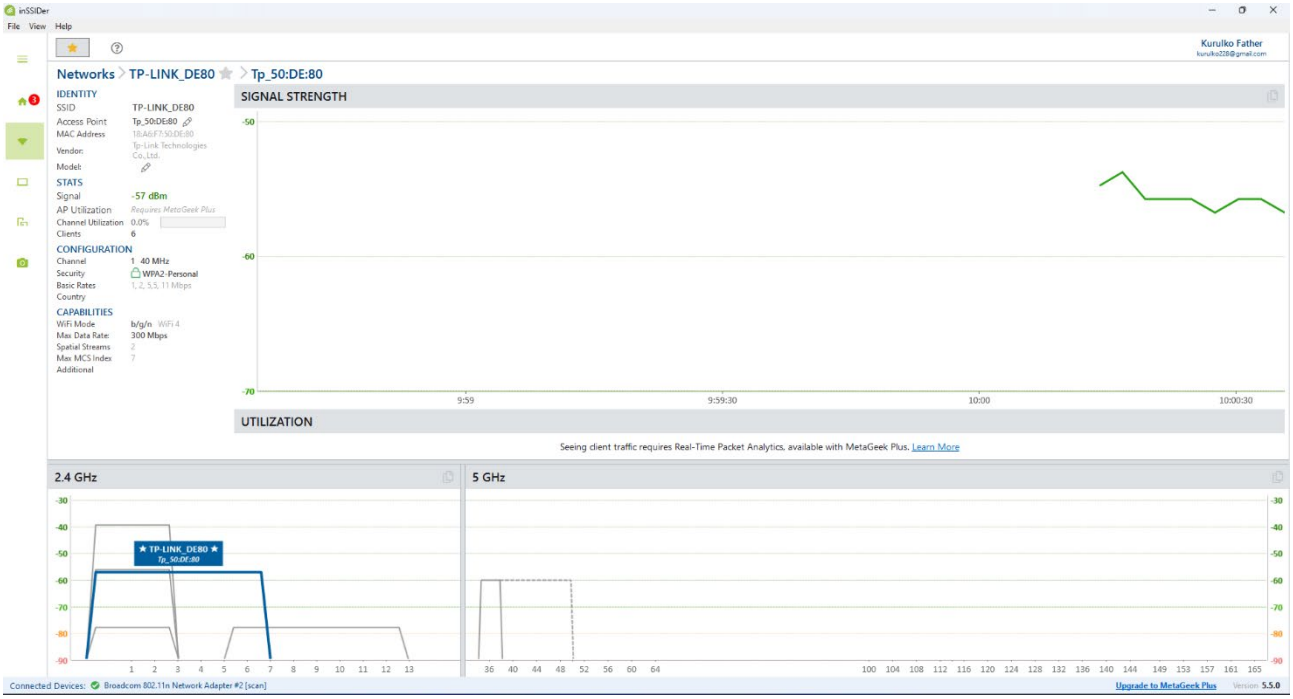

Рисунок 1.5 – Детальна інформація про поточне підключення в inSSIDer додатку.

1.5. Переваги та недоліки існуючих рішень

1. **WiFi Analyzer:**

Переваги**:**

- Безкоштовний та відкритий код
- Простий у використанні
- Доступний для Windows, macOS та Linux
- Сканує та відображає всі доступні WiFi-мережі
- Переглядає інформацію про мережу
- Виявляє та аналізує мережеві проблеми
- Створення звітів про мережу
- Моніторинг продуктивності WiFi-мережі

Недоліки**:**

- Не показує приблизну відстань до мережі, що важливо для оцінки якості сигналу та оптимального розташування маршрутизаторів.

- Не підтримує фільтрування по частоті 6 ГГц, що обмежує використання з новими пристроями та мережами, які працюють у цьому діапазоні.
- Не вимірює поточну швидкість завантаження, що обмежує оцінку пропускної здатності та швидкості інтернет-з'єднання в реальному часі.

2. **inSSIDer:**

Переваги**:**

- Безкоштовний
- Пропонує більш просунуті функції, ніж WiFi Analyzer
- Детальне відображення спектра WiFi

Недоліки**:**

- Недоступний для Linux
- Всі інші недоліки ті ж самі, як і у WiFi Analyzer

2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ

У розділі визначаються функціональні вимоги до програми-аналізатора WiFi. Обговорюється вибір технологій для розробки: мови програмування, бібліотек для роботи з WiFi. Описується загальна архітектура додатку, проектується інтерфейс користувача. Наводяться необхідні діаграми та специфікації, що допомагають краще зрозуміти структуру та функціонування додатку, а також реалізацію функціоналу для збору даних про WiFi-мережі, розробку інтерфейсу користувача та тестування додатку.

2. 1. Вимоги до додатку

Функціональні вимоги:

- Сканування мереж: Додаток повинен вміти сканувати навколишні WiFi-мережі та надавати інформацію про них, включаючи SSID, силу сигналу, тип шифрування, канал та діапазон частот.
- Перегляд інформації про мережу: Додаток повинен дозволяти переглядати інформацію про мережу, такі як SSID, BSSID, канал, частота, тип шифрування, рівень сигналу, IP-адреса, MAC-адреса.
- Візуалізація даних: Зручне відображення результатів аналізу у вигляді графіків та таблиць, що дозволяє користувачам легко інтерпретувати отримані дані.
- Моніторинг продуктивності WiFi-мережі: Додаток повинен мати можливість моніторити (відстежувати) продуктивність WiFi-мережі, такі як швидкість завантаження.

Нефункціональні вимоги:

- Простота використання: Додаток повинен бути простим у використанні та зрозумілим для користувачів з різним рівнем технічних знань.
- Ефективність: Додаток повинен працювати ефективно та не використовувати занадто багато ресурсів акумулятора або процесора.

- Безпека: Додаток повинен бути безпечним та захищати дані користувачів.

2. 2. Архітектура додатку

Архітектура додатку для аналізу WiFi мереж буде базуватись на патерні MVVM (Model-View-ViewModel). Цей патерн забезпечує розділення логіки додатку на три основні компоненти, що сприяє підвищенню зручності розробки, тестування та підтримки коду. Принцип даного патерну можна побачити на рис. 2.1.

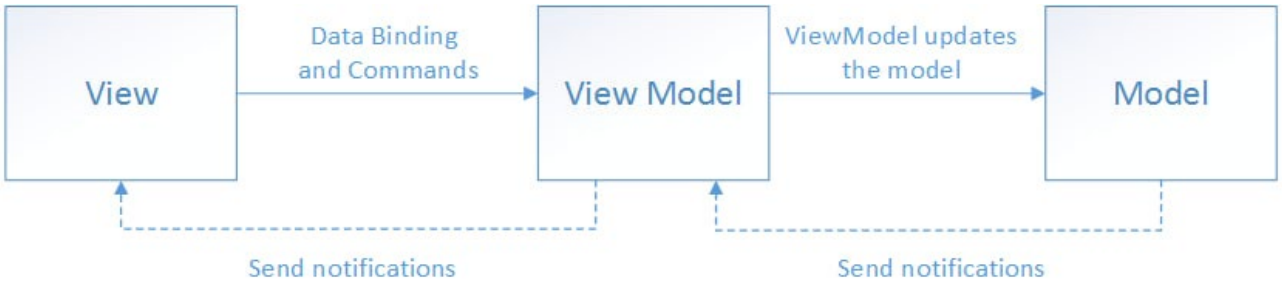

Рисунок 2.1 – Діаграма патерну MVVM [12].

Патерн Model-View-ViewModel (MVVM) базується на розподілі функціональної частини додатка на три ключові компоненти:

- View - представлення або користувацький інтерфейс
- Model - модель або дані, які використовуються в додатку
- ViewModel - проміжний шар між представленням і даними, який забезпечує їх взаємодію

Перевага використання цього патерну полягає у меншій зв'язаності між компонентами та розподілі відповідальності між ними. Тобто Model відповідає за дані, View відповідає за графічний інтерфейс, а ViewModel - за логіку додатка.

Додаток буде використовувати сервіси для взаємодії з нативними API та іншими джерелами даних, які передаються у ViewModel через механізм Dependency Injection (DI).

Основні компоненти архітектури:

- Models (моделі)
    - Опис: Представляють дані та бізнес-логіку додатку, інкапсулюють (приховують) дані та можуть містити методи для виконання бізнес-логіки.
    - Завдання: Зберігати та обробляти дані, виконувати бізнес-логіку.
    - Приклад: Модель WiFi мережі з властивостями, такими як SSID, сила сигналу, частота.
- Views (види)
    - Опис: Відповідають за відображення користувацького інтерфейсу, містять XAML розмітку для представлення даних.
    - Завдання: Забезпечувати інтерфейс для взаємодії з користувачем, використовувати прив'язки даних для відображення з ViewModel, реагувати на події користувача.
    - Приклад: Сторінка зі списком доступних WiFi мереж та їх деталями.
- ViewModels
    - Опис: Містять логіку презентації, використовуються як посередники між Views і Models, містять властивості та команди для взаємодії з Views.
    - Завдання: Отримання та обробка даних з Models, обробка подій користувача, виклик методів сервісів для отримання/збереження даних.
    - Приклад: ViewModel для головної сторінки зі списком доступних WiFi мереж.
- Services (сервіси)
    - Опис: Надають доступ до даних та функціоналу, таких як сканування WiFi мереж, інкапсулюють(приховують) бізнес-логіку і взаємодію з API.
    - Завдання: Виконання запитів до зовнішніх джерел для отримання даних, обробка даних перед їх передачею до ViewModel.
    - Приклад: Сервіс, що сканує доступні WiFi мережі та повертає список знайдених мереж.

Інтеграція компонентів:

- Dependency Injection (DI)

- Опис: Спрощує керування залежностями між компонентами за допомогою механізму Dependency Injection (DI), що дозволяє легко ін'єктувати сервіси у ViewModels.
- Налаштування: Сервіси і ViewModels реєструються в конфігураційному файлі додатку для ін'єкції у реальному часі під час виконання. Реєстрація здійснюється через методи, що додають сервіси до контейнера залежностей.

- Прив’язка View до ViewModel
  - Опис: Забезпечує зв'язок між Views і ViewModels через механізм прив'язки даних, що автоматично оновлює інтерфейс користувача при зміні даних у ViewModel.
  - Налаштування: У XAML розмітці View встановлюється контекст прив'язки до відповідного ViewModel. Властивості і команди ViewModel використовуються для прив'язки елементів інтерфейсу, таких як списки або текстові поля.

Загальний потік роботи додатку:

1. Старт додатку: При запуску додатку ініціалізується головна сторінка, яка відображає інтерфейс користувача.
2. View: Користувач взаємодіє з інтерфейсом, який визначений у View. Елементи інтерфейсу, такі як списки або кнопки, прив'язані до властивостей і команд ViewModel.
3. ViewModel: ViewModel завантажується і викликає необхідні методи для отримання даних. Наприклад, при завантаженні ViewModel може викликати метод сервісу для сканування доступних WiFi мереж.
4. Service: Сервіс виконує запити до нативних API або інших джерел даних для отримання інформації про WiFi мережі. Отримані дані передаються назад до ViewModel.
5. Binding (прив’язка): Дані, отримані від сервісу, передаються до View через механізм прив'язки даних. Інтерфейс користувача автоматично оновлюється для відображення актуальної інформації.

Діаграма даної архітектури зображена на рисунку 2.2.

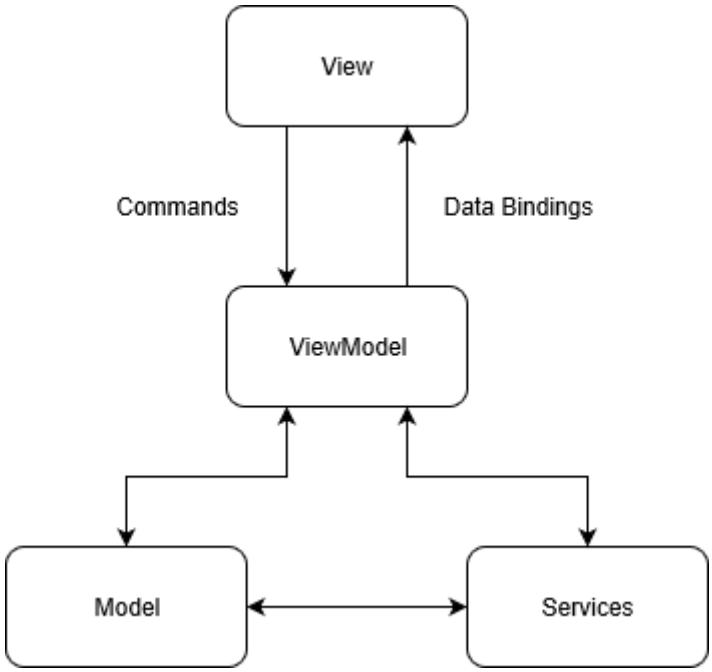

Рисунок 2.2 – Діаграма архітектури даного додатку [13].

2.3. Вибір технологій та інструментів

Для створення додатку аналізу WiFi-мереж виберемо платформу .NET MAUI (Multi-platform App UI). Цей вибір зумовлений кількома ключовими факторами, що роблять .NET MAUI ідеальним рішенням для розробки такого типу додатків:

- Багатоплатформність: Підтримка Windows, macOS, iOS та Android з однієї кодової бази.
- Єдина кодова база: Можливість писати код один раз для усіх платформ, що зменшує час розробки і спрощує підтримку.
- Розширені UI/UX: Багатий набір інтерфейсів та інструментів для створення привабливих додатків.
- Інтеграція з API: Легка інтеграція з платформо-специфічними API, наприклад, для сканування WiFi мереж.
- Активна підтримка: Підтримка від Microsoft та широка спільнота розробників, що забезпечує доступ до ресурсів і підтримку.

Інші використані технології:

- .NET 6/7: Останні версії .NET з усіма необхідними можливостями для розробки високопродуктивних додатків.
- XAML: Мова розмітки для створення складних інтерфейсів користувача.
- C#: Основна мова програмування для високої продуктивності та безпеки.
- Native WiFi API: Для взаємодії з WiFi мережами, забезпечення точності і актуальності даних.
- Speed Test: Використовується для вимірювання швидкості Інтернет-з'єднання.
- Entity Framework Core: Інструмент для роботи з базами даних.
- Managed WiFi: Для керування WiFi-з'єднаннями і отримання інформації про них.
- Microcharts: Бібліотека для візуалізації даних у вигляді графіків.

Вибір інструментів:

- Visual Studio: IDE від Microsoft для розробки .NET MAUI.
- NuGet: Менеджер пакетів для .NET, спрощує встановлення бібліотек і інструментів.

2.4. Проектування інтерфейсу користувача

Програма складається з трьох основних сторінок, кожна з яких надає користувачу унікальну інформацію про його WiFi-мережу та оточуючі мережі. Перша сторінка - "Домашня" (Home) - зосереджена на поточному підключенні та характеристиках мережі, а також на знаходженні швидкості завантаження. Друга сторінка - "Підключено" (Connected) - надає більш детальну інформацію про підключену мережу, включаючи IP-адресу та інші деталі. Та остання сторінка "Мережі" (Networks) відображає список доступних мереж та їх аналіз для вибору найкращого з'єднання у двох варіаціях: таблиця та графіки.

1. Перша сторінка (головна) – Домашня (Home)

- Загальна інформація мережі (Network Details)

Орієнтований інтерфейс даної сторінки зображений на рис. 2.3.

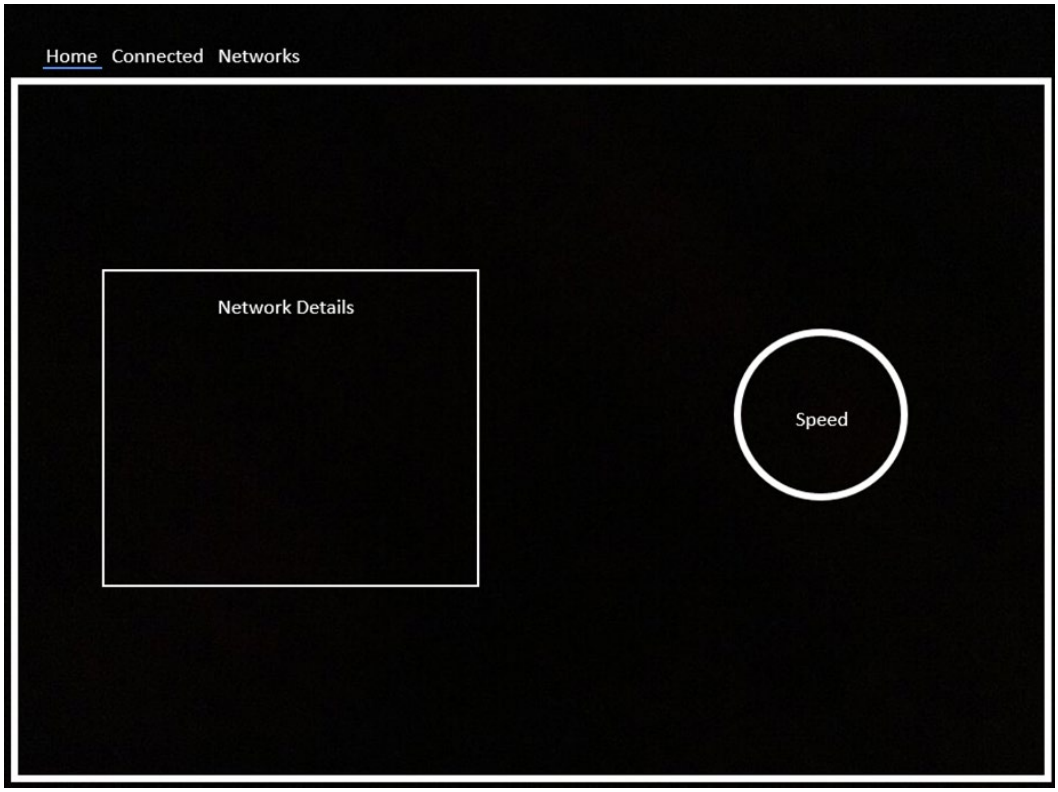

Рисунок 2.3 – Орієнтований інтерфейс першої (головної) сторінки.

2. Друга сторінка - Підключено (Connected)

- Загальна інформація мережі (Network Details)
- IP-адреси мережі (IP Address Details)
- Безпека мережі (Security)
- Інфраструктура мережі (Infrastructure)

Орієнтований інтерфейс даної сторінки зображений на рис. 2.4.

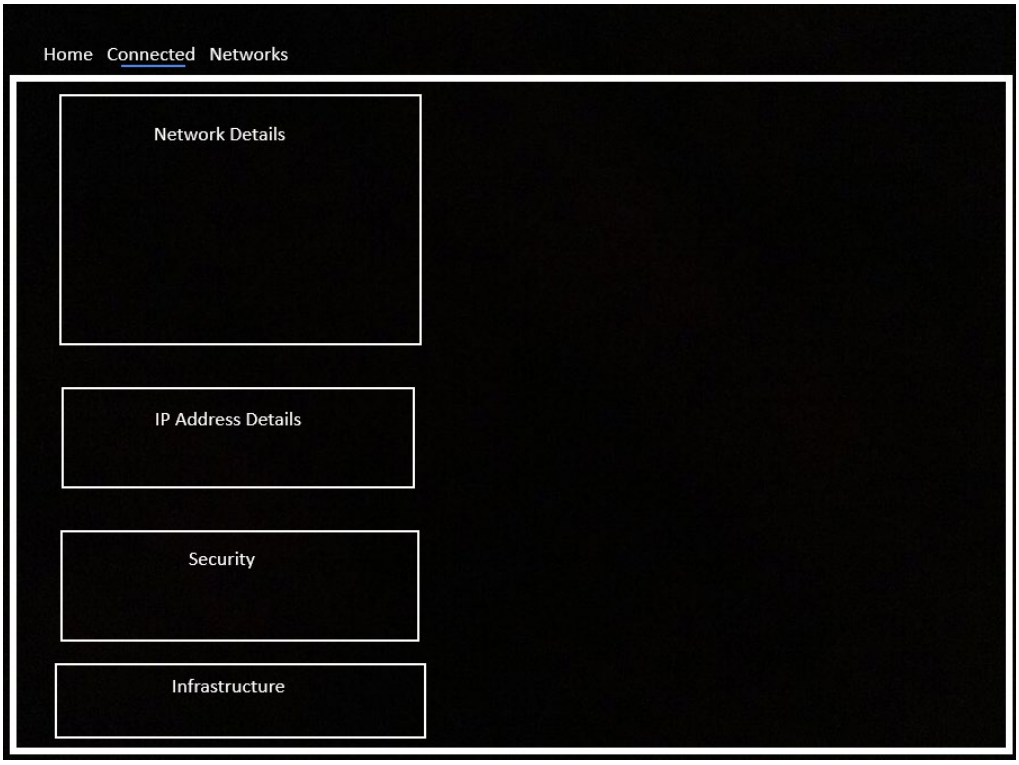

Рисунок 2.4 – Орієнтований інтерфейс другої сторінки.

3. Третя сторінка(остання) – Мережі (Networks)

а) Таблиця (Table)

- Відображення всіх доступних мереж
- Сортування: Можливість сортувати дані за будь-яким стовпчиком у таблиці (наприклад, SSID, Рівень сигналу, Відстань)
- Фільтрація: Можливість фільтрувати дані за частотою мережі WiFi (2.4 ГГц, 5 ГГц або 6 ГГц)

Орієнтований інтерфейс даної сторінки зображений на рис. 2.5.

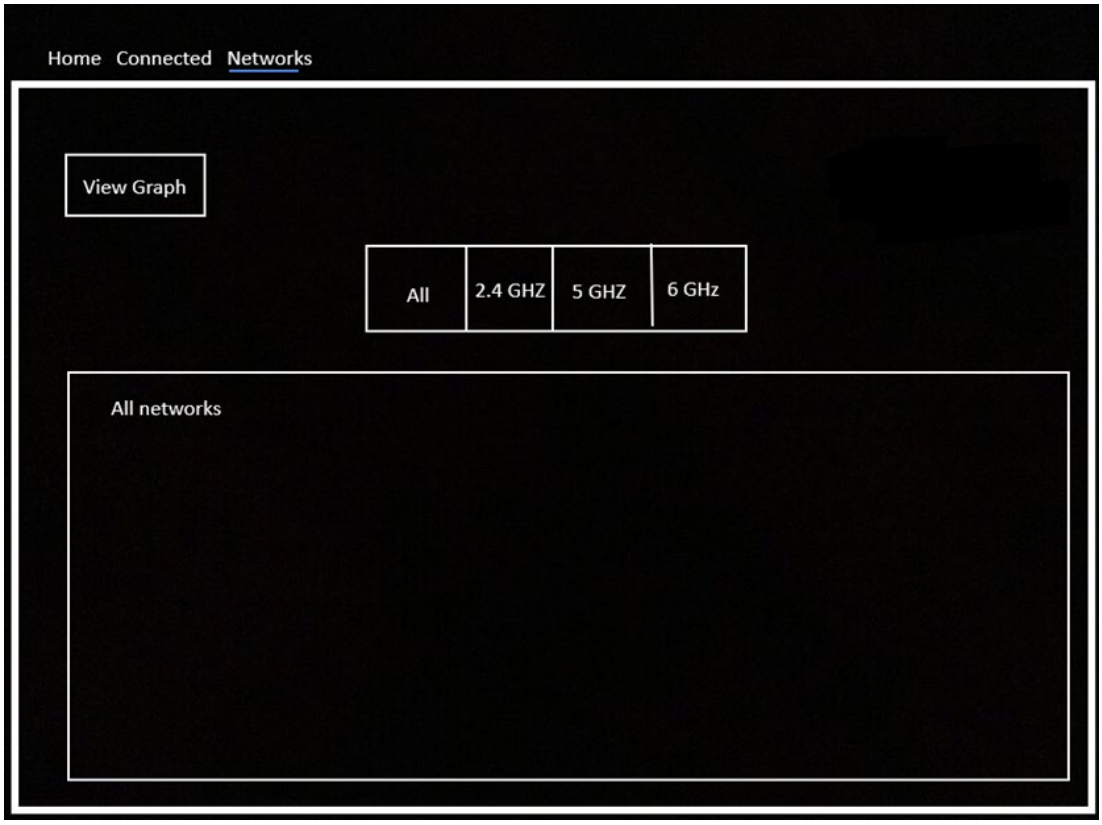

Рисунок 2.5 – Орієнтований інтерфейс третьої сторінки (таблиця).

б) Графіки (Graphs)

- Графік сили сигналу: Відображає силу сигналу кожної WiFi-мережі у таблиці відфільтрованої по номеру каналу:
  - Вісь X: SSID мережі WiFi та з номером каналу
  - Вісь Y: Сила сигналу (у децибелах)
- Графік відстані: Відображає відстань до кожної WiFi-мережі у таблиці:
  - Вісь X: SSID мережі WiFi
  - Вісь Y: Відстань (у метрах)

Орієнтований інтерфейс даної сторінки зображений на рис. 2.6.

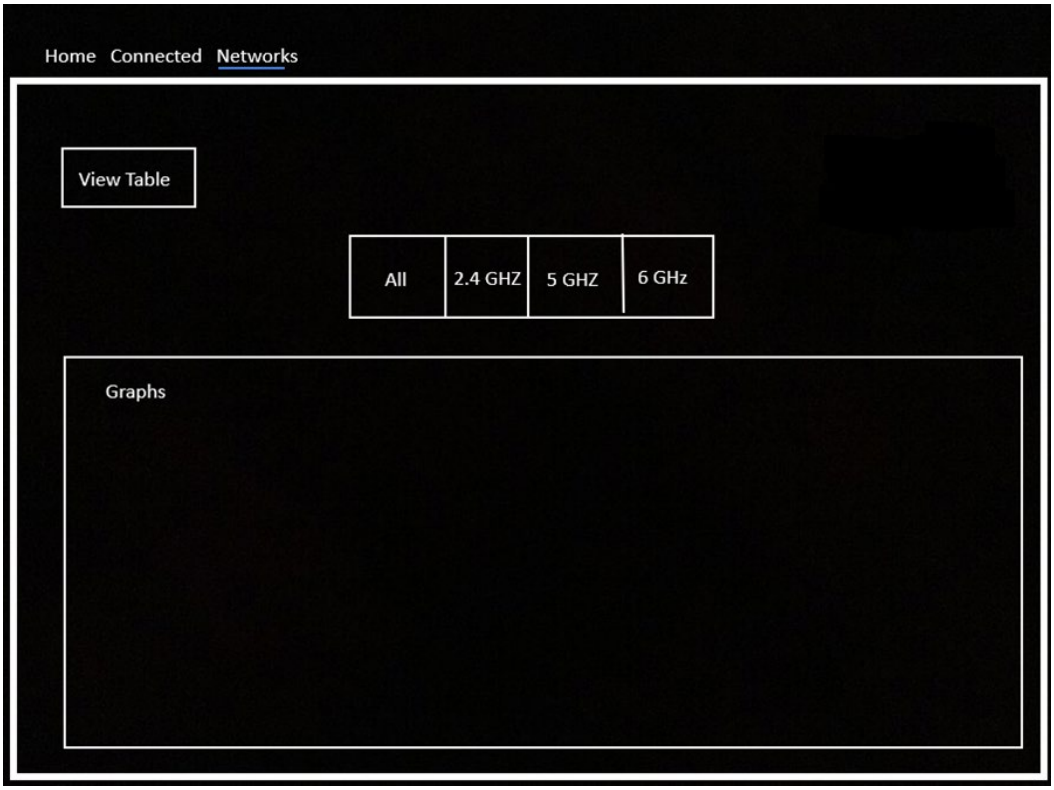

Рисунок 2.6 – Орієнтований інтерфейс третьої сторінки (графіки).

2.5. Реалізація функціоналу для збору даних про WiFi-мережі

Спочатку створимо пустий .NET MAUI проект в IDE Visual Studio 2022 (рис. 2.8).

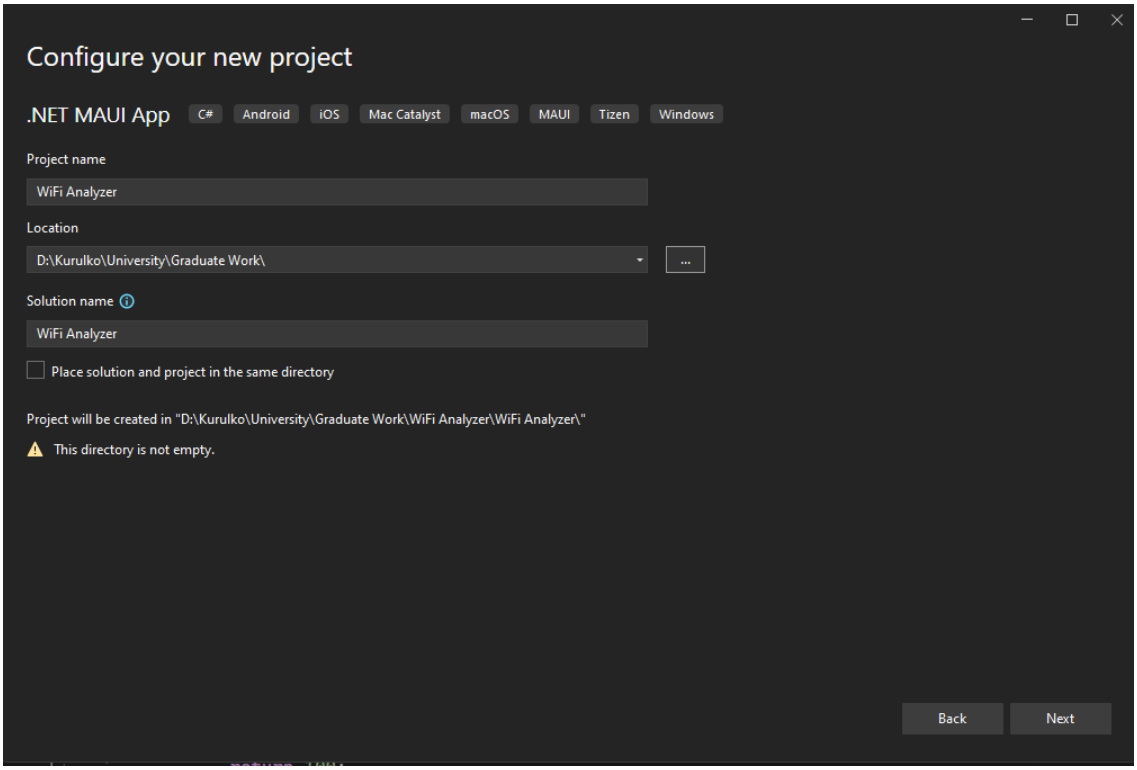

Рисунок 2.7 – Створення нового проекту в Visual Studio 2022

Далі встановимо всі потрібні Nuget пакети (рис. 2.8).

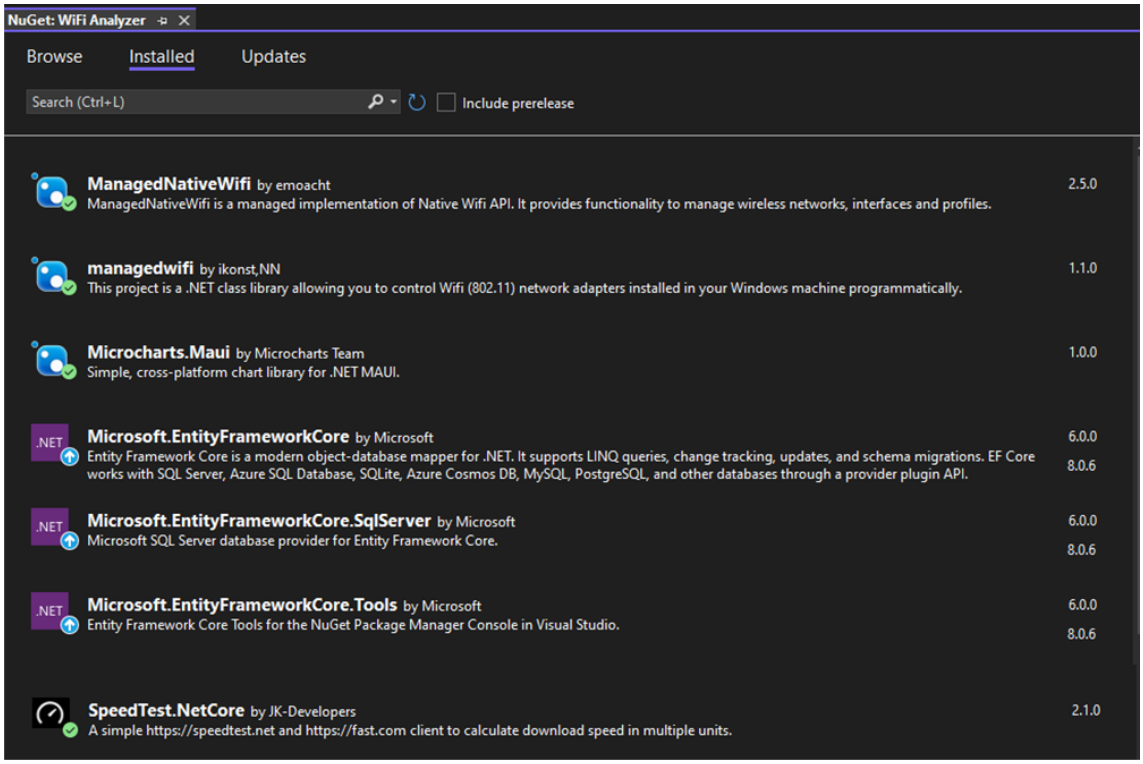

Рисунок 2.8 – Встановлення всіх потрібних Nuget пакетів

Та одразу підключимо базу даних, для цього створимо контекст в папці Database:

```
public class WiFiAnalyzerContext : DbContext
{
    public DbSet<WiFiNetwork> WiFiNetworks { get; set; } = null!;
    public WiFiAnalyzerContext(DbContextOptions<WiFiAnalyzerContext> options) :
base(options)
        => Database.EnsureCreated();
}
```

Добавимо файл зі рядком підключення до локальної бази даних в корні проекту:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb; Database=WiFi_Analyzer_3;
MultipleActiveResultSets=True;"
  }
}
```

Та зареєструємо це все в класі MauiProgram (в головному файлі проекту):

```
IConfiguration configuration = new ConfigurationBuilder()
```

```
            .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
            .Build();
        string connection = configuration.GetConnectionString("DefaultConnection")!;
        services.AddDbContext<WiFiAnalyzerContext>(opts =>
        {
            opts.UseSqlServer(connection);
            opts.EnableSensitiveDataLogging();
        });
```

Далі реалізуємо всі моделі, які потрібні для відображення мереж. Для цього створимо окрему папку для моделей під назвою “Models”, яка буде заходиться в корні проекту.

Спочатку створимо головний інтерфейс IEntityBase в папці “Models/Database”, який буде визначати базові властивості, які повинні мати всі інші класи в базі даних:

```
public interface IEntityBase
{
    [Key]
    long Id { get; set; }
}
```

Його властивості:

- Id: Унікальний ідентифікатор.

Далі створимо WiFiNetwork клас в тій самій папці:

```
[Index(nameof(MacAddress), IsUnique = true)]
public class WiFiNetwork : IEntityBase
{
    public long Id { get; set; }
    [Required]
    public string SSID { get; set; } = null!;
    [Required]
    public string Protocol { get; set; } = null!;
    public DateTime LastSeen { get; set; }
    [Required]
    public byte[] MacAddress { get; set; } = null!;
    [NotMapped]
    public string StringMacAddress => MacAddress.MacAddressToString();
    [Column("Frequency")]
    public long FrequencyInHz { get; set; }
    public int FrequencyInkHz => (int)(FrequencyInHz / Math.Pow(10, 3));
    public int FrequencyInMHz => (int)(FrequencyInHz / Math.Pow(10, 6));
```

```
    public int FrequencyInGHz => (int)(FrequencyInHz / Math.Pow(10, 9));
    public int Channel { get; set; }
    [Column("Secured")]
    public bool IsSecured { get; set; }
    [Column("Authentication")]
    public Dot11AuthAlgorithm AuthenticationAlgorithm { get; set; }
    [NotMapped]
    public NetworkStates? NetworkStates { get; set; }
    [NotMapped]
    public IPAddressInfo? IPAddressInfo { get; set; }
    [NotMapped]
    public NetworkSecurityInfo? NetworkSecurityInfo { get; set; }
    [NotMapped]
    public NetworkInfrastructureInfo? NetworkInfrastructureInfo { get; set; }
}
```

Він має такі властивості:

- Id: Унікальний ідентифікатор.
- SSID: Назва мережі (обов'язково).
- Protocol: Протокол мережі (обов'язково).
- LastSeen: Час останнього виявлення.
- MacAddress: MAC-адреса (обов'язково).
- StringMacAddress: MAC-адреса у вигляді рядка (не зберігається).
- FrequencyInHz: Частота в герцах.
- FrequencyInkHz/MHz/GHz: Частота в кГц, МГц, ГГц (обчислюються, не зберігаються).
- Channel: Номер каналу.
- IsSecured: Чи захищена мережа.
- AuthenticationAlgorithm: Алгоритм аутентифікації.
- NetworkStates/IPAddressInfo/NetworkSecurityInfo/NetworkInfrastructureInfo : Додаткова інформація про мережу (не зберігаються)

Інші допоміжні моделі (класи) для роботи з мережею:

- Клас DownloadSpeed

Клас ‘DownloadSpeed’ містить властивості для швидкості завантаження та одиниць виміру, а також використовує оператор явного перетворення для

перетворення об'єктів типу ‘SpeedTest.Net.Models.DownloadSpeed’ на екземпляри ’DownloadSpeed’(екземпляр самого класу), встановлюючи значення швидкості та одиниць виміру:

```
public class DownloadSpeed
{
    public double Speed { get; set; }
    public SpeedTestUnit Unit { get; set; }
    public static explicit operator DownloadSpeed(SpeedTest.Net.Models.DownloadSpeed
_downloadSpeed)
    {
        DownloadSpeed downloadSpeed = new();
        downloadSpeed.Speed = _downloadSpeed.Speed;
        downloadSpeed.Unit = _downloadSpeed.Unit.ParseToSpeedTestUnit();
        return downloadSpeed;
    }
}
```

- Клас IPAddressInfo

Клас `IPAddressInfo` містить властивості для приватної IPv4 адреси, маски підмережі та, можливо, публічної IPv4 адреси:

```
public class IPAddressInfo
{
    public string PrivateIPv4 { get; set; } = null!;
    public string SubnetMask { get; set; } = null!;
    public string? PublicIPv4 { get; set; }
}
```

- Клас NetworkInfrastructureInfo

Клас `NetworkInfrastructureInfo` містить властивості для інтерфейсу мережі, його типу та статусу операційної діяльності:

```
public class NetworkInfrastructureInfo
{
    public string Interface { get; set; } = null!;
    public NetworkInterfaceType InterfaceType { get; set; }
    public OperationalStatus OperationalStatus { get; set; }
}
```

- Клас NetworkSecurityInfo

Клас `NetworkSecurityInfo` містить властивості для алгоритму аутентифікації та шифрування мережі:

```
public class NetworkSecurityInfo
{
    public Dot11AuthAlgorithm Authentication { get; set; }
    public Dot11CipherAlgorithm Encryption { get; set; }
}
```

- Клас NetworkStates

Клас `NetworkStates` містить властивості, що вказують на підключення до мережі, силу сигналу у міліватах та у відсотках, а також відстань до точки доступу у метрах:

```
public class NetworkStates
{
    public bool IsConnected { get; set; }
    public int SignalStrengthIndBm { get; set; }
    public int SignalStrengthInPercentage
    {
        get
        {
            const int minRssi = -100; // Minimum RSSI value
            if (SignalStrengthIndBm <= minRssi)
                return 0;
            const int maxRssi = -50;  // Maximum RSSI value
            if (SignalStrengthIndBm >= maxRssi)
                return 100;
            return 2 * (SignalStrengthIndBm + 100);
        }
    }
    public double DistanceInMeters { get; set; }
}
```

Умовно поділимо даний додаток на дві частини:

1. Перша пов’язана з поточною мережею
2. Інша - зі всіма мережами

Спочатку створимо допоміжний абстрактний клас NetworkService в папці “Services/Network”, який буде базовим для цих двох частин.

Деякі основні моменти реалізації цього допоміжного класу:

- Визначення протоколу за частотою та силою сигналу:

```
protected string FindProtocolString(WlanBssEntry bssEntry)
{
    int rssi = bssEntry.rssi;
    uint chCenterFrequency = bssEntry.chCenterFrequency;

    if (chCenterFrequency >= 2400000 && chCenterFrequency < 2500000)
    {
        // 2.4 GHz band
        if (rssi <= -90)
            return "802.11b";
        else if (rssi <= -75)
            return "802.11g";
        else if (rssi <= -50)
            return "802.11n";
        else
            return "802.11ax"; // Likely 802.11ax due to strong signal in 2.4 GHz
    }
    else if (chCenterFrequency >= 5000000 && chCenterFrequency < 6000000)
    {
        // 5 GHz band
        if (rssi <= -70)
            return "802.11a";
        else if (rssi <= -50)
            return "802.11n";
        else
            return "802.11ac/ax"; // Likely newer standard (ac or ax) due to
strong signal
    }
    else if (chCenterFrequency >= 57000000 && chCenterFrequency <= 66000000)
    {
        // Tentative check for 60 GHz band (needs verification)
        return "802.11ad";
    }
    else
    {
        // Frequency band not identified
        return "Unknown";
    }
}
```

- Знаходження каналу за допомогою частоти:

```
protected int GetChannelFromFrequency(long frequency)
```

```
    {
        int frequencyInMHz = FrequencyInMHz(frequency);

        if (frequencyInMHz >= 2412 && frequencyInMHz <= 2472)
        {
            return (frequencyInMHz - 2407) / 5;
        }
        else if (frequencyInMHz == 2484)
        {
            return 14;
        }
        else if (frequencyInMHz >= 5180 && frequencyInMHz <= 5825)
        {
            return (frequencyInMHz - 5000) / 5;
        }
        return -1; // Unknown channel
    }
```

- Розрахунок приблизної відстані до мережі за частотою сигналу та його силою:

```
    protected double CalculateDistance(int signalStrength, long frequency)
    {
        int frequencyInMHz = FrequencyInMHz(frequency);
        double exp = (27.55 - (20 * Math.Log10(frequencyInMHz)) +
Math.Abs(signalStrength)) / 20.0;
        return Math.Pow(10.0, exp);
    }
```

Тепер почнемо з реалізації першої частини:

Створимо сервіси для отримання даних про поточну мережу у папці "Services/Network/ConnectedNetwork".

Сервіс IConnectedNetworkService:

```
public interface IConnectedNetworkService
{
    WiFiNetwork GetConnectedWiFiNetwork();
    NetworkStates GetConnectedNetworkStates();
    Task<IPAddressInfo> GetConnectedIPAddressInfo();
    NetworkSecurityInfo GetConnectedNetworkSecurityInfo();
    NetworkInfrastructureInfo GetConnectedNetworkInfrastructureInfo();
}
```

Основні елементи реалізації цього :

- GetConnectedWiFiNetwork метод, який відповідає за знаходження даної підключеної мережі у вигляді WiFiNetwork класу:

```
public WiFiNetwork GetConnectedWiFiNetwork()
{
    WlanBssEntry connectedBssEntry = GetConnectedWlanBssEntry();
    WiFiNetwork wiFiNetwork = new();
    long frequency = GetFrequencyFromChannel(connectedBssEntry.chCenterFrequency);
    string currentSSID = GetStringForSSID(connectedBssEntry.dot11Ssid);
    wiFiNetwork.SSID = currentSSID;
    wiFiNetwork.Channel = GetChannelFromFrequency(frequency);
    wiFiNetwork.FrequencyInHz = frequency;
    wiFiNetwork.Protocol = FindProtocolString(connectedBssEntry);
    wiFiNetwork.MacAddress = connectedBssEntry.dot11Bssid;
    WlanAvailableNetwork wlanAvailableNetwork =
GetWlanAvailableNetworkByProfileName(currentSSID)!.Value;
    wiFiNetwork.IsSecured = wlanAvailableNetwork.securityEnabled;
    wiFiNetwork.AuthenticationAlgorithm =
wlanAvailableNetwork.dot11DefaultAuthAlgorithm;
    return wiFiNetwork;
}
```

- GetConnectedNetworkStates метод, який відповідає за знаходження станів даної підключеної мережі у вигляді NetworkStates класу:

```
public NetworkStates GetConnectedNetworkStates()
{
    WlanBssEntry connectedBssEntry = GetConnectedWlanBssEntry();
    long frequency = GetFrequencyFromChannel(connectedBssEntry.chCenterFrequency);
    int signalStrength = connectedBssEntry.rssi;
    NetworkStates networkStates = new();
    networkStates.DistanceInMeters = CalculateDistance(signalStrength, frequency);
    networkStates.IsConnected = true;
    networkStates.SignalStrengthIndBm = signalStrength;
    return networkStates;
}
```

- GetConnectedIPAddressInfo метод, який відповідає за знаходження властивостей адрес мережі у вигляді IPAddressInfo класу:

```
public async Task<IPAddressInfo> GetConnectedIPAddressInfo()
{
    IPAddressInfo iPAddressInfo = new ();
    iPAddressInfo.PrivateIPv4 = GetPrivateIPv4();
    iPAddressInfo.PublicIPv4 = await GetPublicIPv4();
    iPAddressInfo.SubnetMask = GetSubnetMask();
    return iPAddressInfo;
}
```

- GetConnectedNetworkSecurityInfo метод, який відповідає за знаходження властивостей інтерфейсу мережі у вигляді NetworkSecurityInfo класу:

```
public NetworkSecurityInfo GetConnectedNetworkSecurityInfo()
{
    NetworkSecurityInfo networkSecurityInfo = new();
    var client = new WlanClient();
    foreach (var wlanInterface in client.Interfaces)
    {
        var currentConnection = wlanInterface.CurrentConnection;
        WlanAvailableNetwork[] networks =
wlanInterface.GetAvailableNetworkList(0);
        foreach (WlanAvailableNetwork network in networks)
        {
            if (network.profileName == currentConnection.profileName)
            {
                networkSecurityInfo.Authentication =
network.dot11DefaultAuthAlgorithm;
                networkSecurityInfo.Encryption =
network.dot11DefaultCipherAlgorithm;
                break;
            }
        }
    }
    return networkSecurityInfo;
}
```

- GetConnectedNetworkInfrastructureInfo метод, який відповідає за знаходження алгоритму аутентифікації та шифрування мереж у вигляді NetworkInfrastructureInfo класу:

```
public NetworkInfrastructureInfo GetConnectedNetworkInfrastructureInfo()
{
    NetworkInfrastructureInfo networkInfrastructureInfo = new();
```

```
        var networkInterfaces = NetworkInterface.GetAllNetworkInterfaces();
        var currentInterface = networkInterfaces.FirstOrDefault(nic =>
nic.OperationalStatus == OperationalStatus.Up);
        if (currentInterface != null)
        {
            networkInfrastructureInfo.InterfaceType =
currentInterface.NetworkInterfaceType;
            networkInfrastructureInfo.OperationalStatus =
currentInterface.OperationalStatus;
            networkInfrastructureInfo.Interface = currentInterface.Description;
        }
        return networkInfrastructureInfo;
    }
```

Також реалізуємо сервіси для вимірювання швидкості завантаження поточної мережі у папці "Services/SpeedTest":

Сервіс ISpeedTestService:

```
public interface ISpeedTestService
{
    Task<DownloadSpeed> GetDownloadSpeedAsync();
}
```

Його реалізація:

```
public class SpeedTestService : ISpeedTestService
{
    public async Task<DownloadSpeed> GetDownloadSpeedAsync()
        => (DownloadSpeed)await
FastClient.GetDownloadSpeed(SpeedTestUnit.MegaBitsPerSecond);
}
```

Тепер друга частина - всі доступні мережі:

Створимо сервіси для отримання даних про всі мережі у папці "Services/Networks".

Сервіс INetworkService:

```
public interface INetworksService
{
    Task UpdateWiFiNetworksAsync();
    Task<IEnumerable<WiFiNetwork>> GetWiFiNetworksAsync();
    Task<IEnumerable<WiFiNetwork>> GetWiFiNetworksWithStatesAsync();
    Task AddWiFiNetworkAsync(WiFiNetwork network);
```

```
    Task UpdateWiFiNetworkAsync(WiFiNetwork network);
    Task DeleteWiFiNetworkByIdAsync(long networkId);
    Task DeleteWiFiNetworkByBSSIDAsync(string BSSID);
    Task<WiFiNetwork?> GetWiFiNetworkByIdAsync(long id);
    Task<WiFiNetwork?> GetWiFiNetworkByBSSIDAsync(string BSSID);
    NetworkStates? GetNetworkStates(WiFiNetwork network);
}
```

Основні елементи реалізації цього :

- UpdateWiFiNetworksAsync метод, який відповідає за оновлення даних про Wi-Fi мережі:

```
    public async Task UpdateWiFiNetworksAsync() {
        WlanClient client = new WlanClient();
        foreach (WlanClient.WlanInterface wlanInterface in client.Interfaces) {
            WlanBssEntry[] bssEntries = wlanInterface.GetNetworkBssList();

            foreach (var bssEntry in bssEntries) {
                string ssid = GetStringForSSID(bssEntry.dot11Ssid);
                long frequency = GetFrequencyFromChannel(bssEntry.chCenterFrequency);
                byte[] macAddress = bssEntry.dot11Bssid;
                int channel = GetChannelFromFrequency(frequency);
                string protocol = FindProtocolString(bssEntry);
                if (GetWlanAvailableNetworkByProfileName(ssid) is WlanAvailableNetwork
wlanAvailableNetwork) {
                    Dot11AuthAlgorithm authenticationAlgorithm =
wlanAvailableNetwork.dot11DefaultAuthAlgorithm;
                    bool isSecured = wlanAvailableNetwork.securityEnabled;
                    WiFiNetwork network = new() {
                        SSID = ssid,
                        FrequencyInHz = frequency,
                        Channel = channel,
                        MacAddress = macAddress,
                        IsSecured = isSecured,
                        AuthenticationAlgorithm = authenticationAlgorithm,
                        Protocol = protocol,
                        LastSeen = DateTime.Now
                    };
                    string macAddressStr = macAddress.MacAddressToString();
                    WiFiNetwork? _network = await
GetWiFiNetworkByBSSIDAsync(macAddressStr);
                    if (_network is null) {
                        await AddWiFiNetworkAsync(network);
```

```
                }
                else {
                    CopyWiFiNetworkValues(network, _network);
                    await UpdateWiFiNetworkAsync(_network);
                }
            }
        }
    }
}
```

- GetNetworkStates метод, який відповідає за отримання даних про стан для конкретної мережі у вигляді NetworkStates класу:

```
public NetworkStates? GetNetworkStates(WiFiNetwork network)
{
    WlanClient client = new WlanClient();
    foreach (WlanClient.WlanInterface wlanInterface in client.Interfaces)
    {
        WlanBssEntry[] bssEntries = wlanInterface.GetNetworkBssList();
        foreach (var bssEntry in bssEntries)
        {
            string macAddressStr = bssEntry.dot11Bssid.MacAddressToString();
            if (network.StringMacAddress == macAddressStr)
            {
                string ssid = GetStringForSSID(bssEntry.dot11Ssid);
                int signalStrength = bssEntry.rssi;
                long frequency =
GetFrequencyFromChannel(bssEntry.chCenterFrequency);
                double distance = CalculateDistance(signalStrength, frequency);
                bool isConnected = false;
                if (wlanInterface.InterfaceState == WlanInterfaceState.Connected)
                {
                    string connectedSSID =
GetStringForSSID(wlanInterface.CurrentConnection.wlanAssociationAttributes.dot11Ssid);
                    if (ssid == connectedSSID)
                        isConnected = true;
                }
                return new NetworkStates()
                {
                    IsConnected = isConnected,
                    DistanceInMeters = distance,
                    SignalStrengthIndBm = signalStrength
                };
            }
        }
```

```
        }
        return null;
    }
```

Щоб всі ці сервіси працювали, потрібно їх зареєструвати в файлі MauiProgram (в головному файлі проекту):

```
services.AddSingleton<IConnectedNetworkService, ConnectedNetworkService>();
services.AddSingleton<ISpeedTestService, SpeedTestService>();
services.AddSingleton<INetworksService, NetworksService>();
```

2.6. Реалізація інтерфейсу користувача

Створимо всього три сторінки:

1. Основні дані поточної мережі та швидкість завантаження

Перейдемо до реалізації ViewModels компонентів. Створимо папку ViewModels, а в ній оснований ViewModel компонент:

```
public abstract class ViewModelBase : INotifyPropertyChanged, IDisposable
{
    Timer? networkStateTimer;
    public ViewModelBase()
        => networkStateTimer = new Timer(UpdateStates, null, TimeSpan.Zero,
TimeSpan.FromSeconds(30));
    protected abstract void UpdateStates(object? _ = null);
    public event PropertyChangedEventHandler? PropertyChanged;
    public ICommand LoadDataCommand => new Command(async () =>
    {
        try
        {
            await LoadDataAsync();
        }
        catch (Exception ex)
        {
            await Shell.Current.DisplayAlert("Error", ex.Message, "OK");
        }
    });
    public async Task LoadDataAsync()
    {
        try
```

```
        {
            await GetDataAsync();
        }
        catch (Exception ex)
        {
            await ErrorHandler.DisplayErrorAsync(ex.Message);
        }
    }
    protected abstract Task GetDataAsync();
    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName =
"")
        => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    public void Dispose()
        => networkStateTimer?.Dispose();
}
```

Далі в цій же папці, створимо папку “Network” в яку помістимо основну ViewModel для мережі:

```
public abstract class NetworkViewModel : ViewModelBase
{
    protected readonly IConnectedNetworkService connectedNetworkService;
    protected readonly INetworksService networksService;
    WiFiNetwork? connectedNetwork;
    public WiFiNetwork? ConnectedNetwork
    {
        get => connectedNetwork;
        set
        {
            connectedNetwork = value;
            OnPropertyChanged(nameof(ConnectedNetwork));
        }
    }
    NetworkStates? networkStates;
    public NetworkStates? NetworkStates
    {
        get => networkStates;
        set
        {
            networkStates = value;
            OnPropertyChanged(nameof(NetworkStates));
        }
    }
```

```
    public NetworkViewModel(IConnectedNetworkService connectedNetworkService,
INetworksService networksService)
        => (this.connectedNetworkService, this.networksService) =
(connectedNetworkService, networksService);
    protected override void UpdateStates(object? state = null)
        => NetworkStates = connectedNetworkService.GetConnectedNetworkStates();
    protected override async Task GetDataAsync()
    {
        ConnectedNetwork = connectedNetworkService.GetConnectedWiFiNetwork();
        await networksService.UpdateWiFiNetworkAsync(ConnectedNetwork);
        UpdateStates();
    }
}
```

Та конкретна реалізація для головної сторінки:

```
public class MainPageViewModel : NetworkViewModel
{
    readonly ISpeedTestService speedTestService;
    DownloadSpeed? downloadSpeed;
    public DownloadSpeed? DownloadSpeed
    {
        get => downloadSpeed;
        set
        {
            downloadSpeed = value;
            OnPropertyChanged(nameof(DownloadSpeed));
        }
    }
    bool isBusy;
    public bool IsBusy
    {
        get => isBusy;
        set
        {
            isBusy = value;
            OnPropertyChanged(nameof(IsBusy));
        }
    }
    public ICommand GetDownloadSpeedCommand => new Command(async () => await
GetDownloadSpeedAsync());
    public MainPageViewModel(IConnectedNetworkService connectedNetworkService,
ISpeedTestService speedTestService, INetworksService networksService) :
base(connectedNetworkService, networksService)
```

```
            => this.speedTestService = speedTestService;
    public async Task GetDownloadSpeedAsync()
    {
        try
        {
            DownloadSpeed = null;
            IsBusy = true;
            DownloadSpeed = await speedTestService.GetDownloadSpeedAsync();
            IsBusy = false;
        }
        catch (Exception ex)
        {
            await ErrorHandler.DisplayErrorAsync(ex.Message);
        }
    }
}
```

Тепер сам інтерфейс для даного ViewModel компонента:

- XAML код Home (головної) сторінки:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:controls="clr-namespace:WiFi_Analyzer.Controls"
             x:Class="WiFi_Analyzer.Pages.MainPage"
             ControlTemplate="{StaticResource CommonFrameTemplate}"
             Title="Home">
        <Grid Margin="20,10,0,0">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <StackLayout Grid.Column="0"
                         HorizontalOptions="CenterAndExpand"
                         VerticalOptions="CenterAndExpand">
                <controls:NetworkDetailsView
                    ConnectedNetwork="{Binding ConnectedNetwork}"
                    NetworkStates="{Binding NetworkStates}"/>
            </StackLayout>
            <StackLayout Grid.Column="1"
                         HorizontalOptions="CenterAndExpand"
                         VerticalOptions="CenterAndExpand" Spacing="20">
                <controls:SpeedTestView
                    DownloadSpeed="{Binding DownloadSpeed}"
```

```
                    IsBusy="{Binding IsBusy}"
                    GetDownloadSpeedCommand="{Binding GetDownloadSpeedCommand}"/>
            </StackLayout>
        </Grid>
</ContentPage>
```

- C# код Home (головної) сторінки.

```
public partial class MainPage : ContentPage
{
    readonly MainPageViewModel mainPageViewModel = null!;

    public MainPage()
    {
        mainPageViewModel = ServiceHelper.GetService<MainPageViewModel>()!;

        InitializeComponent();

        BindingContext = mainPageViewModel;
    }

    protected override async void OnAppearing()
    {
        base.OnAppearing();
        await mainPageViewModel.LoadDataAsync();
    }

    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        mainPageViewModel.Dispose();
    }
}
```

2. Більш детальна інформація про поточну мережу (Connected)

Її реалізація ViewModel:

```
public class ConnectedNetworkViewModel : NetworkViewModel
{
    IPAddressInfo? _IPAddressInfo;
    public IPAddressInfo? IPAddressInfo
    {
        get => _IPAddressInfo;
        set
```

```
        {
            _IPAddressInfo = value;
            OnPropertyChanged(nameof(IPAddressInfo));
        }
    }
    NetworkSecurityInfo? networkSecurityInfo;
    public NetworkSecurityInfo? NetworkSecurityInfo
    {
        get => networkSecurityInfo;
        set
        {
            networkSecurityInfo = value;
            OnPropertyChanged(nameof(NetworkSecurityInfo));
        }
    }
    NetworkInfrastructureInfo? networkInfrastructureInfo;
    public NetworkInfrastructureInfo? NetworkInfrastructureInfo
    {
        get => networkInfrastructureInfo;
        set
        {
            networkInfrastructureInfo = value;
            OnPropertyChanged(nameof(NetworkInfrastructureInfo));
        }
    }
    public ConnectedNetworkViewModel(IConnectedNetworkService connectedNetworkService,
INetworksService networksService) : base(connectedNetworkService, networksService) { }

    protected override async Task GetDataAsync()
    {
        await base.GetDataAsync();
        IPAddressInfo = await connectedNetworkService.GetConnectedIPAddressInfo();
        NetworkSecurityInfo =
connectedNetworkService.GetConnectedNetworkSecurityInfo();
        NetworkInfrastructureInfo =
connectedNetworkService.GetConnectedNetworkInfrastructureInfo();
    }
}
```

Тепер сам інтерфейс для даного ViewModel компонента:

- XAML код Connected сторінки.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="WiFi_Analyzer.Pages.ConnectedNetworkPage"
             xmlns:controls="clr-namespace:WiFi_Analyzer.Controls"
             ControlTemplate="{StaticResource CommonFrameTemplate}"
             Title="Connected">
    <ScrollView>
        <StackLayout Margin="20,10,0,0">
            <!-- Table NetworkDetails -->
            <controls:NetworkDetailsView
                ConnectedNetwork="{Binding ConnectedNetwork}"
                NetworkStates="{Binding NetworkStates}"/>
            <!-- Table IPAddressInfo -->
            <controls:IPAddressInfoView
                IPAddressInfo="{Binding IPAddressInfo}" />
            <!-- Table NetworkSecurityInfo -->
            <controls:NetworkSecurityInfoView
                NetworkSecurityInfo="{Binding NetworkSecurityInfo}" />
            <!-- Table NetworkInfrastructureInfo -->
            <controls:NetworkInfrastructureInfoView
                NetworkInfrastructureInfo="{Binding NetworkInfrastructureInfo}"/>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

- C# код Connected сторінки.

```
public partial class ConnectedNetworkPage : ContentPage
{
    readonly ConnectedNetworkViewModel connectedNetworkViewModel = null!;
    public ConnectedNetworkPage()
    {
        connectedNetworkViewModel =
ServiceHelper.GetService<ConnectedNetworkViewModel>()!;
        InitializeComponent();
        BindingContext = connectedNetworkViewModel;
    }
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        await connectedNetworkViewModel.LoadDataAsync();
    }
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
```

```
        connectedNetworkViewModel.Dispose();
    }
}
```

3. Інформація про всі доступні мережі.

Реалізуємо основний ViewModel компонент для мереж:

```
public abstract class NetworksViewModel : ViewModelBase
{
    protected readonly INetworksService networksService;
    public IEnumerable<WiFiNetwork> Networks = Enumerable.Empty<WiFiNetwork>();
    protected IEnumerable<WiFiNetwork> filteredWiFiNetworks =
Enumerable.Empty<WiFiNetwork>();
    public virtual IEnumerable<WiFiNetwork> FilteredWiFiNetworks
    {
        get => filteredWiFiNetworks;
        set
        {
            filteredWiFiNetworks = value;
            OnPropertyChanged(nameof(FilteredWiFiNetworks));
            OnPropertyChanged(nameof(HasFilteredNetworks));
        }
    }
    public bool HasFilteredNetworks => FilteredWiFiNetworks.Any();
    public ICommand FilterByGHzCommand => new Command<string>(parameter =>
FilterByGHz(parameter));
    public NetworksViewModel(INetworksService networksService)
        => this.networksService = networksService;
    protected override async Task GetDataAsync()
    {
        await networksService.UpdateWiFiNetworksAsync();
        FilteredWiFiNetworks = Networks = await
networksService.GetWiFiNetworksWithStatesAsync();
    }
    protected override async void UpdateStates(object? state = null)
        => FilteredWiFiNetworks = Networks = await
networksService.GetWiFiNetworksWithStatesAsync();
    protected void FilterByGHz(string parameter)
        => FilteredWiFiNetworks = NetworksFilter.FilterByGHz(Networks, parameter);
}
```

Ця сторінка поділяється в дві частини:

a. Таблиця з інформацію про доступні мережі. Його ViewModel компонент:

```
public class NetworksTableViewModel : NetworksViewModel
{
    public ICommand SortCommand => new Command<string>(propertyName =>
SortBy(propertyName));
    public NetworksTableViewModel(INetworksService networksService) :
base(networksService)
    {
    }
    public Dictionary<string, OrderBy> SortDirections { get; set; } = new();
    void SortBy(string propertyName)
    {
        OrderBy currentDirection = SortDirections.ContainsKey(propertyName) ?
SortDirections[propertyName] : OrderBy.Ascending;
        OrderBy newDirection = ChangeOrderBy(currentDirection);
        SortDirections = new Dictionary<string, OrderBy>();
        SortDirections[propertyName] = newDirection;
        FilteredWiFiNetworks =
FilteredWiFiNetworks.AsQueryable().DynamicOrderBy(propertyName,
newDirection).ToList();
        OnPropertyChanged(nameof(SortDirections));
    }
    OrderBy ChangeOrderBy(OrderBy orderBy)
        => orderBy == OrderBy.Ascending ? OrderBy.Descending : OrderBy.Ascending;
}
```

Тепер сам інтерфейс для даного ViewModel компонента:

- C# код Networks сторінки (таблиця).

```
public partial class NetworksTablePage : ContentPage
{
    readonly NetworksTableViewModel networksTableViewModel = null!;
    public NetworksTablePage()
    {
        networksTableViewModel = ServiceHelper.GetService<NetworksTableViewModel>()!;
        InitializeComponent();
        BindingContext = networksTableViewModel;
    }
    protected override void OnAppearing()
    {
        base.OnAppearing();
        UpdateNetworks();
    }
    async void UpdateNetworks()
        => await networksTableViewModel.LoadDataAsync();
```

```
    async void NavigateToGraphPage(object sender, EventArgs e)
        => await Navigation.PushAsync(new NetworksGraphPage());
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        networksTableViewModel.Dispose();
    }
}
```

б. Графіки силу сигналу та відстані від для всіх доступних мереж. Його ViewModel компонент зображений на рис. 2.9.

Рисунок 2.9 – NetworksGraphPageViewModel клас, який буде ViewModel компонентом для Networks сторінки (графіки).

Тепер сам інтерфейс для даного ViewModel компонента:

- XAML код Networks сторінки (графіки).

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:micro="clr-namespace:Microcharts.Maui;assembly=Microcharts.Maui"
             x:Class="WiFi_Analyzer.Pages.Networks.NetworksGraphPage"
             xmlns:controls="clr-namespace:WiFi_Analyzer.Controls"
             ControlTemplate="{StaticResource CommonFrameTemplate}"
             Title="Networks Graph">
    <ScrollView>
        <StackLayout Background="Black" Margin="10">
            <Button Text="View Table" Clicked="NavigateToTablePage"
                    WidthRequest="150" HorizontalOptions="StartAndExpand" />
            <controls:NetworksFilterByGHzView FilterByGHzCommand="{Binding
FilterByGHzCommand}" />
            <StackLayout IsVisible="{Binding HasFilteredNetworks}">
                <Label Text="dBm" FontSize="24" FontAttributes="Bold"
                       HorizontalTextAlignment="Center"
VerticalTextAlignment="Center" Margin="10,0,10,10" />
                <micro:ChartView HeightRequest="500" Margin="10" Chart="{Binding
DBmChart}" />
                <Label Text="Distance" FontSize="24" FontAttributes="Bold"
                       HorizontalTextAlignment="Center"
VerticalTextAlignment="Center" Margin="10,0,10,10" />
                <micro:ChartView HeightRequest="500" Margin="10" Chart="{Binding
DistanceChart}"/>
            </StackLayout>
            <Label IsVisible="{Binding HasNoFilteredNetworks}" Text="No network found"
                   TextColor="Red" Margin="0, 20, 0, 0"
                   HorizontalOptions="Center" VerticalOptions="Center"
FontAttributes="Bold" FontSize="25"/>
        </StackLayout>
    </ScrollView>
</ContentPage>
```

- C# код Networks сторінки (графіки).

```
public partial class NetworksGraphPage : ContentPage
{
    readonly NetworksGraphViewModel networksGraphViewModel = null!;
    public NetworksGraphPage()
    {
        networksGraphViewModel = ServiceHelper.GetService<NetworksGraphViewModel>()!;
        InitializeComponent();
```

```
        BindingContext = networksGraphViewModel;
    }
    protected override void OnAppearing() {
        base.OnAppearing();
        UpdateNetworks();
    }
    async void UpdateNetworks() => await networksGraphViewModel.LoadDataAsync();
    async void NavigateToTablePage(object sender, EventArgs e)
        => await Navigation.PushAsync(new NetworksTablePage());
    protected override void OnDisappearing() {
        base.OnDisappearing();
        networksGraphViewModel.Dispose();
    }
}
```

Щоб ці всі ViewModels компоненти працювали, їх потрібно зареєструвати в MauiProgram (в головному файлі проекту):

```
        services.AddSingleton<ConnectedNetworkViewModel>();
        services.AddSingleton<NetworksTableViewModel>();
        services.AddSingleton<NetworksGraphViewModel>();
        services.AddSingleton<MainPageViewModel>();
```

Та щоб всі ці сторінки працювали, їх потрібно зареєструвати в AppShell.xaml (файл, який визначає навігаційну структуру та макет додатка:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="WiFi_Analyzer.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:WiFi_Analyzer.Pages"
    xmlns:all_networks="clr-namespace:WiFi_Analyzer.Pages.Networks"
    Shell.FlyoutBehavior="Disabled">

    <Tab Title="Home" Route="MainPage">
        <ShellContent ContentTemplate="{DataTemplate local:MainPage}" />
    </Tab>
    <Tab Title="Connected" Route="ConnectedNetworkPage">
        <ShellContent ContentTemplate="{DataTemplate local:ConnectedNetworkPage}" />
    </Tab>
    <Tab Title="Networks" Route="NetworksTablePage">
```

```
        <ShellContent ContentTemplate="{DataTemplate all_networks:NetworksTablePage}"
/>
    </Tab>
    <ShellContent Route="NetworksGraphPage"
                  ContentTemplate="{DataTemplate all_networks:NetworksGraphPage}" />
</Shell>
```

2.7. Тестування додатку

1. Home(основна) сторінка зображена рис. 2.10.

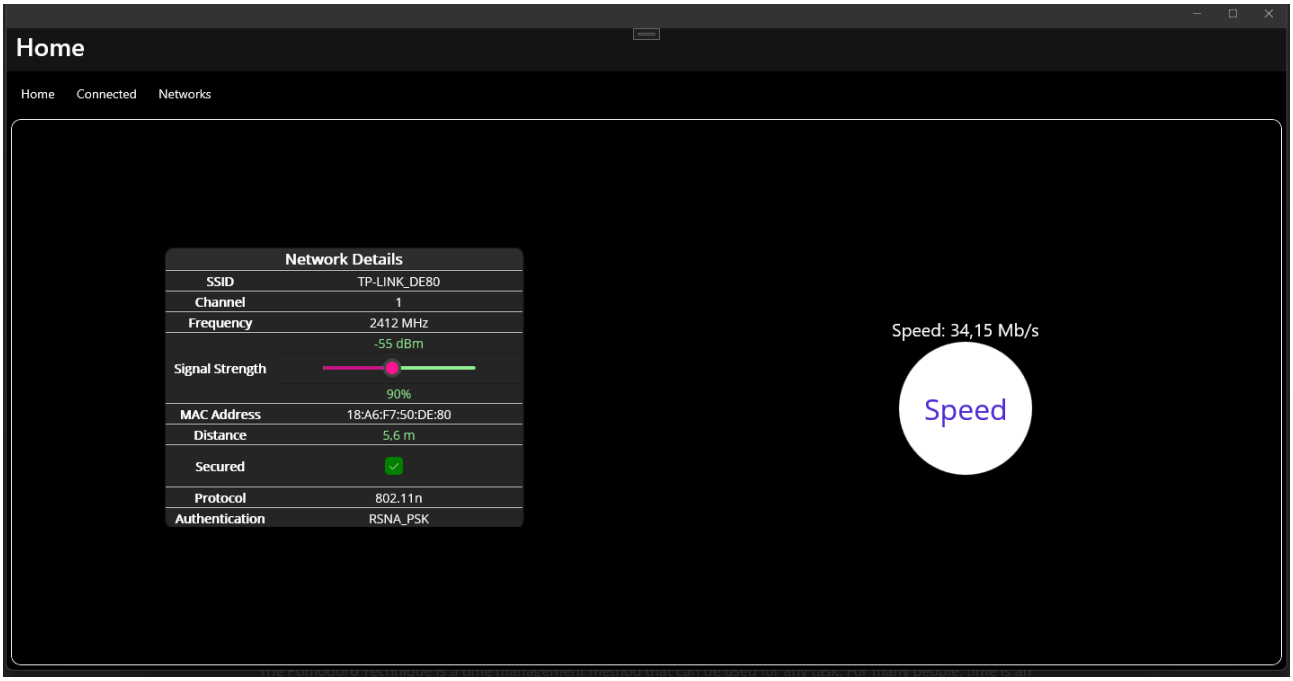

Рисунок 2.10 – Home (основна) сторінка додатку.

2. Детальна інформація характеристик поточної мережі зображена рис. 2.11.

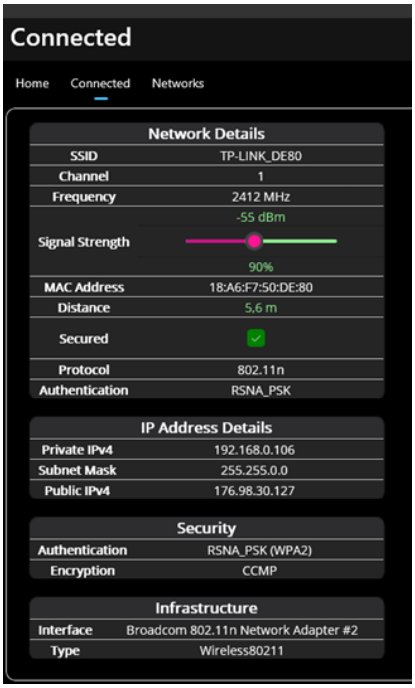

Рисунок 2.11 – Connected сторінка.

3. Всі доступні мережі:

а. Таблиця доступних мереж зображена рис. 2.12.

Рисунок 2.12 – Networks сторінка (таблиця).

Тепер можемо фільтрувати по частоті:

- 2.4 ГГц (рис. 2.13).

Рисунок 2.13 – Networks сторінка (таблиця) відфільтрована по частоті 2.4 ГГц.

- 5 ГГц (рис. 2.14).

Рисунок 2.14 – Networks сторінка (таблиця) відфільтрована по частоті 5 ГГц.

- 6 ГГц (рис. 2.15).

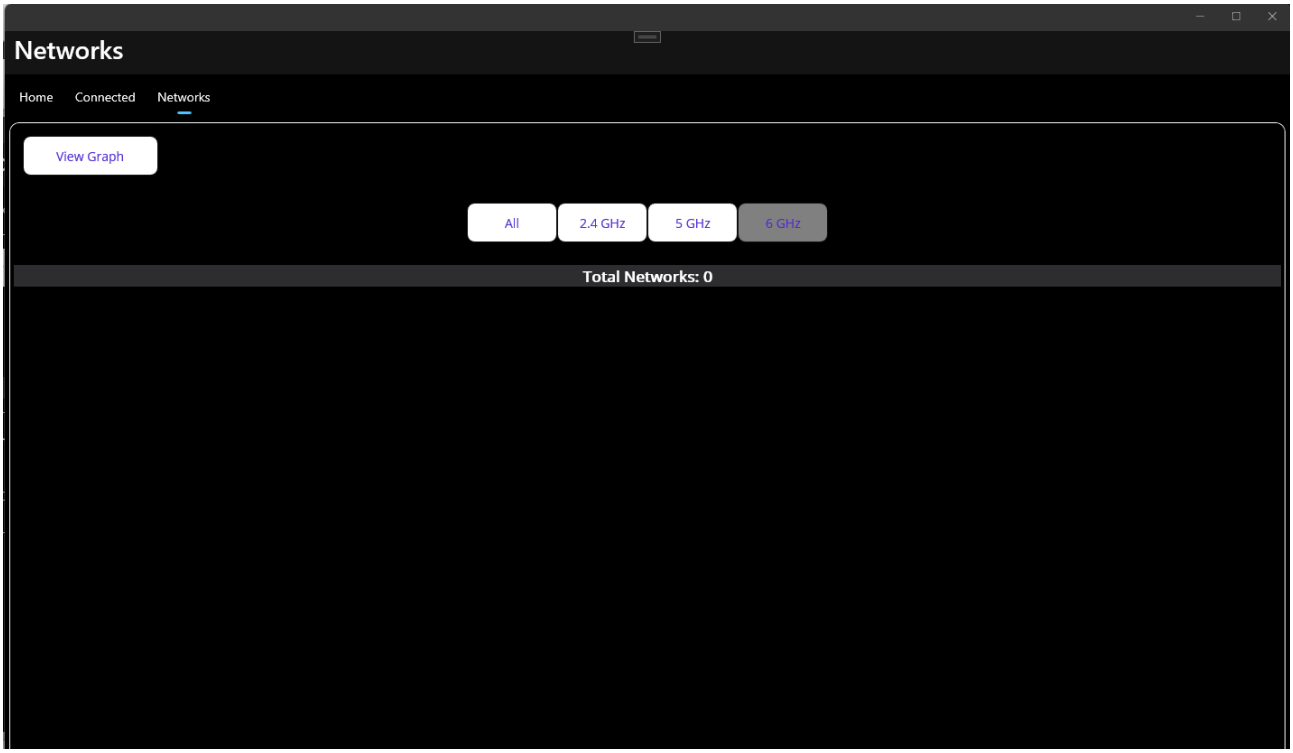

Рисунок 2.15 – Networks сторінка (таблиця) відфільтрована по частоті 6 ГГц.

б. Графіки силу сигналу та відстані від для всіх доступних мереж зображена рисунках 2.16 та 2.17.

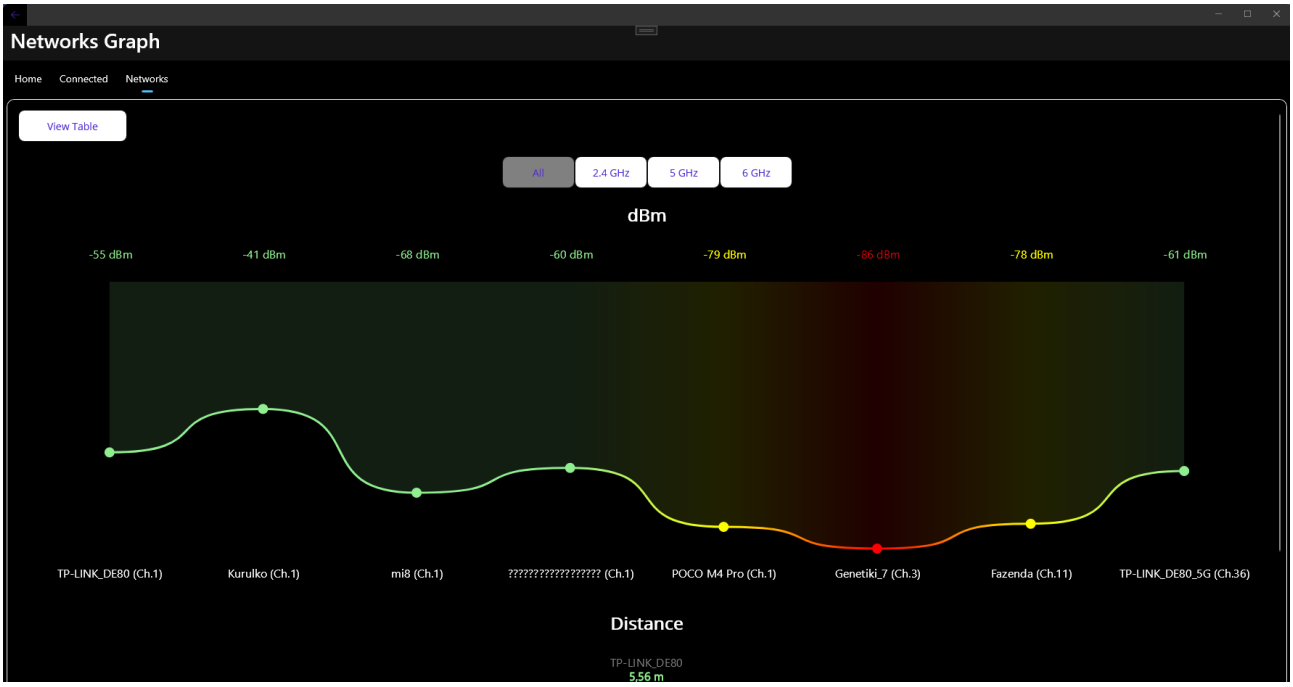

Рисунок 2.16 – Networks сторінка (графік сили сигналу мереж).

Рисунок 2.17 – Networks сторінка (графік відстаней до мережі).

Тепер можемо фільтрувати по частоті:

- 2.4 ГГц (рис. 2.18 та 2.19).

Рисунок 2.18 – Networks сторінка (графік сили сигналу мереж) відфільтрована по частоті 2.4 ГГц.

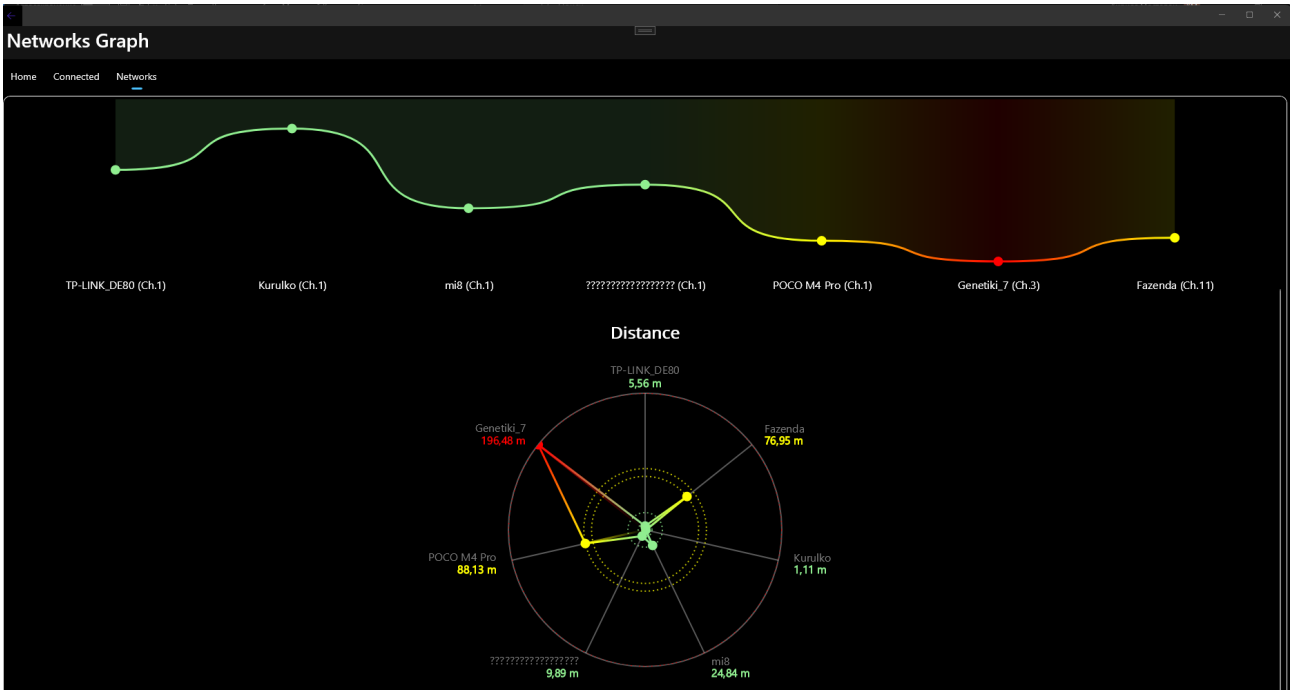

Рисунок 2.19 – Networks сторінка (графік відстаней до мережі) відфільтрована по частоті 2.4 ГГц.

- 5 ГГц (рис. 2.20 та 2.21).

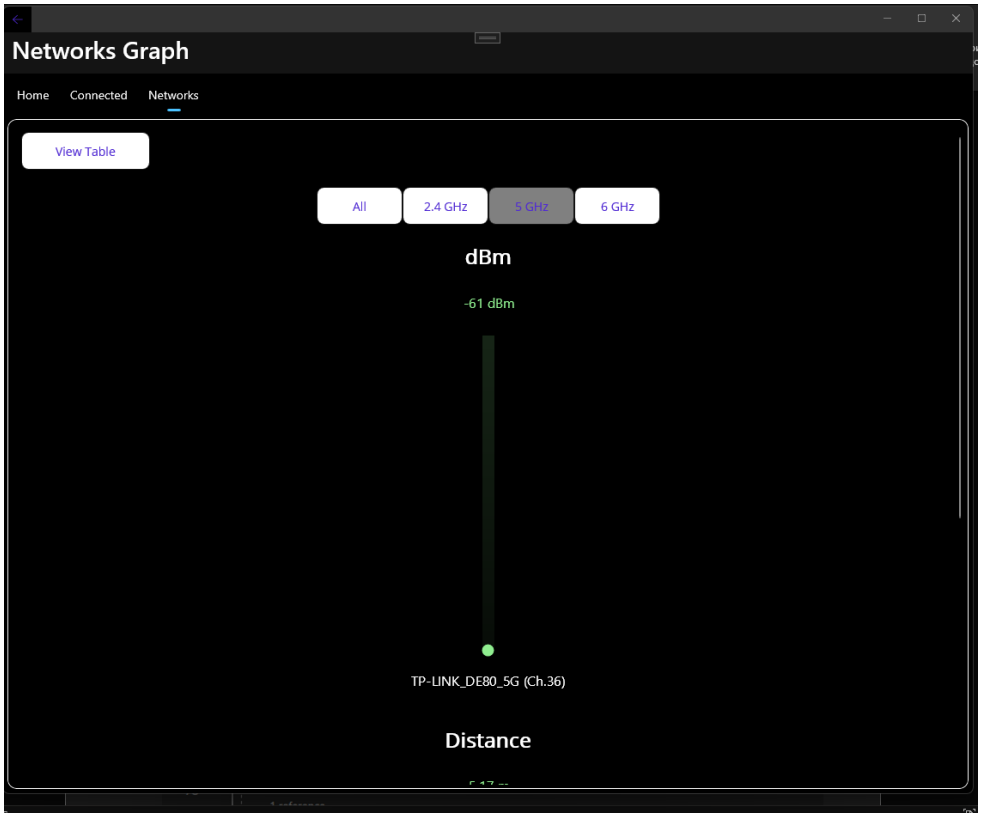

Рисунок 2.20 – Networks сторінка (графік сили сигналу мереж) відфільтрована по частоті 5 ГГц.

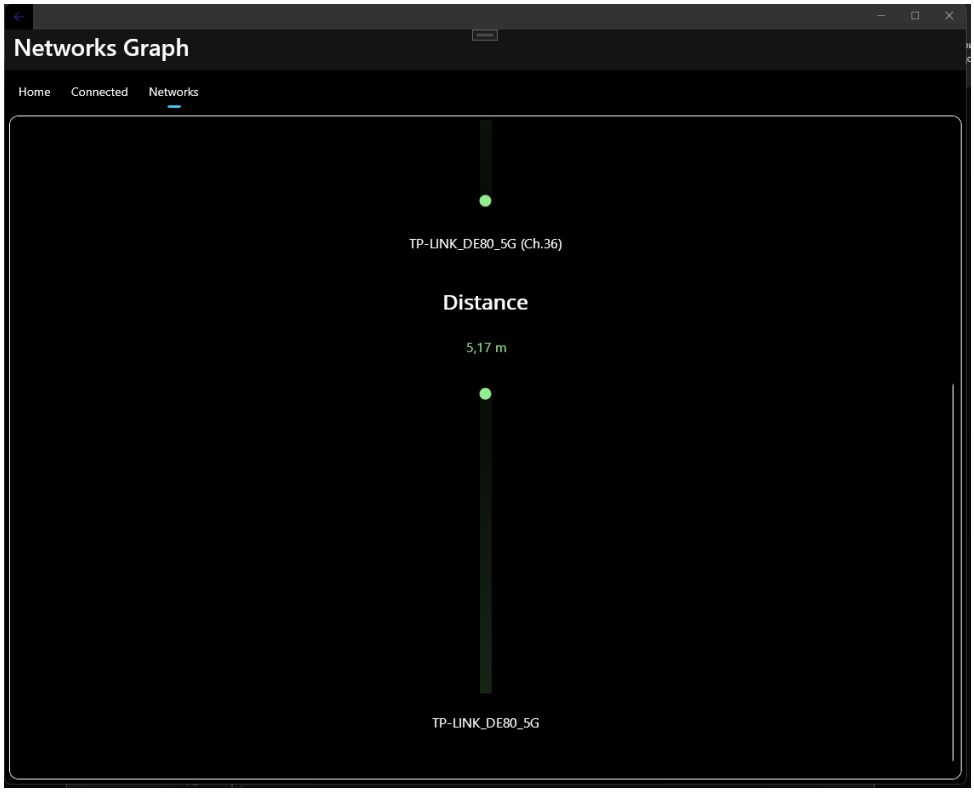

Рисунок 2.21 – Networks сторінка (графік відстаней до мережі) відфільтрована по частоті 5 ГГц.

- 6 ГГц (рис. 2.22).

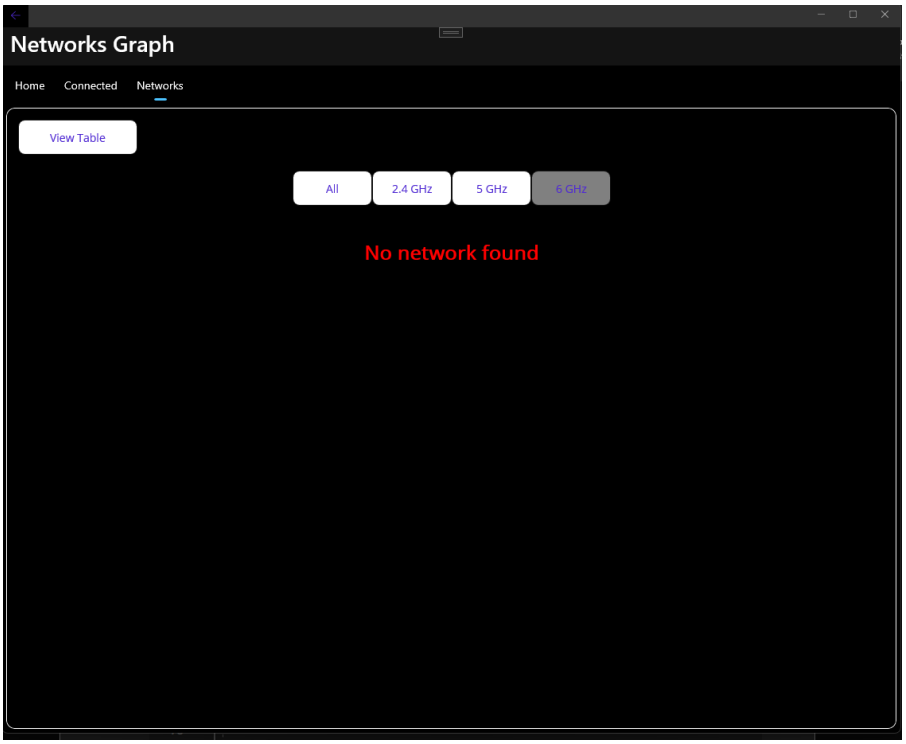

Рисунок 2.22 – Networks сторінка (графіки) відфільтрована по частоті 6 ГГц.

Також, якщо не буде підключення до Інтернету, то відобразиться повідомлення про це, та буде закрита програма автоматично, як це зображено на рис. 2.23.

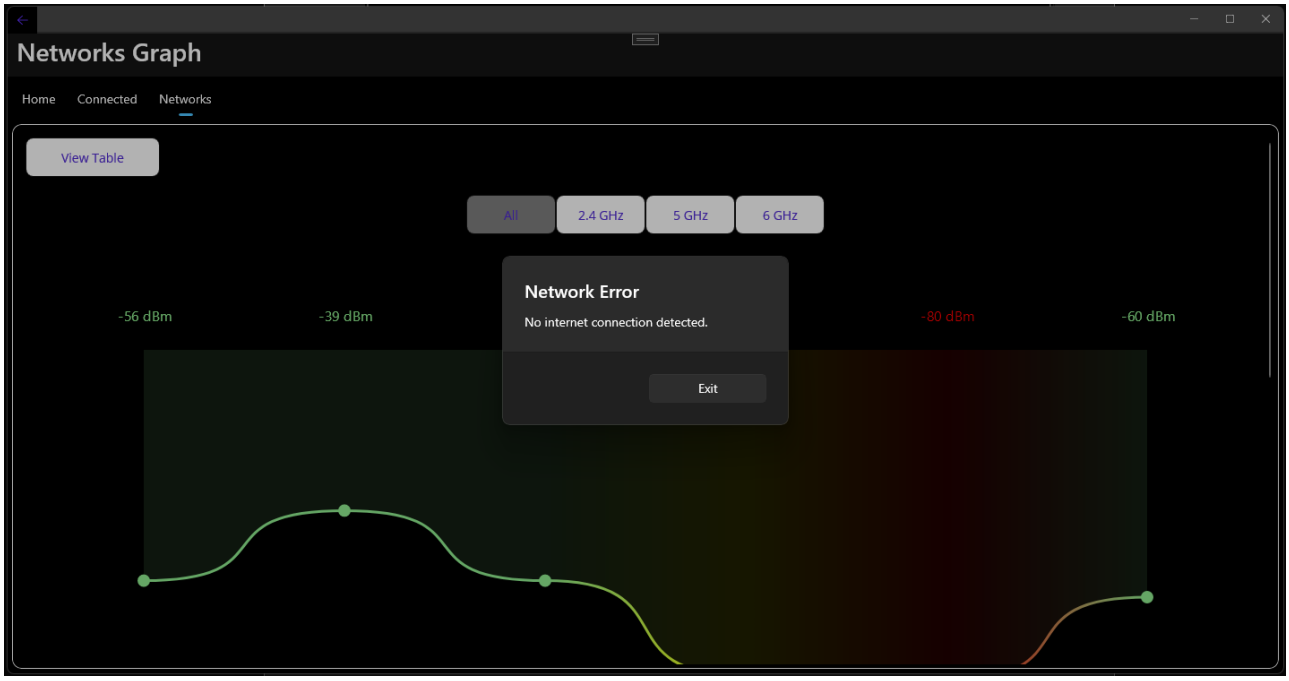

Рисунок 2.23 – Відображення повідомлення про відсутність підключення до Інтернету.

3. АНАЛІЗ РЕЗУЛЬТАТІВ

Розділ містить оцінку ефективності створеного додатку, порівняння додатку з існуючими рішеннями та результати різних сценаріїв тестування додатку.

3. 1. Оцінка ефективності додатку

Оцінюючи його ефективність, можна відзначити декілька ключових аспектів:

1. Функціональність: Додаток відповідає всім вимогам, включаючи сканування WiFi-мереж, надання детальної інформації та моніторинг продуктивності.
2. Архітектура: Використання патерну MVVM забезпечує зручну розробку, тестування та підтримку.
3. Технології та інструменти: .NET MAUI, .NET 6/7, XAML, C# дозволяють швидку розробку та багатоплатформність. Використання Visual Studio та NuGet є стандартом.
4. Інтерфейс користувача: Зручний та легкий для розширення.
5. Ефективність: Додаток працює оптимально, швидко реагує на запити та економно використовує ресурси.

Узагальнюючи, додаток відповідає всім вимогам та має всі передумови для успішної реалізації завдань, що стоять перед ним.

3.2. Порівняння з існуючими рішеннями

Порівнюючи з існуючими рішеннями, додаток, який був описаний, має кілька переваг:

1. Багатоплатформність: Додаток на .NET MAUI підтримує Windows, macOS, iOS та Android, що робить його більш доступним порівняно з WiFi Analyzer (для Windows, macOS та Linux) та inSSIDer (лише для Windows та macOS).
2. Просунуті функції: У порівнянні з WiFi Analyzer та inSSIDer, додаток на .NET MAUI має деякі просунуті можливості, такі як вимірювання відстані до мережі, фільтрація за частотою 6 ГГц та вимірювання швидкості завантаження в реальному часі.
3. Ефективність та простота використання: Завдяки патерну MVVM та .NET MAUI, додаток є ефективним і простим у використанні, подібно до інших розглянутих рішень.
4. Безкоштовність: Додаток є безкоштовним, як і WiFi Analyzer та inSSIDer.
5. Користувацький інтерфейс: Додаток має зручний та інтуїтивно зрозумілий інтерфейс.

Загалом, додаток має потенціал стати конкурентоспроможним у порівнянні з існуючими рішеннями, завдяки своїм функціям та простоті використання.

3.3. Обговорення результатів тестування

Під час тестування виявлено, що додаток надзвичайно корисний для виявлення та аналізу WiFi-мереж. Він точно визначає доступні мережі та надає ключову інформацію про них, що дозволяє ефективно аналізувати якість сигналу, швидкість та інші параметри для оптимізації мереж.

Особливу увагу заслуговує можливість вимірювати відстань до мережі, що допомагає користувачам покращити розташування маршрутизаторів і забезпечити краще підключення. Функції для аналізу мережевих проблем та створення звітів, виявлені під час тестування, також відзначаються своєю корисністю та легкістю використання.

Загалом, результати тестування підтверджують високу ефективність та корисність додатку, роблячи його привабливим для широкого кола користувачів, що прагнуть оптимізувати свої WiFi-мережі.

ВИСНОВКИ

В процесі розробки програмного аналізатора для діагностики та аналізу стану WiFi-мереж було створено зручний інструмент, який поєднує найкращі характеристики існуючих рішень та вдосконалює їх, додаючи новий функціонал. Програма дозволяє користувачам зручно відстежувати та аналізувати параметри WiFi-мереж, а також проводити базовий аналіз безпеки мереж, що сприяє їх оптимізації та забезпечує стабільність.

Додаток реалізує збір даних про мережі, аналізує сигнал та відображує результати користувачеві. Використання асинхронних методів забезпечило швидку та ефективну роботу програми, гарантуючи плавний користувацький досвід.

Розроблено інтуїтивно зрозумілий інтерфейс, що спрощує навігацію та дозволяє користувачам легко використовувати програму.

Надійність та стабільність роботи забезпечується механізмами обробки винятків і відображення повідомлень про помилки. Крім того, впроваджені можливості розширення функціоналу через сервіси та інтеграції дозволяє достатньо просто додавати нові функції.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wired Equivalent Privacy (WEP). URL: https://en.wikipedia.org/wiki/Wired_Equivalent_Privacy (Last accessed: 18.06.2024).
2. Wi-Fi Protected Access (WPA). URL: https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access (Last accessed: 18.06.2024).
3. Wi-Fi Protected Access 2 (WPA2). URL: https://www.howtogeek.com/204697/wi-fi-security-should-you-use-wpa2-aes-wpa2-tkip-or-both/ (Last accessed: 18.06.2024).
4. Wi-Fi Protected Access 3 (WPA3). URL: https://www.pcmag.com/explainers/what-is-wpa3-secure-wifi-how-to-set-it-up-on-your-router (Last accessed: 18.06.2024).
5. Temporal Key Integrity Protocol (TKIP). URL: https://en.wikipedia.org/wiki/Temporal_Key_Integrity_Protocol (Last accessed: 18.06.2024).
6. Advanced Encryption Standard (AES). URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard (Last accessed: 18.06.2024).
7. WPA3 Encryption and Configuration. URL: https://documentation.meraki.com/MR/Wi-Fi_Basics_and_Best_Practices/WPA3_Encryption_and_Configuration_Guide (Last accessed: 18.06.2024).
8. Extensible Authentication Protocol (EAP). URL: https://en.wikipedia.org/wiki/Extensible_Authentication_Protocol (Last accessed: 18.06.2024).
9. Wireless Standards. URL: https://www.makeuseof.com/tag/understanding-common-wifi-standards-technology-explained/ (Last accessed: 18.06.2024).
10. WiFi Analyzer. URL: https://apps.microsoft.com/detail/9nblggh33n0n?hl=en-us&gl=UA (Last accessed: 18.06.2024).

11. InSSIDer. URL: https://inssider.en.softonic.com/ (Last accessed: 18.06.2024).
12. Model-View-ViewModel (MVVM). URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (Last accessed: 18.06.2024).
13. Model-View-ViewModel (MVVM) pattern. URL: https://www.plainionist.net/Mvvm-Dialogs/ (Last accessed: 18.06.2024).

ДОДАТОК А
ЛІСТИНГ ПРОГРАМИ АНАЛІЗАТОРА WI-FI МЕРЕЖ

Програма для аналізу WiFi мереж написана мовою C# для платформи .NET MAUI:

- **MauiProgram.cs:**

```
using Microcharts.Maui;
using Microsoft.Extensions.Logging;
using WiFi_Analyzer.Helpers;
using WiFi_Analyzer.Providers;
namespace WiFi_Analyzer;
public static class MauiProgram
{
        public static MauiApp CreateMauiApp()
        {
                MauiAppBuilder builder = MauiApp.CreateBuilder();
    builder
      .UseMauiApp<App>()
      .UseMicrocharts()
      .ConfigureFonts(fonts =>
      {
        fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
        fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
      });
    IServiceCollection services = builder.Services;
    services.AddMSSQLServer();
    services.AddServices();
    services.AddViewModels();
#if DEBUG
    builder.Logging.AddDebug();
#endif
    MauiApp app = builder.Build();
    ServiceHelper.Initialize(app.Services);
    return app;
  }
}
```

- **AppShell.xaml:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
  x:Class="WiFi_Analyzer.AppShell"
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:WiFi_Analyzer.Pages"
  xmlns:all_networks="clr-namespace:WiFi_Analyzer.Pages.Networks"
  Shell.FlyoutBehavior="Disabled">
  <Tab Title="Home" Route="MainPage">
    <ShellContent ContentTemplate="{DataTemplate local:MainPage}" />
  </Tab>
  <Tab Title="Connected" Route="ConnectedNetworkPage">
    <ShellContent ContentTemplate="{DataTemplate local:ConnectedNetworkPage}" />
  </Tab>
  <Tab Title="Networks" Route="NetworksTablePage">
    <ShellContent ContentTemplate="{DataTemplate all_networks:NetworksTablePage}" />
  </Tab>
```

```
   <ShellContent Route="NetworksGraphPage"
            ContentTemplate="{DataTemplate all_networks:NetworksGraphPage}" />
</Shell>
```

- **AppShell.xaml.cs:**

```
using WiFi_Analyzer.Helpers;
namespace WiFi_Analyzer;
public partial class AppShell : Shell
{
          public AppShell()
     => InitializeComponent();
   protected override async void OnAppearing()
   {
     if (!Connectivity.Current.NetworkAccess.Equals(NetworkAccess.Internet))
     {
       await ErrorHandler.DisplayNetworkErrorAsync("No internet connection detected.");
       Environment.Exit(0);
     }
     base.OnAppearing();
   }
}
```

- **App.xaml:**

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         xmlns:local="clr-namespace:WiFi_Analyzer"
         x:Class="WiFi_Analyzer.App">
   <Application.Resources>
     <ResourceDictionary>
       <ResourceDictionary.MergedDictionaries>
         <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
         <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
         <ResourceDictionary Source="AppResources.xaml" />
       </ResourceDictionary.MergedDictionaries>
     </ResourceDictionary>
   </Application.Resources>
</Application>
```

- **App.xaml.cs:**

```
using WiFi_Analyzer.Pages;
namespace WiFi_Analyzer;
public partial class App : Application
{
   public App()
   {
     InitializeComponent();
     MainPage = new AppShell();
   }
}
```

- **appsetting.json:**

```
{
 "ConnectionStrings": {
   "DefaultConnection": "Server=(localdb)\\mssqllocaldb; Database=WiFi_Analyzer_3; MultipleActiveResultSets=True;"
 }
}
```

- **IEntityBase.cs:**

```
using System.ComponentModel.DataAnnotations;
namespace WiFi_Analyzer.Models;
public interface IEntityBase
{
   [Key]
```

```
    long Id { get; set; }
}
```

- **WiFiNetwork.cs:**

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using WiFi_Analyzer.Extensions;
using static NativeWifi.Wlan;
namespace WiFi_Analyzer.Models;
[Index(nameof(MacAddress), IsUnique = true)]
public class WiFiNetwork : IEntityBase
{
    public long Id { get; set; }
    [Required]
    public string SSID { get; set; } = null!;
    [Required]
    public string Protocol { get; set; } = null!;
    public DateTime LastSeen { get; set; }
    [Required]
    public byte[] MacAddress { get; set; } = null!;
    [NotMapped]
    public string StringMacAddress => MacAddress.MacAddressToString();
    [Column("Frequency")]
    public long FrequencyInHz { get; set; }
    public int FrequencyInkHz => (int)(FrequencyInHz / Math.Pow(10, 3));
    public int FrequencyInMHz => (int)(FrequencyInHz / Math.Pow(10, 6));
    public int FrequencyInGHz => (int)(FrequencyInHz / Math.Pow(10, 9));
    public int Channel { get; set; }
    [Column("Secured")]
    public bool IsSecured { get; set; }
    [Column("Authentication")]
    public Dot11AuthAlgorithm AuthenticationAlgorithm { get; set; }
    [NotMapped]
    public NetworkStates? NetworkStates { get; set; }
    [NotMapped]
    public IPAddressInfo? IPAddressInfo { get; set; }
    [NotMapped]
    public NetworkSecurityInfo? NetworkSecurityInfo { get; set; }
    [NotMapped]
    public NetworkInfrastructureInfo? NetworkInfrastructureInfo { get; set; }
}
```

- **DownloadSpeed.cs:**

```
using SpeedTest.Net.Enums;
using WiFi_Analyzer.Extensions;
namespace WiFi_Analyzer.Models;
public class DownloadSpeed
{
    public double Speed { get; set; }
    public SpeedTestUnit Unit { get; set; }
    public static explicit operator DownloadSpeed(SpeedTest.Net.Models.DownloadSpeed _downloadSpeed)
    {
        DownloadSpeed downloadSpeed = new();
        downloadSpeed.Speed = _downloadSpeed.Speed;
        downloadSpeed.Unit = _downloadSpeed.Unit.ParseToSpeedTestUnit();
        return downloadSpeed;
    }
}
```

- **IPAddressInfo.cs:**

```
namespace WiFi_Analyzer.Models;
public class IPAddressInfo
```

```
{
   public string PrivateIPv4 { get; set; } = null!;
   public string SubnetMask { get; set; } = null!;
   public string? PublicIPv4 { get; set; }
}
```

- **NetworkInfrastructureInfo.cs:**

```
using System.Net.NetworkInformation;
namespace WiFi_Analyzer.Models;
public class NetworkInfrastructureInfo
{
   public string Interface { get; set; } = null!;
   public NetworkInterfaceType InterfaceType { get; set; }
   public OperationalStatus OperationalStatus { get; set; }
}
```

- **NetworkSecurityInfo.cs:**

```
using static NativeWifi.Wlan;
namespace WiFi_Analyzer.Models;
public class NetworkSecurityInfo
{
   public Dot11AuthAlgorithm Authentication { get; set; }
   public Dot11CipherAlgorithm Encryption { get; set; }
}
```

- **NetworkStates.cs:**

```
namespace WiFi_Analyzer.Models;
public class NetworkStates
{
   public bool IsConnected { get; set; }
   public int SignalStrengthIndBm { get; set; }
   public int SignalStrengthInPercentage
   {
      get
      {
         const int minRssi = -100; // Minimum RSSI value
         if (SignalStrengthIndBm <= minRssi)
            return 0;
         const int maxRssi = -50;  // Maximum RSSI value
         if (SignalStrengthIndBm >= maxRssi)
            return 100;
         return 2 * (SignalStrengthIndBm + 100);
      }
   }
   public double DistanceInMeters { get; set; }
}
```

- **ViewModelBase.cs:**

```
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using WiFi_Analyzer.Helpers;
namespace WiFi_Analyzer.ViewModels;

public abstract class ViewModelBase : INotifyPropertyChanged, IDisposable
{
   Timer? networkStateTimer;
   public ViewModelBase()
      => networkStateTimer = new Timer(UpdateStates, null, TimeSpan.Zero, TimeSpan.FromSeconds(30));
   protected abstract void UpdateStates(object? _ = null);
   public event PropertyChangedEventHandler? PropertyChanged;
   public ICommand LoadDataCommand => new Command(async () =>
```

```
    {
        try
        {
            await LoadDataAsync();
        }
        catch (Exception ex)
        {
            await Shell.Current.DisplayAlert("Error", ex.Message, "OK");
        }
    });

    public async Task LoadDataAsync()
    {
        try
        {
            await GetDataAsync();
        }
        catch (Exception ex)
        {
            await ErrorHandler.DisplayErrorAsync(ex.Message);
        }
    }
    protected abstract Task GetDataAsync();
    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = "")
        => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    public void Dispose()
        => networkStateTimer?.Dispose();
}
```

- **NetworkViewModel.cs:**

```
using WiFi_Analyzer.Models;
using WiFi_Analyzer.Services.ConnectedNetwork;
using WiFi_Analyzer.Services.Networks;
namespace WiFi_Analyzer.ViewModels.Network;
public abstract class NetworkViewModel : ViewModelBase
{
    protected readonly IConnectedNetworkService connectedNetworkService;
    protected readonly INetworksService networksService;
    WiFiNetwork? connectedNetwork;
    public WiFiNetwork? ConnectedNetwork
    {
        get => connectedNetwork;
        set
        {
            connectedNetwork = value;
            OnPropertyChanged(nameof(ConnectedNetwork));
        }
    }

    NetworkStates? networkStates;
    public NetworkStates? NetworkStates
    {
        get => networkStates;
        set
        {
            networkStates = value;
            OnPropertyChanged(nameof(NetworkStates));
        }
    }
    public NetworkViewModel(IConnectedNetworkService connectedNetworkService, INetworksService
networksService)
```

```
    => (this.connectedNetworkService, this.networksService) = (connectedNetworkService, networksService);
  protected override void UpdateStates(object? state = null)
    => NetworkStates = connectedNetworkService.GetConnectedNetworkStates();
  protected override async Task GetDataAsync()
  {
    ConnectedNetwork = connectedNetworkService.GetConnectedWiFiNetwork();
    await networksService.UpdateWiFiNetworkAsync(ConnectedNetwork);
    UpdateStates();
  }
}
```

- **MainPageViewModel.cs:**

```
using System.Windows.Input;
using WiFi_Analyzer.Helpers;
using WiFi_Analyzer.Models;
using WiFi_Analyzer.Services.ConnectedNetwork;
using WiFi_Analyzer.Services.Networks;
using WiFi_Analyzer.Services.SpeedTest;
using WiFi_Analyzer.ViewModels.Network;
namespace WiFi_Analyzer.ViewModels;
public class MainPageViewModel : NetworkViewModel
{
  readonly ISpeedTestService speedTestService;
  DownloadSpeed? downloadSpeed;
  public DownloadSpeed? DownloadSpeed
  {
    get => downloadSpeed;
    set
    {
      downloadSpeed = value;
      OnPropertyChanged(nameof(DownloadSpeed));
    }
  }
  bool isBusy;
  public bool IsBusy
  {
    get => isBusy;
    set
    {
      isBusy = value;
      OnPropertyChanged(nameof(IsBusy));
    }
  }
  public ICommand GetDownloadSpeedCommand => new Command(async () => await GetDownloadSpeedAsync());
  public MainPageViewModel(IConnectedNetworkService connectedNetworkService, ISpeedTestService
speedTestService, INetworksService networksService) : base(connectedNetworkService, networksService)
    => this.speedTestService = speedTestService;
  public async Task GetDownloadSpeedAsync()
  {
    try
    {
      DownloadSpeed = null;
      IsBusy = true;
      DownloadSpeed = await speedTestService.GetDownloadSpeedAsync();
      IsBusy = false;
    }
    catch (Exception ex)
    {
      await ErrorHandler.DisplayErrorAsync(ex.Message);
    }
  }
```

```
}
```

- **ConnectedNetworkViewModel.cs:**

```
using WiFi_Analyzer.Models;
using WiFi_Analyzer.Services.ConnectedNetwork;
using WiFi_Analyzer.Services.Networks;
using WiFi_Analyzer.ViewModels.Network;
namespace WiFi_Analyzer.ViewModels;
public class ConnectedNetworkViewModel : NetworkViewModel
{
  IPAddressInfo? _IPAddressInfo;
  public IPAddressInfo? IPAddressInfo
  {
    get => _IPAddressInfo;
    set
    {
      _IPAddressInfo = value;
      OnPropertyChanged(nameof(IPAddressInfo));
    }
  }
  NetworkSecurityInfo? networkSecurityInfo;
  public NetworkSecurityInfo? NetworkSecurityInfo
  {
    get => networkSecurityInfo;
    set
    {
      networkSecurityInfo = value;
      OnPropertyChanged(nameof(NetworkSecurityInfo));
    }
  }
  NetworkInfrastructureInfo? networkInfrastructureInfo;
  public NetworkInfrastructureInfo? NetworkInfrastructureInfo
  {
    get => networkInfrastructureInfo;
    set
    {
      networkInfrastructureInfo = value;
      OnPropertyChanged(nameof(NetworkInfrastructureInfo));
    }
  }
  public ConnectedNetworkViewModel(IConnectedNetworkService connectedNetworkService, INetworksService
networksService) : base(connectedNetworkService, networksService) { }
  protected override async Task GetDataAsync()
  {
    await base.GetDataAsync();
    IPAddressInfo = await connectedNetworkService.GetConnectedIPAddressInfo();
    NetworkSecurityInfo = connectedNetworkService.GetConnectedNetworkSecurityInfo();
    NetworkInfrastructureInfo = connectedNetworkService.GetConnectedNetworkInfrastructureInfo();
  }
}
```

- **NetworksViewModel.cs:**

```
using System.Windows.Input;
using WiFi_Analyzer.Helpers;
using WiFi_Analyzer.Models;
using WiFi_Analyzer.Services.Networks;
namespace WiFi_Analyzer.ViewModels;
public abstract class NetworksViewModel : ViewModelBase
{
  protected readonly INetworksService networksService;
  public IEnumerable<WiFiNetwork> Networks = Enumerable.Empty<WiFiNetwork>();
  protected IEnumerable<WiFiNetwork> filteredWiFiNetworks = Enumerable.Empty<WiFiNetwork>();
```

```
    public virtual IEnumerable<WiFiNetwork> FilteredWiFiNetworks
    {
        get => filteredWiFiNetworks;
        set
        {
            filteredWiFiNetworks = value;
            OnPropertyChanged(nameof(FilteredWiFiNetworks));
            OnPropertyChanged(nameof(HasFilteredNetworks));
        }
    }
    public bool HasFilteredNetworks => FilteredWiFiNetworks.Any();
    public ICommand FilterByGHzCommand => new Command<string>(parameter => FilterByGHz(parameter));
    public NetworksViewModel(INetworksService networksService)
        => this.networksService = networksService;
    protected override async Task GetDataAsync()
    {
        await networksService.UpdateWiFiNetworksAsync();
        FilteredWiFiNetworks = Networks = await networksService.GetWiFiNetworksWithStatesAsync();
    }
    protected override async void UpdateStates(object? state = null)
        => FilteredWiFiNetworks = Networks = await networksService.GetWiFiNetworksWithStatesAsync();

    protected void FilterByGHz(string parameter)
        => FilteredWiFiNetworks = NetworksFilter.FilterByGHz(Networks, parameter);
}
```

- **NetworksTableViewModel.cs:**

```
using System.Windows.Input;
using WiFi_Analyzer.Enums;
using WiFi_Analyzer.Services.Networks;
using WiFi_Analyzer.Extensions;
namespace WiFi_Analyzer.ViewModels;
public class NetworksTableViewModel : NetworksViewModel
{
    public ICommand SortCommand => new Command<string>(propertyName => SortBy(propertyName));
    public NetworksTableViewModel(INetworksService networksService) : base(networksService)
    {   }
    public Dictionary<string, OrderBy> SortDirections { get; set; } = new();
    void SortBy(string propertyName)
    {
        OrderBy currentDirection = SortDirections.ContainsKey(propertyName) ? SortDirections[propertyName] :
OrderBy.Ascending;
        OrderBy newDirection = ChangeOrderBy(currentDirection);
        SortDirections = new Dictionary<string, OrderBy>();
        SortDirections[propertyName] = newDirection;
        FilteredWiFiNetworks = FilteredWiFiNetworks.AsQueryable().DynamicOrderBy(propertyName,
newDirection).ToList();
        OnPropertyChanged(nameof(SortDirections));
    }
    OrderBy ChangeOrderBy(OrderBy orderBy)
        => orderBy == OrderBy.Ascending ? OrderBy.Descending : OrderBy.Ascending;
}
```

- **NetworksGraphViewModel.cs:**

```
using Microcharts;
using SkiaSharp;
using WiFi_Analyzer.Helpers;
using WiFi_Analyzer.Models;
using WiFi_Analyzer.Services.Networks;
namespace WiFi_Analyzer.ViewModels;
public class NetworksGraphViewModel : NetworksViewModel
{
```

```
public bool HasNoFilteredNetworks => !HasFilteredNetworks;
Chart dBmChart = null!;
public Chart DBmChart
{
   get => dBmChart;
   set
   {
      dBmChart = value;
      OnPropertyChanged(nameof(DBmChart));
   }
}
Chart distanceChart = null!;
public Chart DistanceChart
{
   get => distanceChart;
   set
   {
      distanceChart = value;
      OnPropertyChanged(nameof(DistanceChart));
   }
}
public override IEnumerable<WiFiNetwork> FilteredWiFiNetworks
{
   get => filteredWiFiNetworks;
   set
   {
      filteredWiFiNetworks = value;
      OnPropertyChanged(nameof(FilteredWiFiNetworks));
      OnPropertyChanged(nameof(HasFilteredNetworks));
      OnPropertyChanged(nameof(HasNoFilteredNetworks));
      UpdateCharts();
   }
}
public NetworksGraphViewModel(INetworksService networksService) : base(networksService)
{   }
void UpdateCharts()
{
   if (HasFilteredNetworks)
   {
      DBmChart = GetDBmChartView();
      DistanceChart = GetDistanceChartView();
   }
   else
   {
      DBmChart = null!;
      DistanceChart = null!;
   }
}
readonly SKColor backgroundColor = SKColor.Parse("#000000");
readonly TimeSpan animationDuration = new (0, 0, 2);
readonly SKColor labelColor = SKColor.Parse("#FFFFFF");
public Chart GetDBmChartView()
{
   var entries = GetDBMEntries();
   Chart chart = entries.Count() == 1 ?
      new PointChart()
      {
         Entries = entries,
         BackgroundColor = backgroundColor,
         AnimationDuration = animationDuration,
```

```
            LabelColor = labelColor,
            LabelOrientation = Orientation.Horizontal,
            ValueLabelOrientation = Orientation.Horizontal,
        } :
        new LineChart()
        {
            Entries = entries,
            BackgroundColor = backgroundColor,
            AnimationDuration = animationDuration,
            LabelColor = labelColor,
            LabelOrientation = Orientation.Horizontal,
            ValueLabelOrientation = Orientation.Horizontal,
        };
    return chart;
}
public Chart GetDistanceChartView()
{
    var entries = GetDistanceEntries();
    Chart chart = entries.Count() == 1 ?
        new PointChart() {
            Entries = entries,
            BackgroundColor = backgroundColor,
            AnimationDuration = animationDuration,
            LabelColor = labelColor,
            LabelOrientation = Orientation.Horizontal,
            ValueLabelOrientation = Orientation.Horizontal,
        } :
        new RadarChart() {
            Entries = entries,
            BackgroundColor = backgroundColor,
            AnimationDuration = animationDuration,
            LabelColor = labelColor,
        };
    return chart;
}
IEnumerable<ChartEntry> GetDBMEntries()
{
    IList<ChartEntry> entries = new List<ChartEntry>();

    foreach (WiFiNetwork network in FilteredWiFiNetworks.OrderBy(n => n.Channel))
    {
        NetworkStates? networkStates = network.NetworkStates;
        if (networkStates is not null)
        {
            int signalStrength = networkStates.SignalStrengthIndBm;
            SKColor color = GetColorBySignalStrength(signalStrength);

            entries.Add(new ChartEntry(signalStrength)
            {
                Label = $"{network.SSID} (Ch.{network.Channel})",
                ValueLabel = $"{signalStrength} dBm",
                ValueLabelColor = color,
                Color = color
            });
        }
    }
    return entries;
}
IEnumerable<ChartEntry> GetDistanceEntries()
{
```

```
        IList<ChartEntry> entries = new List<ChartEntry>();
        foreach (var network in FilteredWiFiNetworks)
        {
            NetworkStates? networkStates = network.NetworkStates;
            if (networkStates is not null)
            {
                SKColor color = GetColorBySignalStrength(networkStates.SignalStrengthIndBm);
                double distance = networkStates.DistanceInMeters;
                entries.Add(new ChartEntry((float)distance)
                {
                    Label = network.SSID,
                    ValueLabel = $"{Math.Round(distance, 2)} m",
                    ValueLabelColor = color,
                    Color = color
                });
            }
        }
        return entries;
    }
    SKColor GetColorBySignalStrength(int signalStrengthIndBm)
        => SKColor.Parse(SingalColorHelper.GetHexColorBySingalStrength(signalStrengthIndBm));
}
```

- **ISpeedTestService.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Services.SpeedTest;
public interface ISpeedTestService
{
    Task<DownloadSpeed> GetDownloadSpeedAsync();
}
```

- **SpeedTestService.cs:**

```
using SpeedTest.Net.Enums;
using SpeedTest.Net;
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Services.SpeedTest;
public class SpeedTestService : ISpeedTestService
{
    public async Task<DownloadSpeed> GetDownloadSpeedAsync()
        => (DownloadSpeed)await FastClient.GetDownloadSpeed(SpeedTestUnit.MegaBitsPerSecond);
}
```

- **NetworkService.cs:**

```
using SpeedTest.Net.Enums;
using SpeedTest.Net;
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Services.SpeedTest;
public class SpeedTestService : ISpeedTestService
{
    public async Task<DownloadSpeed> GetDownloadSpeedAsync()
        => (DownloadSpeed)await FastClient.GetDownloadSpeed(SpeedTestUnit.MegaBitsPerSecond);
}
```

- **IConnectedNetworkService.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Services.ConnectedNetwork;
public interface IConnectedNetworkService
{
    WiFiNetwork GetConnectedWiFiNetwork();
    NetworkStates GetConnectedNetworkStates();
    Task<IPAddressInfo> GetConnectedIPAddressInfo();
    NetworkSecurityInfo GetConnectedNetworkSecurityInfo();
    NetworkInfrastructureInfo GetConnectedNetworkInfrastructureInfo();
```

}

- **ConnectedNetworkService.cs:**

```
using NativeWifi;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using System.Net;
using WiFi_Analyzer.Models;
using static NativeWifi.Wlan;
namespace WiFi_Analyzer.Services.ConnectedNetwork;
public class ConnectedNetworkService : NetworkService, IConnectedNetworkService
{
  WlanBssEntry GetConnectedWlanBssEntry()
  {
    WlanClient client = new();

    foreach (WlanClient.WlanInterface wlanInterface in client.Interfaces)
    {
      if (wlanInterface.InterfaceState == WlanInterfaceState.Connected)
      {
        Dot11Ssid ssid = wlanInterface.CurrentConnection.wlanAssociationAttributes.dot11Ssid;
        WlanBssEntry[] bssEntries = wlanInterface.GetNetworkBssList();

        foreach (WlanBssEntry bssEntry in bssEntries)
        {
          string currentSSID = GetStringForSSID(ssid);
          string bssEntrySSID = GetStringForSSID(bssEntry.dot11Ssid);

          if (currentSSID == bssEntrySSID)
            return bssEntry;
        }
      }
    }
    throw new Exception("No internet connection detected.");
  }
  public NetworkStates GetConnectedNetworkStates()
  {
    WlanBssEntry connectedBssEntry = GetConnectedWlanBssEntry();
    long frequency = GetFrequencyFromChannel(connectedBssEntry.chCenterFrequency);
    int signalStrength = connectedBssEntry.rssi;
    NetworkStates networkStates = new();
    networkStates.DistanceInMeters = CalculateDistance(signalStrength, frequency);
    networkStates.IsConnected = true;
    networkStates.SignalStrengthIndBm = signalStrength;
    return networkStates;
  }
  public WiFiNetwork GetConnectedWiFiNetwork()
  {
    WlanBssEntry connectedBssEntry = GetConnectedWlanBssEntry();
    WiFiNetwork wiFiNetwork = new();
    long frequency = GetFrequencyFromChannel(connectedBssEntry.chCenterFrequency);
    string currentSSID = GetStringForSSID(connectedBssEntry.dot11Ssid);
    wiFiNetwork.SSID = currentSSID;
    wiFiNetwork.Channel = GetChannelFromFrequency(frequency);
    wiFiNetwork.FrequencyInHz = frequency;
    wiFiNetwork.Protocol = FindProtocolString(connectedBssEntry);
    wiFiNetwork.MacAddress = connectedBssEntry.dot11Bssid;
    WlanAvailableNetwork wlanAvailableNetwork = GetWlanAvailableNetworkByProfileName(currentSSID)!.Value;
    wiFiNetwork.IsSecured = wlanAvailableNetwork.securityEnabled;
    wiFiNetwork.AuthenticationAlgorithm = wlanAvailableNetwork.dot11DefaultAuthAlgorithm;
    return wiFiNetwork;
```

```
}
public async Task<IPAddressInfo> GetConnectedIPAddressInfo()
{
   IPAddressInfo iPAddressInfo = new ();
   iPAddressInfo.PrivateIPv4 = GetPrivateIPv4();
   iPAddressInfo.PublicIPv4 = await GetPublicIPv4();
   iPAddressInfo.SubnetMask = GetSubnetMask();
   return iPAddressInfo;
}
string GetPrivateIPv4()
{
   var host = Dns.GetHostEntry(Dns.GetHostName());
   var ipAddress = host.AddressList.FirstOrDefault(ip => ip.AddressFamily == AddressFamily.InterNetwork);
   return ipAddress!.ToString();
}
string GetSubnetMask()
{
   var networkInterfaces = NetworkInterface.GetAllNetworkInterfaces();
   foreach (var networkInterface in networkInterfaces)
   {
      if (networkInterface.NetworkInterfaceType == NetworkInterfaceType.Wireless80211 ||
         networkInterface.NetworkInterfaceType == NetworkInterfaceType.Ethernet)
      {
         foreach (var unicastIPAddressInformation in networkInterface.GetIPProperties().UnicastAddresses)
         {
            if (unicastIPAddressInformation.Address.AddressFamily == AddressFamily.InterNetwork)
            {
               return unicastIPAddressInformation.IPv4Mask!.ToString();
            }
         }
      }
   }
   return null!;
}
async Task<string> GetPublicIPv4()
{
   using (HttpClient httpClient = new())
      return (await httpClient.GetStringAsync("http://icanhazip.com")).Trim();
}
public NetworkSecurityInfo GetConnectedNetworkSecurityInfo()
{
   NetworkSecurityInfo networkSecurityInfo = new();
   var client = new WlanClient();
   foreach (var wlanInterface in client.Interfaces)
   {
      var currentConnection = wlanInterface.CurrentConnection;
      WlanAvailableNetwork[] networks = wlanInterface.GetAvailableNetworkList(0);
      foreach (WlanAvailableNetwork network in networks)
      {
         if (network.profileName == currentConnection.profileName)
         {
            networkSecurityInfo.Authentication = network.dot11DefaultAuthAlgorithm;
            networkSecurityInfo.Encryption = network.dot11DefaultCipherAlgorithm;
            break;
         }
      }
   }
   return networkSecurityInfo;
}
public NetworkInfrastructureInfo GetConnectedNetworkInfrastructureInfo()
```

```
    {
        NetworkInfrastructureInfo networkInfrastructureInfo = new();
        var networkInterfaces = NetworkInterface.GetAllNetworkInterfaces();
        var currentInterface = networkInterfaces.FirstOrDefault(nic => nic.OperationalStatus == OperationalStatus.Up);
        if (currentInterface != null)
        {
            networkInfrastructureInfo.InterfaceType = currentInterface.NetworkInterfaceType;
            networkInfrastructureInfo.OperationalStatus = currentInterface.OperationalStatus;
            networkInfrastructureInfo.Interface = currentInterface.Description;
        }
        return networkInfrastructureInfo;
    }
}
```

- **INetworksService.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Services.Networks;
public interface INetworksService
{
    Task UpdateWiFiNetworksAsync();
    Task<IEnumerable<WiFiNetwork>> GetWiFiNetworksAsync();
    Task<IEnumerable<WiFiNetwork>> GetWiFiNetworksWithStatesAsync();
    Task AddWiFiNetworkAsync(WiFiNetwork network);
    Task UpdateWiFiNetworkAsync(WiFiNetwork network);
    Task DeleteWiFiNetworkByIdAsync(long networkId);
    Task DeleteWiFiNetworkByBSSIDAsync(string BSSID);
    Task<WiFiNetwork?> GetWiFiNetworkByIdAsync(long id);
    Task<WiFiNetwork?> GetWiFiNetworkByBSSIDAsync(string BSSID);
    NetworkStates? GetNetworkStates(WiFiNetwork network);
}
```

- **NetworksService.cs:**

```
using Microsoft.EntityFrameworkCore;
using NativeWifi;
using WiFi_Analyzer.Database;
using WiFi_Analyzer.Extensions;
using WiFi_Analyzer.Models;
using static NativeWifi.Wlan;
namespace WiFi_Analyzer.Services.Networks;
public class NetworksService : NetworkService, INetworksService
{
    readonly WiFiAnalyzerContext db;
    public NetworksService(WiFiAnalyzerContext db)
        => this.db = db;
    DbSet<WiFiNetwork> dbSet => db.WiFiNetworks;
    public async Task<IEnumerable<WiFiNetwork>> GetWiFiNetworksWithStatesAsync()
    {
        IEnumerable<WiFiNetwork> networks = await GetWiFiNetworksAsync();
        foreach (WiFiNetwork network in networks)
            network.NetworkStates = GetNetworkStates(network);
        return networks;
    }
    public async Task<IEnumerable<WiFiNetwork>> GetWiFiNetworksAsync()
        => await dbSet.ToListAsync();
    public async Task AddWiFiNetworkAsync(WiFiNetwork network)
    {
        var existingNetwork = await GetWiFiNetworkByIdAsync(network.Id);
        if (existingNetwork is null)
        {
            await dbSet.AddAsync(network);
            await SaveChangesAsync();
        }
```

```
}
public async Task UpdateWiFiNetworkAsync(WiFiNetwork network)
{
   WiFiNetwork? _network = await GetWiFiNetworkByBSSIDAsync(network.StringMacAddress)!;
   if (_network is not null)
   {
      CopyWiFiNetworkValues(network, _network);
      dbSet.Update(_network);
      await SaveChangesAsync();
   }
}
async Task DeleteWiFiNetworkAsync(WiFiNetwork? network)
{
   if (network is not null)
   {
      dbSet.Remove(network);
      await SaveChangesAsync();
   }
}
public async Task DeleteWiFiNetworkByIdAsync(long networkId)
{
   WiFiNetwork? network = await GetWiFiNetworkByIdAsync(networkId);
   await DeleteWiFiNetworkAsync(network);
}
public async Task DeleteWiFiNetworkByBSSIDAsync(string BSSID)
{
   WiFiNetwork? network = await GetWiFiNetworkByBSSIDAsync(BSSID);
   await DeleteWiFiNetworkAsync(network);
}
public async Task<WiFiNetwork?> GetWiFiNetworkByIdAsync(long id)
   => (await GetWiFiNetworksAsync()).SingleOrDefault(n => n.Id == id);
public async Task<WiFiNetwork?> GetWiFiNetworkByBSSIDAsync(string BSSID)
   => (await GetWiFiNetworksAsync()).SingleOrDefault(n => n.StringMacAddress == BSSID);
public async Task UpdateWiFiNetworksAsync()
{
   WlanClient client = new WlanClient();
   foreach (WlanClient.WlanInterface wlanInterface in client.Interfaces)
   {
      WlanBssEntry[] bssEntries = wlanInterface.GetNetworkBssList();
      foreach (var bssEntry in bssEntries)
      {
         string ssid = GetStringForSSID(bssEntry.dot11Ssid);
         long frequency = GetFrequencyFromChannel(bssEntry.chCenterFrequency);
         byte[] macAddress = bssEntry.dot11Bssid;
         int channel = GetChannelFromFrequency(frequency);
         string protocol = FindProtocolString(bssEntry);
         if (GetWlanAvailableNetworkByProfileName(ssid) is WlanAvailableNetwork wlanAvailableNetwork)
         {
            Dot11AuthAlgorithm authenticationAlgorithm = wlanAvailableNetwork.dot11DefaultAuthAlgorithm;
            bool isSecured = wlanAvailableNetwork.securityEnabled;
            WiFiNetwork network = new()
            {
               SSID = ssid,
               FrequencyInHz = frequency,
               Channel = channel,
               MacAddress = macAddress,
               IsSecured = isSecured,
               AuthenticationAlgorithm = authenticationAlgorithm,
               Protocol = protocol,
               LastSeen = DateTime.Now
```

```
            };
            string macAddressStr = macAddress.MacAddressToString();
            WiFiNetwork? _network = await GetWiFiNetworkByBSSIDAsync(macAddressStr);
            if (_network is null)
            {
               await AddWiFiNetworkAsync(network);
            }
            else
            {
               CopyWiFiNetworkValues(network, _network);
               await UpdateWiFiNetworkAsync(_network);
            }
         }
      }
   }
   public NetworkStates? GetNetworkStates(WiFiNetwork network)
   {
      WlanClient client = new WlanClient();
      foreach (WlanClient.WlanInterface wlanInterface in client.Interfaces)
      {
         WlanBssEntry[] bssEntries = wlanInterface.GetNetworkBssList();

         foreach (var bssEntry in bssEntries)
         {
            string macAddressStr = bssEntry.dot11Bssid.MacAddressToString();

            if (network.StringMacAddress == macAddressStr)
            {
               string ssid = GetStringForSSID(bssEntry.dot11Ssid);
               int signalStrength = bssEntry.rssi;
               long frequency = GetFrequencyFromChannel(bssEntry.chCenterFrequency);
               double distance = CalculateDistance(signalStrength, frequency);
               bool isConnected = false;
               if (wlanInterface.InterfaceState == WlanInterfaceState.Connected)
               {
                  string connectedSSID =
GetStringForSSID(wlanInterface.CurrentConnection.wlanAssociationAttributes.dot11Ssid);
                  if (ssid == connectedSSID)
                     isConnected = true;
               }
               return new NetworkStates()
               {
                  IsConnected = isConnected,
                  DistanceInMeters = distance,
                  SignalStrengthIndBm = signalStrength
               };
            }
         }
      }

      return null;
   }
   void CopyWiFiNetworkValues(WiFiNetwork fromNetwork, WiFiNetwork toNetwork)
   {
      toNetwork.SSID = fromNetwork.SSID;
      toNetwork.FrequencyInHz = fromNetwork.FrequencyInHz;
      toNetwork.Channel = fromNetwork.Channel;
      toNetwork.IsSecured = fromNetwork.IsSecured;
      toNetwork.AuthenticationAlgorithm = fromNetwork.AuthenticationAlgorithm;
```

```
        toNetwork.Protocol = fromNetwork.Protocol;
        toNetwork.LastSeen = fromNetwork.LastSeen;
    }
    async Task SaveChangesAsync()
        => await db.SaveChangesAsync();
}
```

- **ServiceProviders.cs:**

```
using Microsoft.Extensions.Configuration;
using WiFi_Analyzer.Services.ConnectedNetwork;
using WiFi_Analyzer.Services.Networks;
using WiFi_Analyzer.Services.SpeedTest;
using WiFi_Analyzer.ViewModels;
using Microsoft.EntityFrameworkCore;
using WiFi_Analyzer.Database;
namespace WiFi_Analyzer.Providers;
public static class ServiceProviders
{
    public static void AddMSSQLServer(this IServiceCollection services)
    {
        IConfiguration configuration = new ConfigurationBuilder()
            .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
            .Build();
        string connection = configuration.GetConnectionString("DefaultConnection")!;
        services.AddDbContext<WiFiAnalyzerContext>(opts =>
        {
            opts.UseSqlServer(connection);
            opts.EnableSensitiveDataLogging();
        });
    }
    public static void AddServices(this IServiceCollection services)
    {
        services.AddSingleton<IConnectedNetworkService, ConnectedNetworkService>();
        services.AddSingleton<ISpeedTestService, SpeedTestService>();
        services.AddSingleton<INetworksService, NetworksService>();
    }
    public static void AddViewModels(this IServiceCollection services)
    {
        services.AddSingleton<ConnectedNetworkViewModel>();
        services.AddSingleton<NetworksTableViewModel>();
        services.AddSingleton<NetworksGraphViewModel>();
        services.AddSingleton<MainPageViewModel>();
    }
}
```

- **WiFiAnalyzerContext.cs:**

```
using Microsoft.EntityFrameworkCore;
using WiFi_Analyzer.Models;

namespace WiFi_Analyzer.Database;
public class WiFiAnalyzerContext : DbContext
{
    public DbSet<WiFiNetwork> WiFiNetworks { get; set; } = null!;
    public WiFiAnalyzerContext(DbContextOptions<WiFiAnalyzerContext> options) : base(options)
        => Database.EnsureCreated();
}
```

- **DynamicComparer.cs:**

```
using System.Collections;
using WiFi_Analyzer.Enums;
namespace WiFi_Analyzer.Comparers;
public class DynamicComparer<T> : IComparer<T>
{
```

```
    readonly string propertyPath;
    readonly OrderBy orderBy;
    public DynamicComparer(string propertyPath, OrderBy orderBy)
        => (this.propertyPath, this.orderBy) = (propertyPath,  orderBy);
    public int Compare(T? x, T? y)
    {
        var xValue = GetPropertyValue(x, propertyPath);
        var yValue = GetPropertyValue(y, propertyPath);
        if (xValue is null && yValue is null)
            return 0; // equal
        else if (xValue is null)
            return 1; // y > x
        else if(yValue is null)
            return -1; // x > y

        int result = Comparer.Default.Compare(xValue, yValue);
        return orderBy == OrderBy.Ascending ? result : -result;
    }
        static object? GetPropertyValue(T? model, string propertyPath)
    {
        if (model is null)
            return null;
        object? propertyValue = model;
        string[] properties = propertyPath.Split('.');
        foreach (string property in properties)
        {
            if (propertyValue is null)
                return null;
            var propertyInfo = propertyValue!.GetType().GetProperty(property);
            if (propertyInfo is null)
                return null;
            propertyValue = propertyInfo.GetValue(propertyValue);
        }
        return propertyValue;
    }
}
```

- **OrderBy.cs:**

```
namespace WiFi_Analyzer.Enums;
public enum OrderBy
{
    Descending, Ascending
}
```

- **EnumerableExtensions.cs:**

```
using System.Linq.Dynamic.Core;
using System.Collections;
using WiFi_Analyzer.Enums;
using WiFi_Analyzer.Comparers;
namespace WiFi_Analyzer.Extensions;
public static class EnumerableExtensions
{
    public static IEnumerable<T> OrderBy<T>(this IEnumerable<T> models, string attribute, OrderBy orderBy)
        => models.AsQueryable().OrderBy($"{attribute} {orderBy}");
    public static IEnumerable<T> OrderBy<T>(this IEnumerable<T> models, string attribute, OrderBy orderBy,
IComparer comparer)
        => models.AsQueryable().OrderBy($"{attribute} {orderBy}", comparer);
    public static IEnumerable<T> DynamicOrderBy<T>(this IEnumerable<T> models, string attribute, OrderBy orderBy)
    {
        IComparer<T> comparer = new DynamicComparer<T>(attribute, orderBy);
        return models.OrderBy(m => m, comparer);
    }
```

```
}
```

- **StringExtensions.cs:**

```
using SpeedTest.Net.Enums;
using WiFi_Analyzer.Enums;
namespace WiFi_Analyzer.Extensions;
public static class StringExtensions
{
   public static SpeedTestUnit ParseToSpeedTestUnit(this string speedTestUnitStr)
     => speedTestUnitStr switch
     {
        "Bps" => SpeedTestUnit.BytesPerSecond,
        "KBps" or "KB/s" => SpeedTestUnit.KiloBytesPerSecond,
        "Kbps" or "Kb/s" => SpeedTestUnit.KiloBitsPerSecond,
        "MBps" or "MB/s" => SpeedTestUnit.MegaBytesPerSecond,
        "Mbps" or "Mb/s" => SpeedTestUnit.MegaBitsPerSecond,
        _ => throw new ArgumentException("Can't parse to SpeedTestUnit")
     };
   public static string ToShortString(this SpeedTestUnit speedTestUnit)
     => speedTestUnit switch
     {
        SpeedTestUnit.BytesPerSecond => "Bps",
        SpeedTestUnit.KiloBytesPerSecond => "KB/s",
        SpeedTestUnit.KiloBitsPerSecond => "Kb/s",
        SpeedTestUnit.MegaBytesPerSecond => "MB/s",
        SpeedTestUnit.MegaBitsPerSecond => "Mb/s",
        _ => speedTestUnit.ToString()
     };
   public static OrderBy ParseToOrderBy(this string orderByStr)
     => orderByStr?.ToLower() switch
     {
        "ascending" or "asc" => OrderBy.Ascending,
        "descending" or "desc" => OrderBy.Descending,
        _ => throw new ArgumentException("Can't parse to OrderBy")
     };
   public static bool TryParseToOrderBy(this string? orderByStr, out OrderBy? orderBy)
   {
      try
      {
         orderBy = ParseToOrderBy(orderByStr!);
         return true;
      }
      catch
      {
         orderBy = default;
         return false;
      }
   }
   public static string MacAddressToString(this byte[] macAddress)
      => string.Join(":", macAddress.Select(b => b.ToString("X2")));
}
```

- **ErrorHandler.cs:**

```
namespace WiFi_Analyzer.Helpers;
public static class ErrorHandler
{
   public static async Task DisplayErrorAsync(string errorMessage)
      => await Shell.Current.DisplayAlert("Error", errorMessage, "OK");
   public static async Task DisplayNetworkErrorAsync(string errorMessage)
      => await Shell.Current.DisplayAlert("Network Error", errorMessage, "Exit");
}
```

- **NetworksFilter.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Helpers;
public static class NetworksFilter
{
   const double GHz_5 = 5;
   const double GHz_6 = 6;
   public static IEnumerable<WiFiNetwork> FilterByGHz(IEnumerable<WiFiNetwork> networks, string parameter)
      => (parameter switch {
         "2.4 GHz" => networks.Where(n => n.FrequencyInGHz < GHz_5),
         "5 GHz" => networks.Where(n => n.FrequencyInGHz >= GHz_5 && n.FrequencyInGHz < GHz_6),
         "6 GHz" => networks.Where(n => n.FrequencyInGHz >= GHz_6),
         "All" or _ => networks,
      }).ToList();
}
```

- **ServiceHelper.cs:**

```
namespace WiFi_Analyzer.Helpers;
public static class ServiceHelper
{
   public static IServiceProvider Services { get; private set; } = null!;
   public static void Initialize(IServiceProvider serviceProvider) =>
      Services = serviceProvider;
   public static T? GetService<T>() => Services.GetService<T>();
}
```

- **SingalColorHelper.cs:**

```
namespace WiFi_Analyzer.Helpers;
public static class SingalColorHelper
{
   public static string GetHexColorBySingalStrength(int signalStrengthIndBm)
   {
      if (signalStrengthIndBm >= -30) // Strong signal (Green)
         return "#00FF00";
      else if (signalStrengthIndBm > -70) // Medium signal (Light Green)
         return "#90EE90";
      else if (signalStrengthIndBm > -80) // Weak signal (Yellow)
         return "#FFFF00";
      else // Very weak signal (Red)
         return "#FF0000";
   }
}
```

- **CheckBoxColorConverter.cs:**

```
using System.Globalization;
namespace WiFi_Analyzer.Converters;
public class CheckBoxColorConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
   {
      if (value is bool boolValue)
      {
         return boolValue ? Colors.Green : Colors.Red;
      }
      return value;
   }
   public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
   {
      throw new NotImplementedException();
   }
}
```

- **LastSeenConverter.cs:**

```
using System.Globalization;
```

```
namespace WiFi_Analyzer.Converters;
public class LastSeenConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
   {
      if (value is DateTime lastSeen)
      {
         TimeSpan difference = DateTime.Now.Subtract(lastSeen);
         bool isThisYear = difference.TotalDays <= 365;
         bool isThisMonth = difference.TotalDays <= 31;
         bool isThisWeek = difference.TotalDays <= 7;
         bool isThisDay = difference.TotalHours <= 24;
         bool isThisHour = difference.TotalMinutes <= 60;
         bool isRecently = difference.TotalMinutes <= 5;
         bool isNow = difference.TotalMinutes <= 1;
         string lastSeenStr;
         if (isNow)
            lastSeenStr = "now";
         else if (isRecently)
            lastSeenStr = "recently";
         else if (isThisHour)
            lastSeenStr = "this hour";
         else if (isThisDay)
            lastSeenStr = "today";
         else if (isThisWeek)
            lastSeenStr = "this week";
         else if (isThisMonth)
            lastSeenStr = "this month";
         else if (isThisYear)
            lastSeenStr = "this year";
         else
            lastSeenStr = "long ago";
         return lastSeenStr;
      }
      return value;
   }
   public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
   {
      throw new NotImplementedException();
   }
}
```

- **SignalStrengthColorConverter.cs:**

```
using System.Globalization;
using WiFi_Analyzer.Helpers;
namespace WiFi_Analyzer.Converters;
public class SignalStrengthColorConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
   {
      if (value is int signalStrength)
         return Color.FromHex(SingalColorHelper.GetHexColorBySingalStrength(signalStrength));
      return value;
   }
   public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
   {
      throw new NotImplementedException();
   }
}
```

- **SortDirectionToSymbolConverter.cs**

```
using System.Globalization;
```

```
using WiFi_Analyzer.Enums;
namespace WiFi_Analyzer.Converters;
public class SortDirectionToSymbolConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
   {
      if (value is OrderBy sortDirection)
         return sortDirection == OrderBy.Ascending ? "↑" : "↓";

      return null!;
   }
   public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
   {
      throw new NotSupportedException();
   }
}
```

- **SpeedTestUnitConverter.cs:**

```
using SpeedTest.Net.Enums;
using System.Globalization;
using WiFi_Analyzer.Extensions;
namespace WiFi_Analyzer.Converters;
public class SpeedTestUnitConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
      => value is SpeedTestUnit unit ? unit.ToShortString() : string.Empty;
   public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
      => value is string valueStr ? valueStr.ParseToSpeedTestUnit() : default;
}
```

- **MainPage.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         xmlns:controls="clr-namespace:WiFi_Analyzer.Controls"
         x:Class="WiFi_Analyzer.Pages.MainPage"
         ControlTemplate="{StaticResource CommonFrameTemplate}"
         Title="Home">
      <Grid Margin="20,10,0,0">
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
         </Grid.ColumnDefinitions>
         <StackLayout Grid.Column="0"
                  HorizontalOptions="CenterAndExpand"
                  VerticalOptions="CenterAndExpand">
            <controls:NetworkDetailsView
               ConnectedNetwork="{Binding ConnectedNetwork}"
               NetworkStates="{Binding NetworkStates}"/>
         </StackLayout>
         <StackLayout Grid.Column="1"
                  HorizontalOptions="CenterAndExpand"
                  VerticalOptions="CenterAndExpand" Spacing="20">
            <controls:SpeedTestView
               DownloadSpeed="{Binding DownloadSpeed}"
               IsBusy="{Binding IsBusy}"
               GetDownloadSpeedCommand="{Binding GetDownloadSpeedCommand}"/>
         </StackLayout>
      </Grid>
</ContentPage>
```

- **MainPage.xaml.cs:**

```
using WiFi_Analyzer.Helpers;
```

```
using WiFi_Analyzer.ViewModels;
namespace WiFi_Analyzer.Pages;
public partial class MainPage : ContentPage
{
   readonly MainPageViewModel mainPageViewModel = null!;
   public MainPage()
   {
      mainPageViewModel = ServiceHelper.GetService<MainPageViewModel>()!;
      InitializeComponent();
      BindingContext = mainPageViewModel;
   }
   protected override async void OnAppearing()
   {
      base.OnAppearing();
      await mainPageViewModel.LoadDataAsync();
   }
   protected override void OnDisappearing()
   {
      base.OnDisappearing();
      mainPageViewModel.Dispose();
   }
}
```

- **ConnectedNetworkPage.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         x:Class="WiFi_Analyzer.Pages.ConnectedNetworkPage"
         xmlns:controls="clr-namespace:WiFi_Analyzer.Controls"
         ControlTemplate="{StaticResource CommonFrameTemplate}"
         Title="Connected">
   <ScrollView>
      <StackLayout Margin="20,10,0,0">
         <!-- Table NetworkDetails -->
         <controls:NetworkDetailsView
            ConnectedNetwork="{Binding ConnectedNetwork}"
            NetworkStates="{Binding NetworkStates}"/>
         <!-- Table IPAddressInfo -->
         <controls:IPAddressInfoView
            IPAddressInfo="{Binding IPAddressInfo}" />
         <!-- Table NetworkSecurityInfo -->
         <controls:NetworkSecurityInfoView
            NetworkSecurityInfo="{Binding NetworkSecurityInfo}" />
         <!-- Table NetworkInfrastructureInfo -->
         <controls:NetworkInfrastructureInfoView
            NetworkInfrastructureInfo="{Binding NetworkInfrastructureInfo}"/>
      </StackLayout>
   </ScrollView>
</ContentPage>
```

- **ConnectedNetworkPage.xaml.cs:**

```
using WiFi_Analyzer.Helpers;
using WiFi_Analyzer.ViewModels;
namespace WiFi_Analyzer.Pages;
public partial class ConnectedNetworkPage : ContentPage
{
   readonly ConnectedNetworkViewModel connectedNetworkViewModel = null!;
   public ConnectedNetworkPage()
   {
      connectedNetworkViewModel = ServiceHelper.GetService<ConnectedNetworkViewModel>()!;
      InitializeComponent();
```

```
        BindingContext = connectedNetworkViewModel;
    }
    protected override async void OnAppearing()
    {
        base.OnAppearing();
        await connectedNetworkViewModel.LoadDataAsync();
    }
    protected override void OnDisappearing()
    {
        base.OnDisappearing();
        connectedNetworkViewModel.Dispose();
    }
}
```

- **NetworksTablePage.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:converters="clr-namespace:WiFi_Analyzer.Converters"
             x:Class="WiFi_Analyzer.Pages.Networks.NetworksTablePage"
             xmlns:controls="clr-namespace:WiFi_Analyzer.Controls"
             ControlTemplate="{StaticResource CommonFrameTemplate}"
             Title="Networks">
    <ContentPage.Resources>
        <ResourceDictionary>
            <converters:SignalStrengthColorConverter x:Key="SignalStrengthColorConverter" />
            <converters:CheckBoxColorConverter x:Key="CheckBoxColorConverter" />
            <converters:LastSeenConverter x:Key="LastSeenConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>
    <ScrollView>
        <CollectionView ItemsSource="{Binding FilteredWiFiNetworks}">
            <CollectionView.Header>
                <StackLayout Orientation="Vertical">
                    <Button Text="View Graph" Clicked="NavigateToGraphPage"  WidthRequest="150"
HorizontalOptions="StartAndExpand" Margin="10"/>
                            <controls:NetworksFilterByGHzView FilterByGHzCommand="{Binding FilterByGHzCommand}"/>
                <controls:TitleLabelView Text="{Binding FilteredWiFiNetworks.Count, StringFormat='Total Networks: {0}'}"
Grid.Row="0" Grid.ColumnSpan="2" Margin="0,5,0,5"/>
                            <Grid Margin="0,10,0,0" IsVisible="{Binding HasFilteredNetworks}">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <controls:SortableHeaderView Grid.Column="0"
                            HeaderText="SSID"
                            SortDirection="{Binding Path=SortDirections[SSID]}"
                            SortCommand="{Binding SortCommand}"
                            SortCommandParameter="SSID" />
                    <controls:SortableHeaderView Grid.Column="1"
                            HeaderText="Protocol*"
```

```
                SortDirection="{Binding Path=SortDirections[Protocol]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="Protocol" />
        <controls:SortableHeaderView Grid.Column="2"
                HeaderText="Frequency"
                SortDirection="{Binding Path=SortDirections[FrequencyInHz]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="FrequencyInHz" />
        <controls:SortableHeaderView Grid.Column="3"
                HeaderText="Connected"
                SortDirection="{Binding Path=SortDirections[NetworkStates.IsConnected]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="NetworkStates.IsConnected" />
        <controls:SortableHeaderView Grid.Column="4"
                HeaderText="Channel"
                SortDirection="{Binding Path=SortDirections[Channel]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="Channel" />
        <controls:SortableHeaderView Grid.Column="5" Grid.ColumnSpan="2"
                HeaderText="Signal Strength"
                SortDirection="{Binding Path=SortDirections[NetworkStates.SignalStrengthIndBm]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="NetworkStates.SignalStrengthIndBm" />
        <controls:SortableHeaderView Grid.Column="7"
                HeaderText="BSSID"
                SortDirection="{Binding Path=SortDirections[StringMacAddress]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="StringMacAddress" />
        <controls:SortableHeaderView Grid.Column="8"
                HeaderText="Distance"
                SortDirection="{Binding Path=SortDirections[NetworkStates.DistanceInMeters]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="NetworkStates.DistanceInMeters" />
        <controls:SortableHeaderView Grid.Column="9"
                HeaderText="Secured"
                SortDirection="{Binding Path=SortDirections[IsSecured]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="IsSecured" />
        <controls:SortableHeaderView Grid.Column="10"
                HeaderText="Authentication"
                SortDirection="{Binding Path=SortDirections[AuthenticationAlgorithm]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="AuthenticationAlgorithm" />
        <controls:SortableHeaderView Grid.Column="11"
                HeaderText="Last Seen"
                SortDirection="{Binding Path=SortDirections[LastSeen]}"
                SortCommand="{Binding SortCommand}"
                SortCommandParameter="LastSeen" />
      </Grid>
    </StackLayout>
  </CollectionView.Header>
  <CollectionView.ItemTemplate>
    <DataTemplate>
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="*" />
```

```
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <!--The first row-->
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="0"
Grid.RowSpan="2">
                <controls:RowLabelView Text="{Binding SSID}" />
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="1"
Grid.RowSpan="2">
                <controls:RowLabelView Text="{Binding Protocol}" />
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="2"
Grid.RowSpan="2">
                <controls:RowLabelView Text="{Binding FrequencyInMHz, StringFormat='{0} MHz'}" />
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="3"
Grid.RowSpan="2">
                <CheckBox IsChecked="{Binding NetworkStates.IsConnected}" IsEnabled="False"
                    Color="{Binding NetworkStates.IsConnected, Converter={StaticResource
CheckBoxColorConverter}}"
                    HorizontalOptions="Center" VerticalOptions="Center" Margin="5"/>
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="4"
Grid.RowSpan="2">
                <controls:RowLabelView Text="{Binding Channel}" />
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="5">
                <controls:RowLabelView
                    Text="{Binding NetworkStates.SignalStrengthIndBm, StringFormat='{0} dBm'}"
                    TextColor="{Binding NetworkStates.SignalStrengthIndBm, Converter={StaticResource
SignalStrengthColorConverter}}" />
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="6">
                <controls:RowLabelView
                    Text="{Binding NetworkStates.SignalStrengthInPercentage, StringFormat='{0}%'}"
                    TextColor="{Binding NetworkStates.SignalStrengthIndBm, Converter={StaticResource
SignalStrengthColorConverter}}" />
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="7"
Grid.RowSpan="2">
                <controls:RowLabelView Text="{Binding StringMacAddress}" />
            </Frame>
            <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="8"
Grid.RowSpan="2">
                <controls:RowLabelView
                    Text="{Binding NetworkStates.DistanceInMeters, StringFormat='{0:F1} m'}"
                    TextColor="{Binding NetworkStates.SignalStrengthIndBm, Converter={StaticResource
SignalStrengthColorConverter}}" />
            </Frame>
```

```
                <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="9"
Grid.RowSpan="2">
                  <CheckBox IsChecked="{Binding IsSecured}" IsEnabled="False"
                      Color="{Binding IsSecured, Converter={StaticResource CheckBoxColorConverter}}"
                      HorizontalOptions="Center" VerticalOptions="Center" Margin="5"/>
                </Frame>
                <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="10"
Grid.RowSpan="2">
                  <controls:RowLabelView Text="{Binding AuthenticationAlgorithm}" />
                </Frame>
                <Frame BorderColor="White" Padding="0" CornerRadius="0" Grid.Row="0" Grid.Column="11"
Grid.RowSpan="2">
                  <controls:RowLabelView Text="{Binding LastSeen, Converter={StaticResource LastSeenConverter}}" />
                </Frame>
                <!--The second row-->
                <Frame Grid.Row="1" Grid.Column="5" Grid.ColumnSpan="2">
                  <controls:SignalStrengthSlider NetworkStates="{Binding NetworkStates}" />
                </Frame>
              </Grid>
            </DataTemplate>
          </CollectionView.ItemTemplate>
        </CollectionView>
      </ScrollView>
    </ContentPage>
```

- **NetworksTablePage.xaml.cs:**

```
using WiFi_Analyzer.Helpers;
using WiFi_Analyzer.ViewModels;
namespace WiFi_Analyzer.Pages.Networks;
public partial class NetworksTablePage : ContentPage
{
  readonly NetworksTableViewModel networksTableViewModel = null!;

  public NetworksTablePage()
  {
    networksTableViewModel = ServiceHelper.GetService<NetworksTableViewModel>()!;
    InitializeComponent();
    BindingContext = networksTableViewModel;
  }
  protected override void OnAppearing()
  {
    base.OnAppearing();
    UpdateNetworks();
  }
  async void UpdateNetworks()
    => await networksTableViewModel.LoadDataAsync();
  async void NavigateToGraphPage(object sender, EventArgs e)
    => await Navigation.PushAsync(new NetworksGraphPage());
  protected override void OnDisappearing()
  {
    base.OnDisappearing();
    networksTableViewModel.Dispose();
  }
}
```

- **NetworksGraphPage.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:micro="clr-namespace:Microcharts.Maui;assembly=Microcharts.Maui"
        x:Class="WiFi_Analyzer.Pages.Networks.NetworksGraphPage"
        xmlns:controls="clr-namespace:WiFi_Analyzer.Controls"
```

```
        ControlTemplate="{StaticResource CommonFrameTemplate}"
        Title="Networks Graph">
  <ScrollView>
    <StackLayout Background="Black" Margin="10">
      <Button Text="View Table" Clicked="NavigateToTablePage"
           WidthRequest="150" HorizontalOptions="StartAndExpand" />

      <controls:NetworksFilterByGHzView FilterByGHzCommand="{Binding FilterByGHzCommand}" />
      <StackLayout IsVisible="{Binding HasFilteredNetworks}">
        <Label Text="dBm" FontSize="24" FontAttributes="Bold"
             HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
             Margin="10,0,10,10" />
        <micro:ChartView HeightRequest="500" Margin="10" Chart="{Binding DBmChart}" />
        <Label Text="Distance" FontSize="24" FontAttributes="Bold"
             HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
             Margin="10,0,10,10" />
        <micro:ChartView HeightRequest="500" Margin="10" Chart="{Binding DistanceChart}"/>
      </StackLayout>
      <Label IsVisible="{Binding HasNoFilteredNetworks}" Text="No network found"
          TextColor="Red" Margin="0, 20, 0, 0"
          HorizontalOptions="Center" VerticalOptions="Center"  FontAttributes="Bold" FontSize="25"/>
    </StackLayout>
  </ScrollView>
</ContentPage>
```

- **NetworksGraphPage.xaml.cs:**

```
using WiFi_Analyzer.Helpers;
using WiFi_Analyzer.ViewModels;
namespace WiFi_Analyzer.Pages.Networks;
public partial class NetworksGraphPage : ContentPage
{
  readonly NetworksGraphViewModel networksGraphViewModel = null!;
  public NetworksGraphPage()
  {
    networksGraphViewModel = ServiceHelper.GetService<NetworksGraphViewModel>()!;
    InitializeComponent();
    BindingContext = networksGraphViewModel;
  }
  protected override void OnAppearing()
  {
    base.OnAppearing();
    UpdateNetworks();
  }
  async void UpdateNetworks()
     => await networksGraphViewModel.LoadDataAsync();
  async void NavigateToTablePage(object sender, EventArgs e)
    => await Navigation.PushAsync(new NetworksTablePage());
  protected override void OnDisappearing()
  {
    base.OnDisappearing();
    networksGraphViewModel.Dispose();
  }
}
```

- **BodyLabelView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         x:Name="this"
         x:Class="WiFi_Analyzer.Controls.BodyLabelView">
  <Label Text="{Binding Text}"
       TextColor="{Binding TextColor}"
```

```
        BackgroundColor="#252526"
        HorizontalTextAlignment="Center"
        VerticalTextAlignment="Center"
        Padding="10,0"
        BindingContext="{x:Reference this}"/>
</ContentView>
```

- **BodyLabelView.xaml.cs:**

```
namespace WiFi_Analyzer.Controls;
public partial class BodyLabelView : ContentView
{
   public static readonly BindableProperty TextProperty = BindableProperty.Create(
      nameof(Text), typeof(string), typeof(BodyLabelView), string.Empty);
   public static readonly BindableProperty TextColorProperty = BindableProperty.Create(
      nameof(TextColor), typeof(Color), typeof(BodyLabelView), Color.FromHex("#FFFFFF"));
   public string Text
   {
      get => (string)GetValue(TextProperty);
      set => SetValue(TextProperty, value);
   }
   public Color TextColor
   {
      get => (Color)GetValue(TextColorProperty);
      set => SetValue(TextColorProperty, value);
   }
   public BodyLabelView()
      => InitializeComponent();
}
```

- **HeaderLabelView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          x:Name="this"
          x:Class="WiFi_Analyzer.Controls.HeaderLabelView">
   <Label Text="{Binding Text}"
        TextColor="White"
        BackgroundColor="#252526"
        HorizontalTextAlignment="Center"
        VerticalTextAlignment="Center"
        FontAttributes="Bold"
        Padding="10,0"
        BindingContext="{x:Reference this}"/>
</ContentView>
```

- **HeaderLabelView.xaml.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Controls;
public partial class HeaderLabelView : ContentView
{
   public static readonly BindableProperty TextProperty = BindableProperty.Create(
      nameof(Text), typeof(string), typeof(HeaderLabelView), string.Empty);
   public string Text
   {
      get => (string)GetValue(TextProperty);
      set => SetValue(TextProperty, value);
   }
   public HeaderLabelView()
      => InitializeComponent();
}
```

- **RowLabelView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Name="this"
             x:Class="WiFi_Analyzer.Controls.RowLabelView">
   <Label Text="{Binding Text}"
          TextColor="{Binding TextColor}"
          HorizontalTextAlignment="Center"
          VerticalTextAlignment="Center"
          Margin="5"
          BindingContext="{x:Reference this}"/>
</ContentView>
```

- **RowLabelView.xaml.cs:**

```
namespace WiFi_Analyzer.Controls;
public partial class RowLabelView : ContentView
{
   public static readonly BindableProperty TextProperty = BindableProperty.Create(
         nameof(Text), typeof(string), typeof(RowLabelView), string.Empty);
   public static readonly BindableProperty TextColorProperty = BindableProperty.Create(
      nameof(TextColor), typeof(Color), typeof(RowLabelView), Color.FromHex("#FFFFFF"));
   public string Text
   {
      get => (string)GetValue(TextProperty);
      set => SetValue(TextProperty, value);
   }
   public Color TextColor
   {
      get => (Color)GetValue(TextColorProperty);
      set => SetValue(TextColorProperty, value);
   }
   public RowLabelView()
      => InitializeComponent();
}
```

- **TitleLabelView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Name="this"
             x:Class="WiFi_Analyzer.Controls.TitleLabelView">
   <Label Text="{Binding Text}"
          FontSize="17"
          TextColor="White"
          BackgroundColor="#2D2D30"
          HorizontalTextAlignment="Center"
          VerticalTextAlignment="Center"
          FontAttributes="Bold"
          BindingContext="{x:Reference this}"/>
</ContentView>
```

- **TitleLabelView.xaml.cs:**

```
namespace WiFi_Analyzer.Controls;

public partial class TitleLabelView : ContentView
{
   public static readonly BindableProperty TextProperty = BindableProperty.Create(
         nameof(Text), typeof(string), typeof(TitleLabelView), string.Empty);
   public string Text
   {
      get => (string)GetValue(TextProperty);
      set => SetValue(TextProperty, value);
   }
   public TitleLabelView()
```

```
    => InitializeComponent();
}
```

- **IPAddressInfoView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Name="this"
        xmlns:local="clr-namespace:WiFi_Analyzer.Controls"
        x:Class="WiFi_Analyzer.Controls.IPAddressInfoView">
  <StackLayout BindingContext="{x:Reference this}">
    <Frame WidthRequest="400" BackgroundColor="#1E1E1E" CornerRadius="10" Padding="0"
      HorizontalOptions="Start" Margin="0,0,0,20">
      <Grid RowSpacing="1">
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto" />
          <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <local:TitleLabelView Text="IP Address Details" Grid.Row="0" Grid.ColumnSpan="2"/>
        <local:GrayBoxView Grid.Row="1" Grid.ColumnSpan="2" />
        <local:HeaderLabelView Text="Private IPv4" Grid.Row="2" Grid.Column="0"/>
        <local:BodyLabelView Text="{Binding IPAddressInfo.PrivateIPv4}" Grid.Row="2" Grid.Column="1"/>
        <local:GrayBoxView Grid.Row="3" Grid.ColumnSpan="2" />
        <local:HeaderLabelView Text="Subnet Mask" Grid.Row="4" Grid.Column="0"/>
        <local:BodyLabelView Text="{Binding IPAddressInfo.SubnetMask}" Grid.Row="4" Grid.Column="1"/>
        <local:GrayBoxView Grid.Row="5" Grid.ColumnSpan="2" />
        <local:HeaderLabelView Text="Public IPv4" Grid.Row="6" Grid.Column="0"/>
        <local:BodyLabelView Text="{Binding IPAddressInfo.PublicIPv4}" Grid.Row="6" Grid.Column="1"/>
      </Grid>
    </Frame>
  </StackLayout>
</ContentView>
```

- **IPAddressInfoView.xaml.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Controls;
public partial class IPAddressInfoView : ContentView
{
  public static readonly BindableProperty IPAddressInfoProperty = BindableProperty.Create(
    nameof(IPAddressInfo), typeof(IPAddressInfo), typeof(IPAddressInfoView), null);
  public IPAddressInfoView()
            => InitializeComponent();
  public IPAddressInfo? IPAddressInfo
  {
    get => (IPAddressInfo?)GetValue(IPAddressInfoProperty);
    set => SetValue(IPAddressInfoProperty, value);
  }
}
```

- **NetworkDetailsView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         x:Name="this"
         xmlns:converters="clr-namespace:WiFi_Analyzer.Converters"
         xmlns:local="clr-namespace:WiFi_Analyzer.Controls"
         x:Class="WiFi_Analyzer.Controls.NetworkDetailsView">
    <ContentView.Resources>
        <ResourceDictionary>
            <converters:SignalStrengthColorConverter x:Key="SignalStrengthColorConverter" />
            <converters:CheckBoxColorConverter x:Key="CheckBoxColorConverter" />
        </ResourceDictionary>
    </ContentView.Resources>
    <StackLayout BindingContext="{x:Reference this}">
        <Frame WidthRequest="400" BackgroundColor="#1E1E1E" CornerRadius="10" Padding="0"
       HorizontalOptions="Start" Margin="0,0,0,20">
            <Grid RowSpacing="1">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="*" />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <local:TitleLabelView Text="Network Details" Grid.Row="0" Grid.ColumnSpan="2"/>
                <local:GrayBoxView Grid.Row="1" Grid.ColumnSpan="2" />
                <local:HeaderLabelView Text="SSID" Grid.Row="2" Grid.Column="0"/>
                <local:BodyLabelView Text="{Binding ConnectedNetwork.SSID}" Grid.Row="2" Grid.Column="1"/>
                <local:GrayBoxView Grid.Row="3" Grid.ColumnSpan="2" />
                <local:HeaderLabelView Text="Channel" Grid.Row="4" Grid.Column="0"/>
                <local:BodyLabelView Text="{Binding ConnectedNetwork.Channel}" Grid.Row="4" Grid.Column="1"/>
                <local:GrayBoxView Grid.Row="5" Grid.ColumnSpan="2" />
                <local:HeaderLabelView Text="Frequency" Grid.Row="6" Grid.Column="0"/>
                <local:BodyLabelView Text="{Binding ConnectedNetwork.FrequencyInMHz, StringFormat='{0} MHz'}"
Grid.Row="6" Grid.Column="1"/>
                <local:GrayBoxView Grid.Row="7" Grid.ColumnSpan="2" />
                <local:HeaderLabelView Text="Signal Strength" Grid.Row="8" Grid.Column="0" Grid.RowSpan="3"/>
                <local:BodyLabelView Text="{Binding NetworkStates.SignalStrengthIndBm, StringFormat='{0} dBm'}"
                    TextColor="{Binding NetworkStates.SignalStrengthIndBm, Converter={StaticResource
SignalStrengthColorConverter}}"
                    Grid.Row="8" Grid.Column="1"/>
```

```
            <Grid Grid.Row="9" Grid.Column="1" BackgroundColor="#252526">
              <local:SignalStrengthSlider NetworkStates="{Binding NetworkStates}" />
            </Grid>
            <local:BodyLabelView Text="{Binding NetworkStates.SignalStrengthInPercentage, StringFormat='{0}%'}"
                TextColor="{Binding NetworkStates.SignalStrengthIndBm, Converter={StaticResource
SignalStrengthColorConverter}}"
                Grid.Row="10" Grid.Column="1"/>
                    <local:GrayBoxView Grid.Row="11" Grid.ColumnSpan="2" />
            <local:HeaderLabelView Text="MAC Address" Grid.Row="12" Grid.Column="0"/>
            <local:BodyLabelView Text="{Binding ConnectedNetwork.StringMacAddress}" Grid.Row="12"
Grid.Column="1"/>
            <local:GrayBoxView Grid.Row="13" Grid.ColumnSpan="2" />
            <local:HeaderLabelView Text="Distance" Grid.Row="14" Grid.Column="0"/>
            <local:BodyLabelView Text="{Binding NetworkStates.DistanceInMeters, StringFormat='{0:F1} m'}"
                  TextColor="{Binding NetworkStates.SignalStrengthIndBm, Converter={StaticResource
SignalStrengthColorConverter}}"
                  Grid.Row="14" Grid.Column="1"/>
            <local:GrayBoxView Grid.Row="15" Grid.ColumnSpan="2" />
            <local:HeaderLabelView Text="Secured" Grid.Row="16" Grid.Column="0"/>
            <Grid Grid.Row="16" Grid.Column="1" BackgroundColor="#252526">
              <CheckBox IsChecked="{Binding ConnectedNetwork.IsSecured}"
               Color="{Binding ConnectedNetwork.IsSecured, Converter={StaticResource CheckBoxColorConverter}}"
               IsEnabled="False" HorizontalOptions="Center" VerticalOptions="Center"/>
            </Grid>
            <local:GrayBoxView Grid.Row="17" Grid.ColumnSpan="2" />
            <local:HeaderLabelView Text="Protocol" Grid.Row="18" Grid.Column="0"/>
            <local:BodyLabelView Text="{Binding ConnectedNetwork.Protocol}" Grid.Row="18" Grid.Column="1"/>
            <local:GrayBoxView Grid.Row="19" Grid.ColumnSpan="2" />
            <local:HeaderLabelView Text="Authentication" Grid.Row="20" Grid.Column="0"/>
            <local:BodyLabelView Text="{Binding ConnectedNetwork.AuthenticationAlgorithm}" Grid.Row="20"
Grid.Column="1"/>
          </Grid>
        </Frame>
      </StackLayout>
</ContentView>
```

- **NetworkDetailsView.xaml.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Controls;
public partial class NetworkDetailsView : ContentView
{
  public static readonly BindableProperty ConnectedNetworkProperty = BindableProperty.Create(
    nameof(ConnectedNetwork), typeof(WiFiNetwork), typeof(NetworkDetailsView));
  public static readonly BindableProperty NetworkStatesProperty = BindableProperty.Create(
    nameof(NetworkStates), typeof(NetworkStates), typeof(NetworkDetailsView));
  public WiFiNetwork ConnectedNetwork
  {
    get => (WiFiNetwork)GetValue(ConnectedNetworkProperty);
    set => SetValue(ConnectedNetworkProperty, value);
  }
  public NetworkStates NetworkStates
  {
    get => (NetworkStates)GetValue(NetworkStatesProperty);
    set => SetValue(NetworkStatesProperty, value);
  }
  public NetworkDetailsView()
    => InitializeComponent();
}
```

- **NetworkInfrastructureInfoView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Name="this"
        xmlns:local="clr-namespace:WiFi_Analyzer.Controls"
        x:Class="WiFi_Analyzer.Controls.NetworkInfrastructureInfoView">
  <StackLayout BindingContext="{x:Reference this}">
    <Frame WidthRequest="400" BackgroundColor="#1E1E1E" CornerRadius="10" Padding="0"
       HorizontalOptions="Start" Margin="0,0,0,20">
      <Grid RowSpacing="1">
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto" />
          <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <local:TitleLabelView Text="Infrastructure" Grid.Row="0" Grid.ColumnSpan="2"/>
        <local:GrayBoxView Grid.Row="1" Grid.ColumnSpan="2" />
        <local:HeaderLabelView Text="Interface" Grid.Row="2" Grid.Column="0"/>
        <local:BodyLabelView Text="{Binding NetworkInfrastructureInfo.Interface}" Grid.Row="2"
Grid.Column="1"/>

        <local:GrayBoxView Grid.Row="3" Grid.ColumnSpan="2" />
        <local:HeaderLabelView Text="Type" Grid.Row="4" Grid.Column="0"/>
        <local:BodyLabelView Text="{Binding NetworkInfrastructureInfo.InterfaceType}" Grid.Row="4"
Grid.Column="1"/>
      </Grid>
    </Frame>
  </StackLayout>
</ContentView>
```

- **NetworkInfrastructureInfoView.xaml.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Controls;
public partial class NetworkInfrastructureInfoView : ContentView
{
  public static readonly BindableProperty NetworkInfrastructureInfoProperty = BindableProperty.Create(
    nameof(NetworkInfrastructureInfo), typeof(NetworkInfrastructureInfo), typeof(NetworkInfrastructureInfoView));
  public NetworkInfrastructureInfo NetworkInfrastructureInfo
  {
    get => (NetworkInfrastructureInfo)GetValue(NetworkInfrastructureInfoProperty);
    set => SetValue(NetworkInfrastructureInfoProperty, value);
  }
  public NetworkInfrastructureInfoView()
          => InitializeComponent();
}
```

- **NetworkSecurityInfoView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Name="this"
        xmlns:local="clr-namespace:WiFi_Analyzer.Controls"
        x:Class="WiFi_Analyzer.Controls.NetworkSecurityInfoView">
  <StackLayout  BindingContext="{x:Reference this}">
    <Frame WidthRequest="400" BackgroundColor="#1E1E1E" CornerRadius="10" Padding="0"
       HorizontalOptions="Start" Margin="0,0,0,20">
      <Grid RowSpacing="1">
        <Grid.RowDefinitions>
```

```
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
          </Grid.ColumnDefinitions>
          <local:TitleLabelView Text="Security" Grid.Row="0" Grid.ColumnSpan="2"/>
          <local:GrayBoxView Grid.Row="1" Grid.ColumnSpan="2" />
          <local:HeaderLabelView Text="Authentication" Grid.Row="2" Grid.Column="0"/>
          <local:BodyLabelView Text="{Binding NetworkSecurityInfo.Authentication, StringFormat='{0} (WPA2)'}"
Grid.Row="2" Grid.Column="1"/>
          <local:GrayBoxView Grid.Row="3" Grid.ColumnSpan="2" />
          <local:HeaderLabelView Text="Encryption" Grid.Row="4" Grid.Column="0"/>
          <local:BodyLabelView Text="{Binding NetworkSecurityInfo.Encryption}" Grid.Row="4" Grid.Column="1"/>
        </Grid>
      </Frame>
    </StackLayout>
</ContentView>
```

- **NetworkSecurityInfoView.xaml.cs:**

```
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Controls;
public partial class NetworkSecurityInfoView : ContentView
{
    public static readonly BindableProperty NetworkSecurityInfoProperty = BindableProperty.Create(
        nameof(NetworkSecurityInfo), typeof(NetworkSecurityInfo), typeof(NetworkSecurityInfoView));
    public NetworkSecurityInfo NetworkSecurityInfo
    {
        get => (NetworkSecurityInfo)GetValue(NetworkSecurityInfoProperty);
        set => SetValue(NetworkSecurityInfoProperty, value);
    }
    public NetworkSecurityInfoView()
                    => InitializeComponent();
}
```

- **SpeedTestView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          xmlns:converters="clr-namespace:WiFi_Analyzer.Converters"
          x:Name="this"
          x:Class="WiFi_Analyzer.Controls.SpeedTestView">
    <ContentView.Resources>
        <ResourceDictionary>
            <converters:SpeedTestUnitConverter x:Key="SpeedTestUnitConverter"/>
        </ResourceDictionary>
    </ContentView.Resources>
    <StackLayout BindingContext="{x:Reference this}">
        <Label HorizontalOptions="Center" VerticalOptions="Center" TextColor="White" FontSize="20"
FontAttributes="Bold">
            <Label.FormattedText>
                <FormattedString>
                    <Span Text="{Binding DownloadSpeed.Speed, StringFormat='Speed: {0:F2}'}" />
                    <Span Text=" " />
                    <Span Text="{Binding DownloadSpeed.Unit, Converter={StaticResource SpeedTestUnitConverter}}" />
                </FormattedString>
            </Label.FormattedText>
        </Label>
```

```
    <Button Text="Speed" Command="{Binding GetDownloadSpeedCommand}" WidthRequest="150"
HeightRequest="150" CornerRadius="80" HorizontalOptions="Center" VerticalOptions="Center" FontSize="Large"/>
    <ActivityIndicator IsRunning="{Binding IsBusy}"
            IsVisible="{Binding IsBusy}"
            HorizontalOptions="Center"
            VerticalOptions="Center"
            Color="Blue"
            WidthRequest="50"
            HeightRequest="50"/>
  </StackLayout>
</ContentView>
```

- **SpeedTestView.xaml.cs:**

```
using System.Windows.Input;
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Controls;
public partial class SpeedTestView : ContentView
{
  public static readonly BindableProperty DownloadSpeedProperty = BindableProperty.Create(
    nameof(DownloadSpeed), typeof(DownloadSpeed), typeof(SpeedTestView));
  public static readonly BindableProperty IsBusyProperty = BindableProperty.Create(
    nameof(IsBusy), typeof(bool), typeof(SpeedTestView), false);
  public static readonly BindableProperty GetDownloadSpeedCommandProperty = BindableProperty.Create(
    nameof(GetDownloadSpeedCommand), typeof(ICommand), typeof(SpeedTestView));
  public DownloadSpeed DownloadSpeed
  {
    get => (DownloadSpeed)GetValue(DownloadSpeedProperty);
    set => SetValue(DownloadSpeedProperty, value);
  }
  public bool IsBusy
  {
    get => (bool)GetValue(IsBusyProperty);
    set => SetValue(IsBusyProperty, value);
  }
  public ICommand GetDownloadSpeedCommand
  {
    get => (ICommand)GetValue(GetDownloadSpeedCommandProperty);
    set => SetValue(GetDownloadSpeedCommandProperty, value);
  }
  public SpeedTestView()
            => InitializeComponent();
}
```

- **GrayBoxView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
      x:Class="WiFi_Analyzer.Controls.GrayBoxView">
  <BoxView BackgroundColor="#3D3D3D" HeightRequest="1" />
</ContentView>
```

- **GrayBoxView.xaml.cs:**

```
namespace WiFi_Analyzer.Controls;
public partial class GrayBoxView : ContentView
{
      public GrayBoxView()
      {
            InitializeComponent();
      }
}
```

- **NetworksFilterByGHzView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Name="this"
        x:Class="WiFi_Analyzer.Controls.NetworksFilterByGHzView">
  <StackLayout Orientation="Horizontal" HorizontalOptions="Center" Margin="0,20,0,20"
BindingContext="{x:Reference this}">
    <Button Text="All"
      BackgroundColor="Gray"
      x:Name="allGHzButton"
      Command="{Binding FilterByGHzCommand}"
      CommandParameter="All"
      WidthRequest="100"
      Clicked="OnGHzButtonClicked"/>
    <Button Text="2.4 GHz"
      x:Name="twoDotFourGHzButton"
      Command="{Binding FilterByGHzCommand}"
      CommandParameter="2.4 GHz"
      WidthRequest="100"
      Clicked="OnGHzButtonClicked"/>
    <Button Text="5 GHz"
      x:Name="fiveGHzButton"
      Command="{Binding FilterByGHzCommand}"
      CommandParameter="5 GHz"
      WidthRequest="100"
      Clicked="OnGHzButtonClicked"/>
    <Button Text="6 GHz"
      x:Name="sixGHzButton"
      Command="{Binding FilterByGHzCommand}"
      CommandParameter="6 GHz"
      WidthRequest="100"
      Clicked="OnGHzButtonClicked"/>
  </StackLayout>
</ContentView>
```

- **NetworksFilterByGHzView.xaml.cs:**

```
using Microsoft.Maui.Graphics.Text;
using System.Windows.Input;
namespace WiFi_Analyzer.Controls;
public partial class NetworksFilterByGHzView : ContentView
{
  public static readonly BindableProperty FilterByGHzCommandProperty = BindableProperty.Create(
    nameof(FilterByGHzCommand), typeof(ICommand), typeof(SpeedTestView));
  public ICommand FilterByGHzCommand
  {
    get => (ICommand)GetValue(FilterByGHzCommandProperty);
    set => SetValue(FilterByGHzCommandProperty, value);
  }
  public NetworksFilterByGHzView()
            => InitializeComponent();
  readonly Color clickedButtonBackgroundColor = Color.FromHex("#808080");
  readonly Color unclickedButtonBackgroundColor = Color.FromHex("#FFFFFF");
  void OnGHzButtonClicked(object sender, EventArgs e)
  {
    Button button = (Button)sender;
    button.BackgroundColor = clickedButtonBackgroundColor;
    string buttonText = button.Text;
        if (buttonText != twoDotFourGHzButton.Text)
      twoDotFourGHzButton.BackgroundColor = unclickedButtonBackgroundColor;
    if (buttonText != fiveGHzButton.Text)
      fiveGHzButton.BackgroundColor = unclickedButtonBackgroundColor;
    if (buttonText != sixGHzButton.Text)
      sixGHzButton.BackgroundColor = unclickedButtonBackgroundColor;
```

```
        if (buttonText != allGHzButton.Text)
            allGHzButton.BackgroundColor = unclickedButtonBackgroundColor;
    }
}
```

- **SignalStrengthSlider.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Name="this"
             x:Class="WiFi_Analyzer.Controls.SignalStrengthSlider">
    <Slider Minimum="{Binding MinSignalStrengthIndBm}"
            Maximum="{Binding MaxSignalStrengthIndBm}"
            Value="{Binding NetworkStates.SignalStrengthIndBm}"
            MinimumTrackColor="MediumVioletRed"
            MaximumTrackColor="LightGreen"
            WidthRequest="170"
            ThumbColor="DeepPink"
            IsEnabled="False"
            IsVisible="{Binding HasNetworkStates}"
            BindingContext="{x:Reference this}" />
</ContentView>
```

- **SignalStrengthSlider.xaml.cs:**

```
using Microcharts;
using WiFi_Analyzer.Models;
namespace WiFi_Analyzer.Controls;
public partial class SignalStrengthSlider : ContentView
{
    public static readonly BindableProperty NetworkStatesProperty = BindableProperty.Create(
        nameof(NetworkStates), typeof(NetworkStates), typeof(SignalStrengthSlider), propertyChanged:
OnNetworkStatesChanged);
    public NetworkStates NetworkStates
    {
        get => (NetworkStates)GetValue(NetworkStatesProperty);
        set => SetValue(NetworkStatesProperty, value);
    }
    bool hasNetworkStates = false;
    public bool HasNetworkStates
    {
        get => hasNetworkStates;
        private set
        {
            hasNetworkStates = value;
            OnPropertyChanged(nameof(HasNetworkStates));
        }
    }
    public SignalStrengthSlider()
        => InitializeComponent();
    public int MaxSignalStrengthIndBm => -50;
    public int MinSignalStrengthIndBm => -100;
    static void OnNetworkStatesChanged(BindableObject bindable, object oldValue, object newValue)
    {
        var slider = (SignalStrengthSlider)bindable;
        slider.HasNetworkStates = newValue is not null;
    }
}
```

- **SortableHeaderView.xaml:**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentView xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
        xmlns:converters="clr-namespace:WiFi_Analyzer.Converters"
        x:Name="this"
        x:Class="WiFi_Analyzer.Controls.SortableHeaderView">
  <ContentView.Resources>
    <ResourceDictionary>
      <converters:SortDirectionToSymbolConverter x:Key="SortDirectionToSymbolConverter" />
    </ResourceDictionary>
  </ContentView.Resources>
  <Frame BorderColor="White" Padding="0" CornerRadius="0" BindingContext="{x:Reference this}">
    <ContentView HorizontalOptions="Center" VerticalOptions="Center" Margin="5">
      <StackLayout Orientation="Horizontal" HorizontalOptions="Center" VerticalOptions="Center">
        <Label Text="{Binding HeaderText}" FontAttributes="Bold" TextColor="White"/>
        <Label Text="{Binding SortDirection, Converter={StaticResource SortDirectionToSymbolConverter}}" />
      </StackLayout>
      <ContentView.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding SortCommand}" CommandParameter="{Binding
SortCommandParameter}" />
      </ContentView.GestureRecognizers>
    </ContentView>
  </Frame>
</ContentView>
```

- **SortableHeaderView.xaml.cs:**

```
using System.Windows.Input;
using WiFi_Analyzer.Enums;
namespace WiFi_Analyzer.Controls;
public partial class SortableHeaderView : ContentView
{
  public static readonly BindableProperty HeaderTextProperty = BindableProperty.Create(
    nameof(HeaderText), typeof(string), typeof(SortableHeaderView), string.Empty);
  public static readonly BindableProperty SortDirectionProperty = BindableProperty.Create(
    nameof(SortDirection), typeof(OrderBy?), typeof(SortableHeaderView), null);
  public static readonly BindableProperty SortCommandProperty = BindableProperty.Create(
    nameof(SortCommand), typeof(ICommand), typeof(SortableHeaderView), null);
  public static readonly BindableProperty SortCommandParameterProperty = BindableProperty.Create(
    nameof(SortCommandParameter), typeof(string), typeof(SortableHeaderView), string.Empty);
  public string HeaderText
  {
    get => (string)GetValue(HeaderTextProperty);
    set => SetValue(HeaderTextProperty, value);
  }
  public OrderBy? SortDirection
  {
    get => (OrderBy?)GetValue(SortDirectionProperty);
    set => SetValue(SortDirectionProperty, value);
  }
  public ICommand SortCommand
  {
    get => (ICommand)GetValue(SortCommandProperty);
    set => SetValue(SortCommandProperty, value);
  }
  public string SortCommandParameter
  {
    get => (string)GetValue(SortCommandParameterProperty);
    set => SetValue(SortCommandParameterProperty, value);
  }
  public SortableHeaderView()
    => InitializeComponent();
}
```