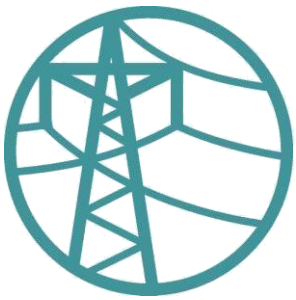


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Karol Bulanowski

Zdalny monitoring przyrządów pomiarowych

Projekt inżynierski

Opiekun pracy:
dr inż. Jakub Wojturski

Rzeszów, 2023

SPIS TREŚCI

1. Wstęp

1.1. Wprowadzenie

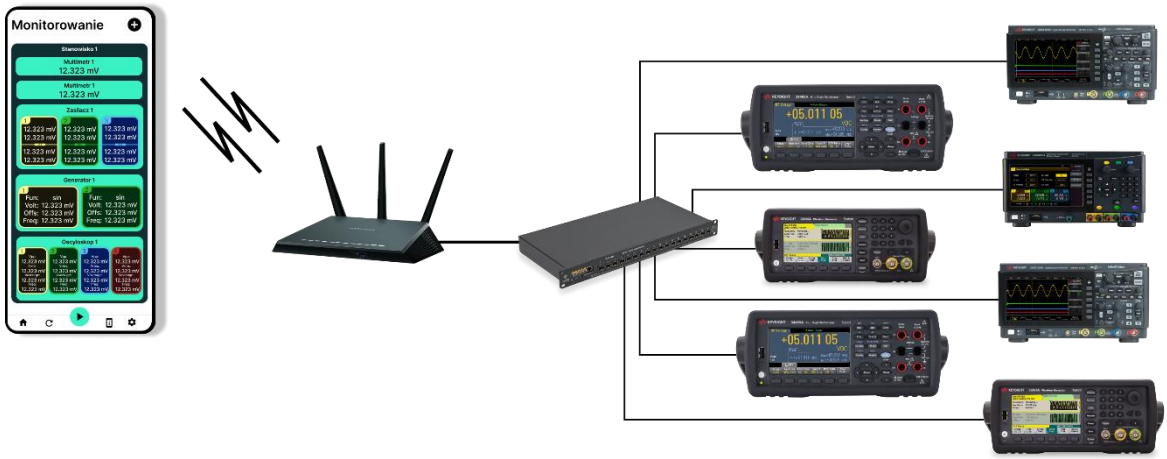
W dzisiejszym zglobalizowanym i technologicznie zaawansowanym świecie, zdalny monitoring urządzeń pomiarowych stał się niezwykle istotnym narzędziem w wielu dziedzinach.

Zdalny monitoring odnosi się do zdolności monitorowania i zarządzania urządzeniami pomiarowymi z dowolnego miejsca, bez konieczności fizycznego obecności operatora na miejscu. Otwiera to nowe perspektywy i umożliwia realizację kompleksowych systemów pomiarowych oraz zapewnia dostęp do danych i informacji w czasie rzeczywistym. Zdalny monitoring urządzeń pomiarowych znajduje zastosowanie w różnych dziedzinach, takich jak przemysł, energetyka, telekomunikacja, medycyna, ochrona środowiska i wiele innych. Przykładowe urządzenia, które mogą być monitorowane zdalnie, to czujniki temperatury, wilgotności, ciśnienia, poziomu substancji chemicznych, prądu, napięcia, a także urządzenia medyczne, takie jak monitory stanu pacjentów czy aparatury diagnostyczne.

Głównym celem zdalnego monitoringu urządzeń pomiarowych jest umożliwienie skutecznego nadzoru i kontroli nad procesami pomiarowymi w czasie rzeczywistym. Dzięki temu operatorzy mogą szybko reagować na wszelkie nieprawidłowości, zmieniać parametry urządzeń lub podejmować odpowiednie działania naprawcze, zanim potencjalne problemy wpłyną na efektywność lub bezpieczeństwo systemu. Zdalny monitoring wprowadza szereg korzyści i rozwiązuje wiele tradycyjnych wyzwań związanych z monitorowaniem urządzeń pomiarowych. Eliminuje konieczność fizycznego sprawdzania stanu urządzeń, co prowadzi do oszczędności czasu i kosztów związanych z podróżami. Ponadto, zdalne monitorowanie umożliwia gromadzenie i analizę danych w czasie rzeczywistym, co zwiększa efektywność procesów, optymalizuje wykorzystanie zasobów i pozwala na podejmowanie informowanych decyzji. Do takiego monitoringu często wykorzystuje się urządzenia przenośne. Są one nierozłączną częścią życia człowieka. Pełnią one ważną rolę w życiu zwykłego użytkownika, jak i w poważnych przedsiębiorstwach. Nierozłączne z urządzeniami mobilnymi są aplikacje, które mocno rozbudowują i poszerzają możliwości tych urządzeń. Jedną z funkcjonalności jest komunikacja z innymi urządzeniami oraz pobieranie i wyświetlanie pomiarów.

1.2. Cel pracy

Celem pracy było stworzenie aplikacji mobilnej, która oferuje możliwość połączenia ze wszystkimi urządzeniami pomiarowymi w sieci lokalnej w standardzie SCPI. Aplikacja ma oferować grupowanie urządzeń w celu odwzorowania ich położenia na poszczególnych stanowiskach w laboratorium. Ma umożliwić sprawny i czytelny monitoring wartości aktualnie mierzonych przez urządzenia pomiarowe przez prowadzącego zajęcia. Na rysunku 1.1 przedstawiono układ pracy aplikacji.



Rys. 1.1. Schemat działania aplikacji

1.3. Motywacja

Wybór tego tematu uzasadnia się tym, że do jego stworzenia wymagana jest znajomość kilku dziedzin z zakresu informatyki, które znajdują się w kręgu zainteresowań oraz są kierunkami, w których chciałoby się rozwijać umiejętności oraz w przyszłości tworzyć, bądź uczestniczyć w podobnych projektach. Mowa o takich dziedzinach jak programowanie aplikacji, znajomość działania urządzeń mobilnych, ich systemu operacyjnego, sieci teleinformatycznych, sposobu komunikacji między urządzeniami mobilnymi i laboratoryjnymi.

2. Technologie

2.1. Wprowadzenie

Przed wykonaniem aplikacji, należało zapoznać się z aktualnie istniejącymi rozwiązaniami na rynku. Znaleziono dostępną w Sklepie Google Play aplikację „IEEE 488.2 SCPI data logger" z 2014r. Oferuje ona jedynie pojedyncze połączenie z jedynym urządzeniem. Kolejnym krokiem było rozplanowanie i zaprojektowanie interfejsu graficznego (ang. User Interface) aplikacji w programie Figma, aby wstępnie rozplanować poszczególne widoki aplikacji oraz położenie wymaganych widżetów. Następnie należało przejrzeć dostępne technologie tj. języki programowania oraz ich platformy (ang. Frameworki), które umożliwiłyby odwzorowanie zaprojektowanego interfejsu graficznego i spełnienie założonych funkcji aplikacji. Po dokładnej analizie możliwości poszczególnych języków wybrano Flutter na podstawie języka Dart, który natywnie oferuje możliwość projektowania aplikacji zgodnie z założeniami Material Design, według których aplikacja została zaprojektowana. Jako środowisko do pisania programu, kompilacji oraz debugowania wybrano uniwersalny edytor Visual Studio Code.

2.2. System operacyjny Android

Android to mobilny system operacyjny opracowany przez Google. Jest to najpowszechniej wykorzystywany system operacyjny na świecie. Działa na miliardach urządzeń, w tym smartfonach, tabletach, inteligentnych zegarkach, okularach, nawigacjach, aparatach, laptopach, mini komputerach, urządzeniach AGD, telewizorach z funkcją Smart TV i wielu innych urządzeniach. Logo systemu przedstawiono na rysunku 2.1.

Rys. 2.1. Logo systemu Android

Oto główne cechy systemu Android [1]:

1) Architektura

System Android bazuje na zmodyfikowanej wersji jądra Linux. Jądro Linux zapewnia podstawowe usługi systemowe, takie jak abstrakcja sprzętowa, zarządzanie procesami i pamięciami. Jądro systemu Android bazuje na architekturze warstwowej, która zawiera różne warstwy, z których każda jest odpowiedzialna za konkretne funkcje.

2) Aplikacje

Android obsługuje ogromny ekosystem aplikacji. Użytkownicy mogą pobierać i instalować aplikacje ze zdalnego repozytorium Google Play Store, które oferuje szeroki wybór aplikacji, gier i narzędzi. Aplikacje Android pisane są głównie w językach takich jak Java, Kotlin, C++, a deweloperzy mogą je dystrybuować za pośrednictwem m. in. Sklepu Play.

3) Bezpieczeństwo

Android posiada wiele warstw zabezpieczeń, aby chronić dane użytkownika i samo urządzenie. Bezpieczeństwo zapewniane jest przez kontrolowanie uprawnień dostępu do zasobów urządzenia jakie aplikacje mogą uzyskiwać, szyfrowanie danych, bezpieczne procesy rozruchowe czy regularne aktualizacje zabezpieczeń.

4) Łączność

Android obsługuje różne opcje nawiązywania łączności np. Wi-Fi, Bluetooth, NFC, podczerwień. Wspiera także różne protokoły komunikacji komórkowej, takie jak GSM, CDMA, 4G LTE i 5G.

5) Wielozadaniowość

Android umożliwia wielozadaniowość, co pozwala na uruchamianie wielu aplikacji jednocześnie i ich działanie w tle. Użytkownicy mogą przełączać się między aplikacjami, korzystać z trybu podziału ekranu i otrzymywać powiadomienia od różnych aplikacji.

6) Interfejs użytkownika

Android posiada konfigurowalny i przyjazny dla użytkownika interfejs (Rys. 2.2). Obejmuje elementy takie jak ekran główny, pasek powiadomień i widok listy aplikacji. Użytkownicy mogą dostosowywać swoje urządzenia poprzez układanie ikon aplikacji, widżetów i zmienianie tła pulpitu.

7) Usługi Google

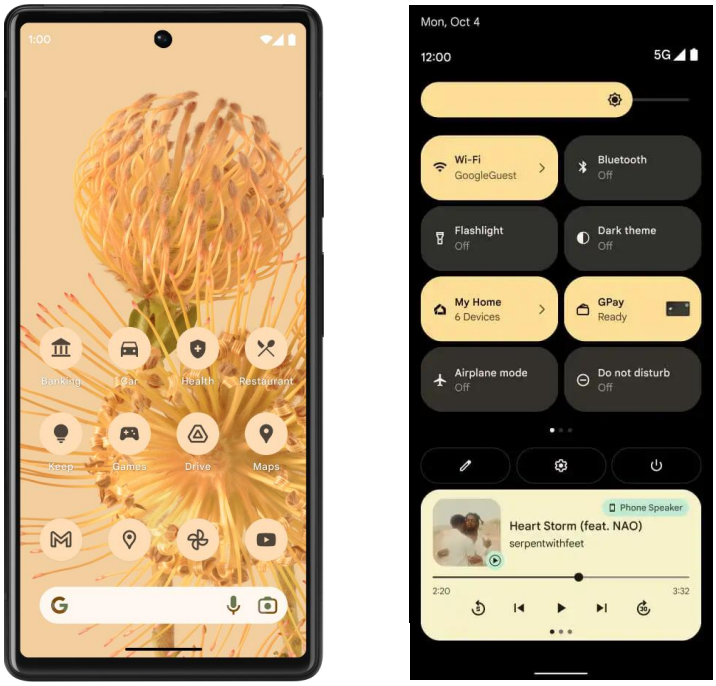
System Android jest silnie zintegrowany z usługami Google, takimi jak Google Search, Google Maps, Gmail, Google Drive, Google Photos. Dzięki tym usługom zwiększona zostaje funkcjonalność urządzeń.

8) Aktualizacje

Android regularnie aktualizowany jest przez Google. Dostarczane są nowe funkcje, poprawki błędów i łatki bezpieczeństwa. Jednak częstotliwość i dostępność aktualizacji mogą się różnić w zależności od konkretnego producenta urządzenia i operatora sieciowego.

9) Dostosowywanie

Android jest bardzo konfigurowalny, zarówno pod względem interfejsu użytkownika, jak i systemu operacyjnego. Producenci urządzeń często dostosowują Androida do swoich interfejsów użytkownika (np. One UI firmy Samsung czy MIUI firmy Xiaomi) i dodają fabrycznie zainstalowane aplikacje.



Rys. 2.2. Interfejs Android 12 i Android 14

Android to elastyczny, wszechstronny i powszechnie używany system operacyjny. Oferuje wiele funkcji i opcji dostosowania zarówno dla użytkowników jak i programistów. Otwarty kod źródłowy przyczynił się do znacznego wzrostu jego popularności oraz adaptacji w różnych rodzajach urządzeń i zastosowań.

2.2.1. Otwartość kodu źródłowego

Kod źródłowy to część oprogramowania, której większość użytkowników komputerów nigdy nie widzi. To kod, którym programiści komputerowi mogą manipulować, aby zmienić sposób działania oprogramowania.

Oprogramowanie „open source” to oprogramowanie z kodem źródłowym, który każdy może przeglądać, modyfikować i ulepszać. Programiści, którzy mają dostęp do kodu źródłowego programu, mogą dodać lub naprawić funkcje, które nie działają poprawnie.

Licencja Open Source charakteryzuje się:

1) swobodnym dostępem do kodu źródłowego, na podstawie którego można tworzyć oprogramowanie,
2) możliwością uruchomienia otwartego kodu bez ponoszenia opłat,
3) prawem do modyfikacji oprogramowania open source dla własnego użytku,
4) zezwoleniem na dystrybucję zmodyfikowanego oprogramowania,
5) koniecznością udostępniania oprogramowania w formie otwartego kodu źródłowego.

Open Source Initative (OSI), organizacja zajmująca się akceptacją licencji otwartego oprogramowania, wyróżnia 5 głównych kryteriów, jakie musi ono spełnić, aby zostać oficjalnie uznane za „open source”:

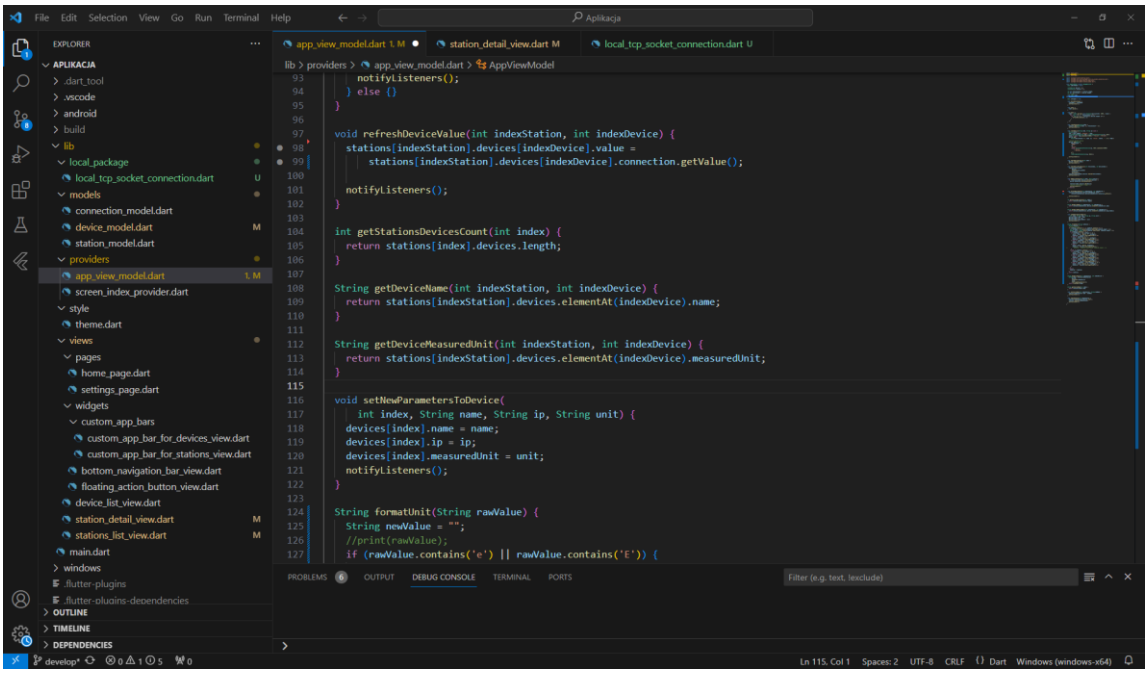
1) całkowita jawność – twórcy kodu nie mogą ukrywać informacji, w tym tych dotyczących bezpieczeństwa,
2) bezpłatność oraz szeroka dostępność,
3) patenty niezbędne do funkcjonowania kodu są udostępnione bez dodatkowych opłat oraz mogą być używane przez nieograniczony czas i nieograniczoną ilość razy,
4) brak zobowiązań w postaci klauzuli poufności, zgód licencyjnych, itp.,
5) zastosowanie konkretnego rozwiązania typu open source nie może wymagać innych technologii niespełniających kryteriów otwartego oprogramowania [2].

2.3. Środowisko programistyczne

2.3.1. Visual Studio Code

Visual Studio Code (znany jako VS Code) to darmowy edytor tekstu typu open source firmy Microsoft (Rys. 2.3). VS Code jest dostępny dla systemów Windows, Linux i macOS. Edytor jest stosunkowo lekki. Pomimo tego zawiera kilka zaawansowanych funkcji, które sprawiły, że VS Code jest jednym z najpopularniejszych środowisk programistycznych w ostatnim czasie.

Sercem Visual Studio Code jest błyskawiczny edytor kodu źródłowego, idealny do codziennego użytku. Dzięki obsłudze setek języków, VS Code pomaga uzyskać natychmiastową produktywność dzięki podświetlaniu składni, dopasowywaniu nawiasów, automatycznemu wcięciu, zaznaczaniu pól, fragmentom i nie tylko. Intuicyjne skróty klawiaturowe, łatwa personalizacja i współtworzone przez społeczność mapowania skrótów klawiaturowych pozwalają z łatwością poruszać się po kodzie.



Rys .2.3. Interfejs Visual Studio Code

Visual Studio Code zawiera wbudowaną obsługę uzupełniania kodu IntelliSense, bogate semantyczne rozumienie kodu i nawigację oraz refaktoryzację kodu. VS Code zawiera interaktywny debugger, dzięki czemu można przechodzić przez kod źródłowy, sprawdzać zmienne, wyświetlać stosy wywołań i wykonywać polecenia w konsoli. Edytor integruje się również z narzędziami do kompilacji i skryptowania. Obsługuje system kontroli wersji Git, dzięki czemu można pracować z kontrolą źródła bez opuszczania edytora [3].

2.3.2. Android Studio (emulator)

Android Studio to oficjalne zintegrowane środowisko programistyczne (ang. Integrated Development Environment) do tworzenia aplikacji na system operacyjny Android. Oparte jest na rozbudowanym edytorze kodu i narzędziach deweloperskich IntelliJ IDEA. W tej pracy wykorzystano jedynie jedną z jego funkcji, jaką jest emulator urządzenia Android (Rys. 2.4).

Rys. 2.4. Widok emulowanego urządzenia Android

Emulator Androida symuluje urządzenia z systemem Android poprzez wirtualizację na komputerze, dzięki czemu można przetestować aplikację na różnych rodzajach sprzętu i poziomach API Androida bez konieczności posiadania każdego fizycznego urządzenia.

Emulator oferuje następujące zalety [4]:

1) Elastyczność: Emulator dostarczany jest z predefiniowanymi konfiguracjami dla różnych urządzeń z systemem Android jak tablety, urządzenia Wear OS i Android TV.
2) Wysokie odwzorowanie: Emulator zapewnia niemal wszystkie możliwości prawdziwego urządzenia z systemem Android. Można symulować przychodzące połączenia telefoniczne i wiadomości tekstowe, określać lokalizację urządzenia, symulować różne prędkości sieci, symulować obrót ekranu i inne czujniki sprzętowe, uzyskiwać dostęp do sklepu Google Play i wiele więcej.

W większości przypadków emulator jest najlepszą opcją do testowania tworzonego rozwiązania.

2.4. MVVM

Model-View-ViewModel (MVVM) to wzorzec architektoniczny oprogramowania, który wspiera oddzielenie interfejsu użytkownika (czyli widoku) od rozwoju logiki biznesowej lub logiki zaplecza (modelu). Architekturę tego wzorca przedstawiono na rysunku 2.5. Model widoku wewnątrz MVVM jest pomostem odpowiedzialnym za konwersję danych w sposób, który zachowuje się zgodnie ze zmianami zachodzącymi w UI.

Rys. 2.5. Diagram działania wzorca MVVM

Kilka zalet korzystania z MVVM:

1) separacja zagadnień: Jest to zasada projektowania polegająca na podzieleniu programu komputerowego na odrębne sekcje, tak aby każda sekcja dotyczyła odrębnego problemu. Problemem jest wszystko, co ma znaczenie dla rozwiązania danego problemu,
2) ulepszona testowalność,
3) zdefiniowana struktura projektu,
4) równoległy rozwój interfejsu użytkownika,
5) abstrakcja widoku, a tym samym zmniejszenie ilości logiki biznesowej wymaganej w kodzie za nim.

Niektóre wady korzystania z MVVM:

1) Ma nieco stromą krzywą uczenia się. Zrozumienie sposobu działania wszystkich warstw może zająć trochę czasu.
2) Dodaje wiele dodatkowych klas, więc nie jest idealny dla projektów o niskim stopniu złożoności [5].

2.5. Język Dart i Flutter

Dart to interpretowany język programowania wysokiego poziomu. Został opracowany przez Google, jest zoptymalizowany pod kątem użytkownika i przeznaczony do tworzenia szybkich aplikacji na dowolną platformę. Jego celem jest zaoferowanie najbardziej produktywnego języka programowania dla rozwoju wieloplatformowego, w połączeniu z elastyczną platformą wykonawczą dla frameworków aplikacji. Jest alternatywą dla języka JavaScript. Pierwsza wersja interpretera zaczęła obowiązywać w połowie jesieni 2011 roku. Na rysunku 2.6 przedstawiono logo języka Dart.

Rys. 2.6. Logo języka Dart

Dart zyskał większą uwagę w 2017 roku, kiedy Google oficjalnie ogłosiło „Flutter beta„ jako narzędzie do tworzenia wieloplatformowych aplikacji mobilnych. Od tego czasu popularność Darta drastycznie wzrosła [6].

Flutter to przenośny zestaw narzędzi UI firmy Google do tworzenia świetnie wyglądających, natywnie kompilowanych aplikacji mobilnych, internetowych i desktopowych z jednej bazy kodu. Flutter działa z istniejącym kodem, jest używany przez programistów i organizacje na całym świecie. Jest darmowy, a jego kod źródłowy otwarty. Na rysunku 2.7 ukazano logo Flutter.

Rys. 2.7. Logo Flutter

Flutter różni się od większości innych opcji tworzenia aplikacji mobilnych, ponieważ nie opiera się na technologii przeglądarki internetowej ani zestawie widżetów dostarczanych z każdym urządzeniem. Zamiast tego Flutter wykorzystuje własny, wysokowydajny silnik renderujący do rysowania widżetów. Ponadto Flutter cechuje się tym, że ma tylko cienką warstwę kodu C/C++. Flutter implementuje większość swojego systemu (kompozycja, gesty, animacje, framework, widżety itp.) w systemie języka Dart, który można łatwo odczytać, zmienić, zastąpić lub usunąć. Daje to deweloperom ogromną kontrolę nad systemem, a także znacznie obniża poprzeczkę dostępności dla większości systemu [7].

2.6. TCP/IP

TCP/IP to zbiór protokołów sieciowych, umożliwiający komputerom współużytkowanie zasobów i wymianę informacji w sieci. Umożliwia hostom komunikowanie się ze sobą bez względu na położenie fizyczne ich własne lub użytkownika, używany system operacyjny czy media transmisyjne. Protokół TCP/IP działa w wielu różnych środowiskach sieciowych, w tym w Internecie i korporacyjnych intranetach. TCP/IP dokładnie definiuje sposób przenoszenia informacji od nadawcy do odbiorcy. Najpierw programy użytkowe wysyłają komunikaty lub strumienie danych do jednego z protokołów warstwy transportowej (ang. Internet Transport Layer Protocol), czyli Protokołu UDP (ang. User Datagram Protocol) lub Protokołu TCP (ang. Transport Control Protocol). Te protokoły odbierają dane z aplikacji, dzielą je na mniejsze części zwane pakietami, dodają adres docelowy, a następnie przekazują pakiety wraz z następną warstwą protokołu, warstwą sieci Internet.

Warstwa sieci Internet zamyka pakiet w datagramie Internet Protocol (IP), umieszcza w nagłówku datagramu. Następnie decyduje, gdzie wysłać datagram, bezpośrednio do miejsca docelowego lub do bramy (ang. gateway), a następnie przekazuje datagram do warstwy interfejsu sieciowego. Warstwa interfejsu sieciowego akceptuje datagramy IP i przesyła je jako ramki za pośrednictwem konkretnego sprzętu sieciowego, takiego jak sieci Ethernet lub Token Ring [8].

2.6.1. WireShark

Wireshark to analizator pakietów sieciowych. Analizator pakietów sieciowych prezentuje przechwycone dane pakietów w możliwie najbardziej szczegółowy sposób. Można myśleć o analizatorze pakietów sieciowych jako o urządzeniu pomiarowym do badania tego, co dzieje się wewnątrz kabla sieciowego, podobnie jak elektryk używa woltomierza do badania tego, co dzieje się wewnątrz kabla elektrycznego (ale oczywiście na wyższym poziomie). W przeszłości takie narzędzia były albo bardzo drogie, albo zastrzeżone, albo jedno i drugie. Jednak wraz z pojawieniem się Wiresharka to się zmieniło. Wireshark jest dostępny za darmo, ma otwarty kod źródłowy i jest jednym z najlepszych dostępnych obecnie analizatorów pakietów [9].

2.7. Język SCPI

W 1987 roku IEEE wydało "IEEE 488.2-1987, Codes, Formats, Protocols and Common Commands for Use with IEEE 488.1-1987". Standard ten definiował role przyrządów i kontrolerów w systemie pomiarowym oraz ustrukturyzowany schemat komunikacji. IEEE 488.2 w szczególności opisuje sposób wysyłania poleceń do przyrządów i wysyłania odpowiedzi do kontrolerów. Niektóre często używane polecenia "porządkowe" (ang. Housekeeping) zostały wyraźnie zdefiniowane, ale każdy producent przyrządu miał za zadanie nazwać wszelkie inne typy poleceń i zdefiniować ich działanie. Norma IEEE 488.2 określała, w jaki sposób należy zaimplementować określone typy funkcji, jeśli zostały one uwzględnione w przyrządzie. Zasadniczo nie określono, które funkcje lub polecenia powinny być zaimplementowane w konkretnym urządzeniu. W związku z tym możliwe było, że dwa podobne przyrządy byłyby zgodne z normą IEEE 488.2, ale mogłyby mieć zupełnie inny zestaw poleceń. Standard Commands for Programmable Instruments (SCPI) to nowy język poleceń do sterowania przyrządami, który wykracza poza normę IEEE 488.2 i obejmuje szeroki zakres funkcji przyrządów w standardowy sposób.

SCPI promuje spójność, z punktu widzenia zdalnego programowania, między przyrządami tej samej klasy i między przyrządami o tych samych możliwościach funkcjonalnych. Dla danej funkcji pomiarowej, takiej jak częstotliwość lub napięcie, SCPI definiuje konkretny zestaw poleceń dostępny dla tej funkcji. W ten sposób dwa oscyloskopy różnych producentów mogą być używane do wykonywania pomiarów częstotliwości w ten sam sposób. Możliwe jest również, aby licznik SCPI dokonywał pomiaru częstotliwości przy użyciu tych samych poleceń, co oscyloskop. Polecenia SCPI są łatwe do nauczenia, nie wymagają objaśnień i mogą być używane zarówno przez początkujących, jak i doświadczonych programistów. Po zapoznaniu się z organizacją i strukturą SCPI można osiągnąć znaczny wzrost wydajności podczas opracowywania programu sterującego, niezależnie od wybranego języka programu sterującego [10].

2.8. Figma i Material Design

Figma to narzędzie do projektowania interfejsów, które szturmem zdobyło świat designu. W przeciwieństwie do innych programów graficznych, który działają jako samodzielne oprogramowanie, Figma jest całkowicie oparta na przeglądarce, a zatem działa na komputerach MacOS, PC z systemem Windows lub Linux, a nawet na Chromebookach. Największą zaletą tego programu jest to, że jest darmowa i oferuje swoje webowe API. Kolejną dużą zaletą Figma jest to, że umożliwia współpracę w czasie rzeczywistym nad tym samym plikiem. Podczas korzystania z konwencjonalnych aplikacji "offline", takich jak Sketch i Photoshop, jeśli projektanci chcą udostępnić swoją pracę, zazwyczaj muszą wyeksportować ją do pliku graficznego, a następnie wysłać e-mailem lub przez komunikator internetowy. W aplikacji Figma, zamiast eksportować statyczne obrazy, można po prostu udostępnić link do pliku Figma, aby klienci i współpracownicy mogli go otworzyć w przeglądarce. To samo w sobie oszczędza znaczną ilość czasu i niedogodności w przepływie pracy projektanta. Ale co ważniejsze, oznacza to, że klienci i współpracownicy mogą wchodzić w interakcje z pracą i przeglądać najnowszą wersję pliku [11].

Material Design to stworzony przez Google język projektowania zorientowany na Androida i aplikacje webowe. To adaptowalny system wytycznych, komponentów i narzędzi, które wspierają najlepsze praktyki projektowania interfejsów użytkownika. Wspierany przez kod open-source, Material usprawnia współpracę między projektantami i programistami co skutkuje sprawnym tworzeniem pięknych, funkcjonalnych rozwiązań [12].

2.9. System kontroli wersji Git

Git służy do przechowywania kodu źródłowego projektu i śledzenia pełnej historii wszystkich zmian w tym kodzie. Pozwala on programistom na bardziej efektywną współpracę nad projektem, zapewniając narzędzia do zarządzania potencjalnie sprzecznymi zmianami od wielu programistów [13].

2.9.1. Platforma Github

GitHub to internetowa platforma kontroli wersji i współpracy dla twórców oprogramowania. Microsoft, największy pojedynczy udziałowiec GitHub, nabył platformę za 7,5 miliarda dolarów w 2018 roku. GitHub, który jest dostarczany za pośrednictwem modelu biznesowego oprogramowania jako usługi (SaaS), został uruchomiony w 2008 roku. Jego podstawą był Git, system zarządzania kodem open source stworzony przez Linusa Torvaldsa w celu przyspieszenia kompilacji oprogramowania. GitHub pozwala deweloperom zmieniać, dostosowywać i ulepszać oprogramowanie z publicznych repozytoriów za darmo, ale pobiera opłaty za prywatne repozytoria, oferując różne płatne plany. Każde publiczne i prywatne repozytorium zawiera wszystkie pliki projektu, a także historię zmian każdego pliku. Repozytoria mogą mieć wielu współpracowników i mogą być publiczne lub prywatne [13].

2.9.3. Github flow

GitHub Flow to prostsza alternatywa dla GitFlow, idealna dla mniejszych zespołów, które nie muszą zarządzać wieloma wersjami. Metoda ta została użyta w pracy. W przeciwieństwie do konkurencyjnego GitFlow, model ten nie posiada gałęzi wydań (ang. Release). Zaczyna się od głównej gałęzi, a następnie programiści tworzą gałęzie funkcji, które rozgałęziają się bezpośrednio z głównej gałęzi, aby odizolować swoją pracę, która jest następnie scalana z główną gałęzią. Po ukończeniu pracy nad daną funkcją gałąź funkcji jest usuwana. Główną ideą tego modelu jest utrzymywanie kodu głównego w stałym stanie nadającym się do wdrożenia, dzięki czemu może on wspierać procesy ciągłej integracji i ciągłego dostarczania.

Zalety i wady modelu GitHub Flow:

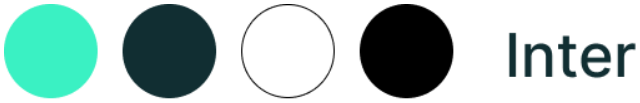
1) Github Flow koncentruje się na zasadach Agile [1], więc jest to szybka i usprawniona strategia rozgałęziania z krótkimi cyklami produkcyjnymi i częstymi wydaniami.
2) Z powodu braku gałęzi rozwojowej, testuje się i automatyzuje zmiany w jednej gałęzi, co pozwala na szybkie, sprawne i ciągłe wdrażanie.

3) Strategia ta jest szczególnie odpowiednia dla małych zespołów i aplikacji internetowych i jest idealna, gdy trzeba utrzymać jedną wersję produkcyjną. Zatem w miarę powiększania się zespołu mogą wystąpić konflikty scalania, ponieważ wszyscy łączą się z tą samą gałęzią i brakuje przejrzystości, co oznacza, że programiści nie mogą zobaczyć, nad czym pracują inni współpracownicy. W związku z tym strategia ta nie jest odpowiednia do obsługi wielu wersji kodu.
4) Brak gałęzi rozwojowych sprawia, że strategia ta jest bardziej podatna na błędy [14].

3. Projektowanie

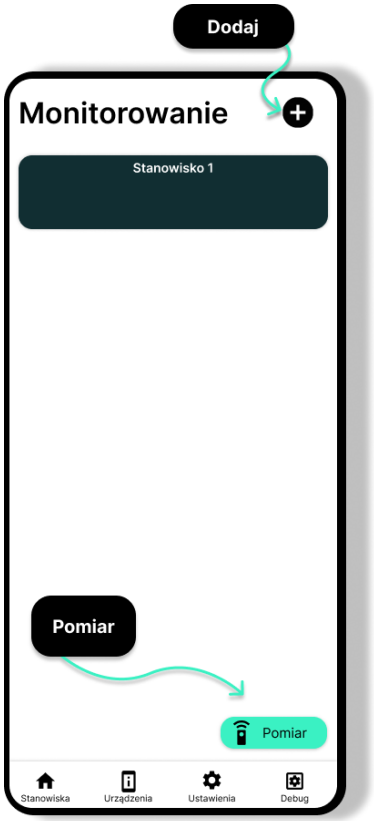
3.1. Projekt graficzny

Jak wcześniej wspomniano, zdecydowano się na wykonanie aplikacji według zaleceń "Material Design". Przed przystąpieniem do wykonania projektu funkcjonalnego aplikacji sporządzono projekt graficzny, który miał unaocznić i zwizualizować sposób rozmieszczenia poszczególnych funkcji. Przestrzeń urządzenia mobilnego typu smartfon jest dość ograniczona, dlatego też każde położenie elementu powinno być przemyślane. Należało też wziąć pod uwagę popularne i przyjęte przez użytkowników gesty. Umożliwiają one spełnienie zaprojektowanych funkcji bez konieczności zamieszczania widocznych elementów w interfejsie aplikacji.



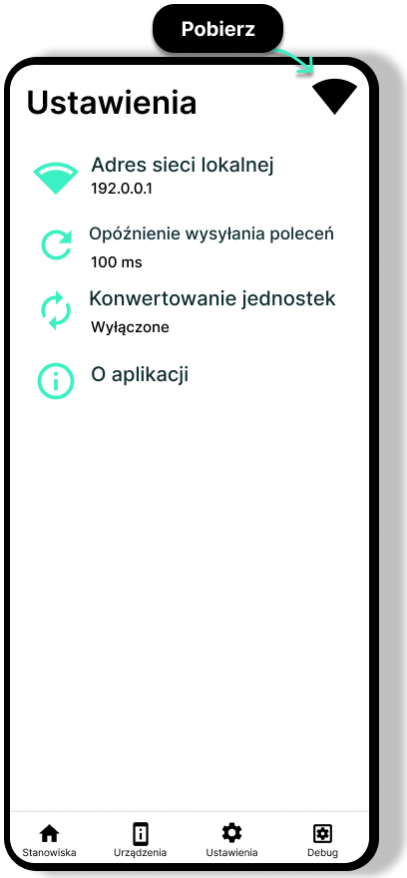
Rys. 3.1. Wykorzystane kolory oraz czcionka w projekcie

Kolorystykę aplikacji oparto na czterech kolorach: "green-light" (#3AF1C3), "green" (#112E32), "white" (#FFFFFF) , "black" (#000000). Jako czcionkę wybrano rodzinę czcionki "Inter" (Rys. 3.1).



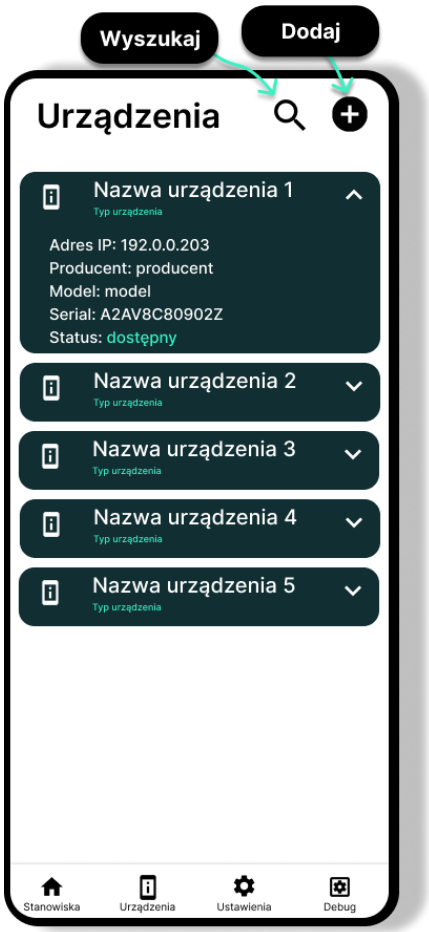
Rys. 3.2. Widok strony głównej po uruchomieniu aplikacji

Na rysunku 3.2 przedstawiono zaprojektowany widok zakładki "Stanowiska". Jest to widok domyślnie otwierany podczas pierwszego uruchamiania aplikacji. W głównym miejscu znajduje się pusta karta o nazwie "Stanowisko 1". Ukazuje w jaki sposób wyświetlane będą poszczególne stanowiska po dodaniu za pomocą przycisku wskazanego jako "Dodaj". Przycisk "Pomiar" umożliwia rozwinięcie opcji i wybranie rodzaju pomiaru – pojedynczego, bądź ciągłego.



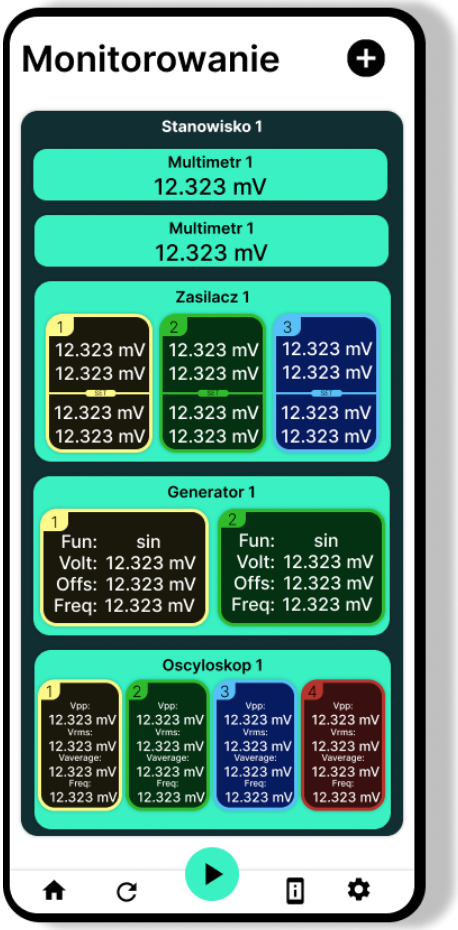
Rys. 3.3. Widok strony "Ustawienia"

Na rysunku 3.3 pokazano wygląd zakładki "Ustawienia". Zawarte są tutaj podstawowe ustawienia aplikacji. Przycisk "Pobierz", służy do pobrania adresów aktualnie połączonej sieci Wi-Fi.



Rys. 3.4. Widok strony "Urządzenia"

Widok strony "Urządzenia" (Rys. 3.4) zaprojektowano tak, aby wszystkie informacje o danym urządzeniu były widoczne, przy jednoczesnym zachowaniu kompaktowości. Dlatego też zastosowano rozwijające penele zawierające klikalny kontener z informacjami o urządzeniu. Przycisk „Wyszukaj” otworzy okno dialogowe i rozpocznie wyszukiwanie urządzeń w sieci. Natomiast przycisk „Dodaj” otworzy okno dialogowe z polami adresów urządzenia, z którym nastąpi próba połączenia. Po rozwinięciu panelu danego urządzenia możliwe jest naciśnięcie całego kontenera z informacjami. Wywoła to wyświetlenie okna dialogowego z ustawieniami i możliwością przypisania urządzenia do stanowiska.



Rys. 3.5. Widok strony głównej stanowisk z dodanymi urządzeniami

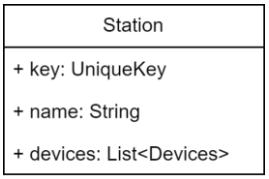
Na rysunku 3.5 ukazano widok zakładki "Stanowiska", w którym to wyświetlone są stanowiska z dodanymi do nich urządzenia różnego typu. Widoczny jest także wygląd poszczególnych kanałów tych urządzeń zależny od typu. Każde widżet „Stanowisko” oraz „Urządzenie” w stanowisku znajduje się w liście z możliwością zmiany kolejności. Sprawia to, że poprzez przytrzymanie danego elementu możliwa jest przemieszczenie go i zmiana jego indeksu (pozycji) w liście.

Rys. 3.6. Zaprojektowane logo aplikacji „SCPI Monitor”

Na końcu zaprojektowano logo aplikacji, które przedstawiono na rysunku 3.6. Na tym etapie zdecydowało się aplikację nazwać „SCPI Monitor”. Logo składa się z czterech ikon mające imitować modele urządzeń. Na ich ekranach umieszczono kolejne litery słowa „SCPI”.

3.2. Projekt funkcjonalny

Jednym z najistotniejszych elementów procesu wytwórczego oprogramowania jest komunikacja. Przełożenie skomplikowanych reguł biznesowych z języka biznesu na język informatyki to niewątpliwie trudne działanie, które trzeba wykonać budując lub modyfikując oprogramowanie. Język naturalny (mowa, pismo), którym posługujemy się by zobrazować rzeczywistość może okazać się zbyt skomplikowany i niejednoznaczny na poszczególnych etapach opisywania powstającego systemu. Niezbędny jest więc taki sposób opisu, który byłby jednakowo interpretowany i zrozumiały dla wszystkich członków zespołu projektowego. Standardem w tej dziedzinie stał się język UML (ang. Unified Modelling Language) – graficzny system wizualizacji, specyfikowania oraz dokumentowania składników systemów informatycznych. UML to notacja umożliwiająca zaprezentowanie systemu w sposób graficzny, w postaci diagramów. Modele zapisane w języku UML prezentują system od ogółu do szczegółu, umożliwiając oglądanie modelu systemu z wybraną w danym momencie szczegółowością. Dlatego też przed przystąpieniem do programowania aplikacji należało przygotować model UML ukazujący poszczególne klasy, obiekty czy metody.

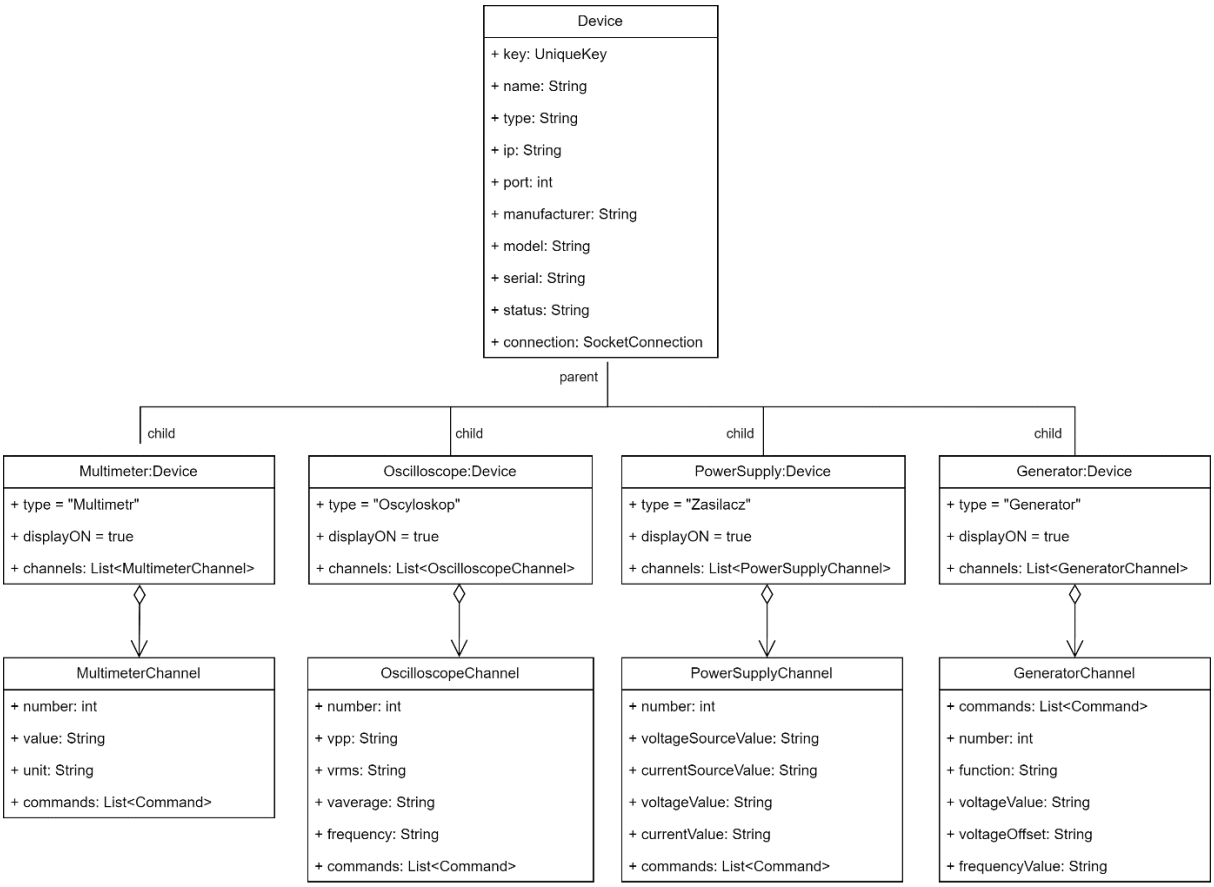


Rys. 3.6. Model stanowiska (ang. Station) w języku UML

Na rysunku 3.6 przedstawiono budowę obiektu klasy "Station", w którym to poszczególne pola odpowiadają za:

1) key: argumentem jest obiekt klasy "Unique Key", którego konstruktor tworzy klucz równy tylko sobie, a wszystkie instancje kluczy są unikalne,
2) name: argumentem jest obiekt typu "String", który oznacza nazwę stanowiska,
3) devices: argumentem jest list obiektów typu "Device", które to będą modyfikowane przez działania użytkownika.

Ważną częścią aplikacji jest implementacja 4 typów obsługiwanych urządzeń tj. multimetr, generator przebiegów, oscyloskop, zasilacz. Należało przeanalizować, które pola i atrybuty dla każdego z urządzeń będą takie same lub pokrewne. Zdecydowano się na utworzenie tych klas w stosunku dziecko-rodzic (ang. parent-child), których schemat ukazano na rysunku 3.7.



Rys. 3.8. Model urządzeń (Device) w języku UML

Jak można zauważyć zdecydowano się na utworzenie klasy "Device" typu rodzic o polach:

1) key: argumentem jest obiekt klasy "Unique Key", którego konstruktor tworzy klucz równy tylko sobie. Wszystkie instancje kluczy są unikalne,
2) name: argumentem jest obiekt typu "String", który oznacza nazwę urządzenia,
3) type: argumentem jest obiekt typu "String", który oznacza typ urządzenia,
4) ip: argumentem jest obiekt typu "String", który oznacza adres IP urządzenia,
5) port: argumentem jest obiekt typu "String", który oznacza adres PORT urządzenia,
6) manufacturer: argumentem jest obiekt typu "String", który oznacza producenta urządzenia,
7) model: argumentem jest obiekt typu "String", który oznacza model urządzenia,

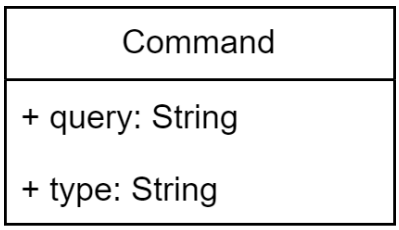
8) serial: argumentem jest obiekt typu "String", który oznacza unikalny numer serial urządzenia,
9) status: argumentem jest obiekt typu "String", który oznacza status urządzenia,
10) connection: argumentem jest obiekt klasy "SocketConnection", który zawiera połączenie urządzenia.

Dalej, dla poszczególnych urządzeń utworzono osobne klasy o adekwatnych do typu urządzenia nazwach typu dziecko, które dziedziczą pola klasy "Device". Każde urządzenie posiada pola obiektu "Device" oraz:

1) type: tutaj jako argument przypisywana jest obiekt typu "String" o wartości zależnej od danego typu urządzenia (Multimetr, Oscyloskop, Zasilacz, Generator),
2) displayON: argumentem jest zmienna typu "bool" oznaczająca włączenie ekranu danego urządzenia,
3) channels: argumentem jest lista obiektów w zależności od typu obiektu (MultimeterChannel, OscilloscopeChannel, PowerSupplyChannel, GeneratorChannel) tj. lista kanałów posiadanych przez dane urządzenie.

Każdy kanał danego urządzenia posiada pola zależne od wartości oczekiwanych i mierzonych przez dane urządzenie. Wspólnymi polami są:

1) number: argumentem jest zmienna typu "int" oznaczająca numer kanału,
2) commands: argumentem jest lista obiektów typu "Command" tj. lista komend wysyłanych do danego typu urządzenia w celu pobrania oczekiwanych wartości, generowana na podstawie wartości pola "number".

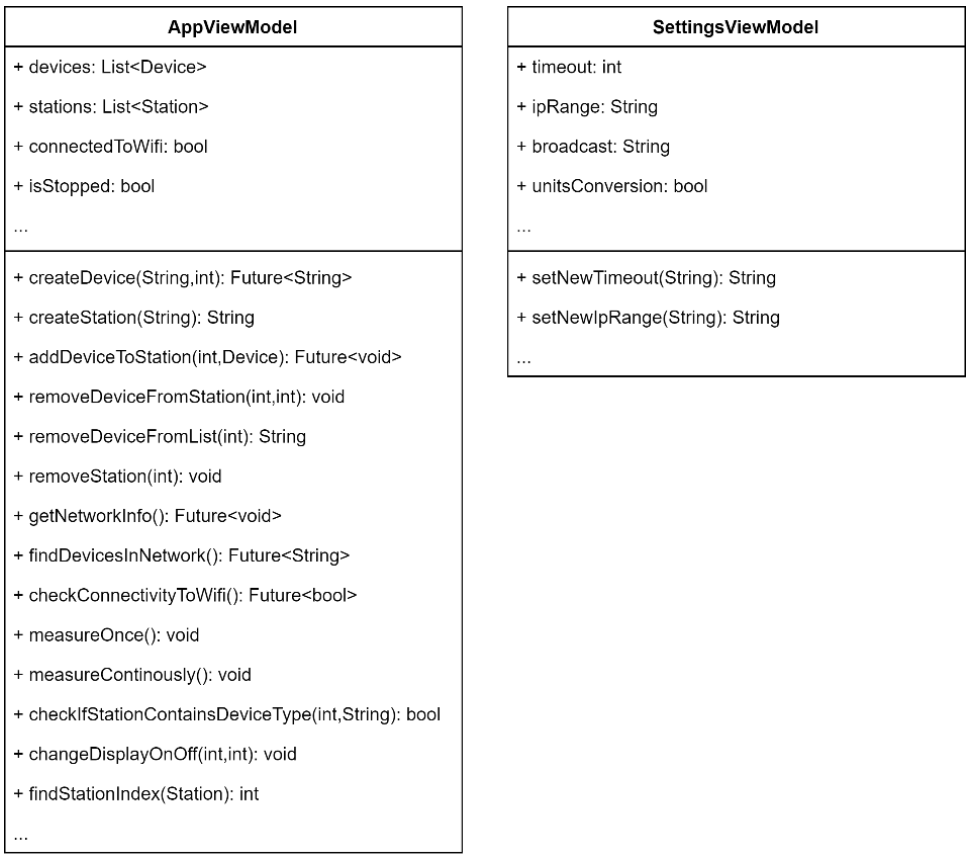


Rys. 3.9. Model komendy (Command) w języku UML

Zgodnie z rysunkiem 3.9 obiekt klasy "Command" ma takie pola jak:

1) query: argumentem jest zmienna typu "String" zawierająca dane polecenie,
2) type: argumentem jest obiekt typu "String" oznaczający typ polecenia (SET – ustawiający, READ – pobierający).

Najważniejszą częścią aplikacji jest (zgodnie z MVVM) widok modelu. Na rysunku 3.10 przedstawiono zaprojektowane dwa modele widoków.



Rys. 3.10. Model widoków modelu (View Models) w języku UML

Widok modelu "AppViewModel" m.in. zawiera pola takie jak:

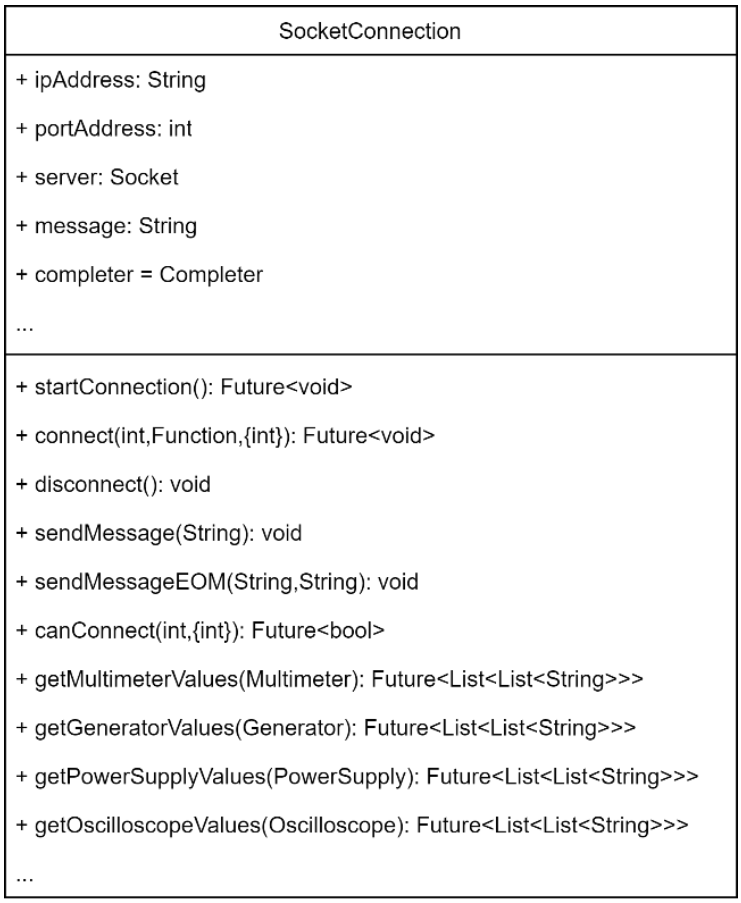
1) devices: lista obiektów typu "Device", zawierająca dostępne urządzenia pomiarow w sieci,
2) stations: lista obiektów typu "Station", zawierająca tworzone i modyfikowane stanowiska przez użytkownika,
3) connectedToWifi: zmienna typu "bool", określająca czy nawiązano połączenie z siecią Wi-Fi,
4) isStopped: zmienna typu "bool", określająca czy zatrzymano wykonywanie pomiarów.

Natomiast widok modelu "SettingsViewModel" odpowiada za ustawienia aplikacji. Zawiera pola:

1) timeout: zmienna typu "int", oznaczająca wartość opóźnienia pomiędzy wykonywanymi pomiarami (ms),
2) ipRange: zmienna typu "String", oznaczająca pobrany adres połączonej sieci Wi-Fi,

3) broadcast: zmienna typu "String", określająca pobrany adres rozgłoszeniowy (ostatni) połączonej sieci Wi-Fi,
4) unitsConversion: zmienna typu "bool", określająca włączenie opcji konwersji liczb.

Ważnym elementem jest także klasa SocketConnection (Rys 3.11). Oparta jest ona na pakiecie "tcp_socket_connection", z której to wybrano przydatne fragmenty kodu.



Rys. 3.11. Model klasy SocketConnection w języku UML

Klasa SocketConnection zawiera takie pola i metody jak:

1) ipAddress: zmienna typu "String", oznaczająca adres IP połączenia,
2) portAddress: zmienna typu "int", oznaczająca adres port połączenia,
3) server: zmienna typu "Socket", oznaczająca połączenie,
4) message: zmienna typu "String", oznaczająca odebraną wiadomość,
5) completer: zmienna typu "Completer", oznaczająca obiekt typu "Future",
6) startConnection(): metoda rozpoczynająca połączenie metodą "connect()",
7) connect(): metoda ustanawiająca połączenie,
8) disconnect(): metoda kończąca połączenie,
9) sendMessageEOM(): metoda wysyłająca ciąg znaków do urządzenia ze znakiem końca linii,

10) sendMessage(): metoda wysyłająca ciąg znaków do urządzenia,
11) canConnect(): metoda sprawdzająca, czy można nawiązać połączenie,
12) getMultimeterValues(), getGeneratorValues(), getPowerSupplyValues(), getOscilloscopeValues(): metody asynchroniczne pobierające wartości urządzeń.

4. Implementacja

4.1. Funkcje

Przed przystąpieniem do kodowania wykonano rekonesans w celu znalezienia urządzeń marki Keysight obsługujących połączenie Socket poprzez LAN. Wybrano interesujące urządzenia i przypisano ich modele do list w programie. Listy urządzeń przedstawiono na listingu 4.1 poniżej.

```
List<String> multimeters = ["34470A", "EDU34450A", "34465A", "34461A"];
List<String> powerSupplies =
["EDU36311A","E36155AGV","E36312A","E36155A","E36155AGV","E36312A","E36155ABV
","E36154ABV","E36154AGV","E36154A","E36233A","E36234A","E36313A","E36232A","
E36231A","E36155ABVX","E36155AGVX","E36154ABVX","E36154AGVX"];
List<String> generators =
["33621A","33622A","33612A","33611A","33522B","33521B","33520B","33519B","335
12B","33511B","33510B","33509B","EDU33211A","EDU33212A"];
List<String> oscilloscopes =
["DSOX3012G","DSOX1204A","DSOX1204G","DSOX1202G","EDUX1052A","EDUX1052G","DSO
X3104G","DSOX3102G","DSOX3054G","DSOX3052G","DSOX3034G","DSOX3032G","DSOX3024
G","DSOX3022G","DSOX3014G","MSOX3104G","MSOX3102G","MSOX3054G","MSOX3052G","M
SOX3034G","MSOX3032G","MSOX3024G","MSOX3022G","MSOX3014G","MSOX3012G"];
```

List. 4.1. Listy urządzeń

Dodatkowo utworzono funkcję "detectDeviceType" przyjmującą jako argument pobrany z urządzenia model typu String i po sprawdzeniu list zwraca typ danego urządzenia typu String (List. 4.2).

```
String detectDeviceType(String model) {
  if (multimeters.contains(model)) {
    return "Multimetr";
  } else if (oscilloscopes.contains(model)) {
    return "Oscyloskop";
  } else if (powerSupplies.contains(model)) {
    return "Zasilacz";
  } else if (generators.contains(model)) {
    return "Generator";
  } else {
    return "Brak informacji";
  }
}
```

List. 4.2. Kod funkcji „detectDeviceType”

Następnie, zgodnie z architekturą MVVM, przystąpiono do zakodowania założonych w rozdziale 3.2 modeli. Na poniższym listingu 4.3 przedstawiono kod opisujący klasę urządzeń "Device" pełniącą rolę rodzica.

```
class Device {
  UniqueKey key;
  String name;
  String type;
  String ip;
  int port;
  String manufacturer;
  String model;
  String serial;
  String status;
  SocketConnection connection;

  Device({
      required this.key,
      required this.name,
      required this.type,
      required this.ip,
      required this.port,
      required this.manufacturer,
      required this.model,
      required this.serial,
      required this.status,
      required this.connection});
}
```

List. 4.3. Kod klasy „Device”

Na poniższym listingu 4.4 ukazano kod opisujący klasę dla danego urządzenia (dziecka). Zamieszczono kod dla urządzenia generator jako przykład, ponieważ schemat klasy dla każdego urządzenia jest taki sam. Przypisywana jest tutaj wartość pola "displayON" na "true" oraz zainicjowana jest lista kanałów dla danego typu urządzenia. Klasa ta dziedziczy pola z klasy Device, zmieniana jest jedynia wartość pola "type" na String z nazwą typu urządzenia.

```
class Generator extends Device {
  bool displayON = true;
  List<GeneratorChannel> channels = [
    GeneratorChannel(1, "-", "0", "0", "0"),
    GeneratorChannel(2, "-", "0", "0", "0")
  ];
  Generator({
    key,
    name,
    ip,
    port,
    manufacturer,
    model,
    serial,
    status,
    connection,
    required this.displayON,
  }) : super(
            key: key,
            name: name,
            type: "Generator",
            ip: ip,
            port: port,
            manufacturer: manufacturer,
            model: model,
            serial: serial,
            status: status,
            connection: connection);
  @override
  String toString() {
    return "Generator";
  }
}
```

List. 4.4. Kod klasy typu urządzenia „Generator”

Dalej zaprogramowano model klasy "Command" opisujący komendę (Listing 4.5).

```
class Command {
  String type;
  String query;
  Command({required this.type, required this.query});
}
```

List. 4.5. Kod klasy „Command”

Do działania powyższych klas stworzono klasy kanałów. Dla przykładu podano klasę "GeneratorChannel" tj. kanał generatora (List. 4.6). W zależności od typu urządzenia w klasie znajdują się inne pola oraz wartości "query" komend. Komendy te generowane są przez konstruktor na podstawie podanego numeru kanału.

```
class GeneratorChannel {
  final int number;
  String function;
  String voltageValue;
  String voltageOffset;
  String frequencyValue;

  late List<Command> commands;
  GeneratorChannel(this.number, this.function, this.voltageValue,
      this.voltageOffset, this.frequencyValue) {
    commands = [
      Command(type: "READ", query: "source$number:function?"),
      Command(type: "READ", query: "source$number:voltage?"),
      Command(type: "READ", query: "source$number:voltage:offset?"),
      Command(type: "READ", query: "source$number:frequency?")
    ];
  }
}
```

List. 4.6. Kod klasy „GeneratorChannel" podanej jako przykład dla urządzenia typu generator

Kolejno zaprogramowano model klasy "Station" opisujący stanowisko (List. 4.7). Każde stanowisko ma nazwę, unikalny klucz, oraz inicjowaną pustą listę urządzeń.

```
class Station {
  UniqueKey key;
  String name;
  List devices = [];

  Station({
    required this.devices,
    required this.key,
    required this.name,
  });
}
```

List. 4.7. Kod klasy „Station"

Następnie według MVVM stworzono główny plik pełniący funkcję Modelu Widoku (ang. View Model). Zawiera on większość metod i pól. Ze względu na obszerności pliku przedstawiono jedynie jego najważniejsze funkcje. Należy wspomnieć, że w celu ograniczenia ilości powtarzanego kodu wykorzystano, który tworzyłby się przy tworzeniu poszczególnych stanów interfejsu aplikacji wykorzystano paczkę "Provider". Usprawnia i zdecydowanie ułatwia ona kodowanie poszczególnych stanów interfejsu aplikacji. Na poniższym listingu 4.9 przedstawiono pola klasy takie jak: devices – lista urządzeń, stations – lista stanowisk z domyślnie utworzonym pustym stanowiskiem oraz innymi mniej ważnymi polami obsługującymi widok.

```
class AppViewModel extends ChangeNotifier {
...
  List<Device> devices = [];
  List<Station> stations = [
    Station(devices: [], key: UniqueKey(), name: "Stanowisko 1")
  ];
  int get stationsCount => stations.length;
  int get devicesCount => devices.length;
  bool connectedToWIFI = false;
  bool isStopped = true;
  bool isPanelExpanded = false;
  String textInfo = "";
  String getTextInfo() => textInfo;
  late bool findDevicesInNetworkBreak;
...
}
```

List. 4.8. Kod modelu/klasy „AppViewModel”

Ważnym elementem w powyżej omawianej klasie jest asynchroniczna metoda "createDevice", to właśnie dzięki niej nawiązywane jest połączenie z urządzeniem o danym adresie IP oraz adresie PORT, a następnie pobranie informacji o urządzeniu poprzez użycie komendy "*IDN?". Ze względu na fakt, iż wykorzystane w tym projekcie są połączenia TCP, wymagane jest stosowanie kodu asynchronicznego. Należy zaczekać na odebrane dane w celu ich dalszego modyfikowania. Zapobiega to wielu błędom. Kod tej metody przedstawiono na listingu 4.9. Na początku inicjowane są domyślne wartości dla zmiennych. Sprawdzane jest czy podany adres IP jest poprawny oraz czy urządzenie o takim adresie nie znajduje się już na liście urządzeń (devices). Tworzone jest połączenie z wykorzystaniem wbudowanego obiektu klasy Socket z ograniczonym czasem na odpowiedź. Socket łączy gniazdo lokalne z gniazdem zdalnym. Dane, jako „Uint8List”, są odbierane przez lokalne gniazdo, udostępniane przez interfejs „Stream” tej klasy i mogą być wysyłane do zdalnego gniazda przez interfejs „IOSink” tej klasy. Po ustanowieniu połączenia załączana jest dla niego metoda "listen" nasłuchująca strumień danych odebranych od urządzenia. Wysyłane jest polecenie "*IDN?". W tym momencie programu wystarczy "jedynie" nawiązać połączenie z urządzeniem, wysłać polecenie i odebrać dane, i połączenie należy zakończyć. Z pomocą w realizacji tego problemu przyszedł obiekt klasy Completer. Pozwala on tworzyć i zarządzać wynikami asynchronicznych metod typu "Future". Taki też completer został utworzony na początku metody. Po wysłaniu polecenia oczekujemy na zakończenie complitera, które to występuje gdy pojawią się dane w strumieniu. Dzięki temu wiadomo, że otrzymano dane połączenie z urządzeniem może być zakończone. Dodatkowo zaimplementowano czasomierz odliczający czas na odpowiedź, aby nie oczekiwano na nią w nieskończoność. Dane wysyłane muszą mieć postać binarną, dlatego też należy jest zakodować. Odebrane dane także mają postać binarną.

Jedyną interesującą w tej chwili odebraną wartością jest ciąg znaków zawierający informacje o urządzeniu odseparowanych przecinkiem. Przypisujemy te dane do wcześniej utworzonych zmiennych. W przypadku braku lub błędu otrzymania informacji o urządzeniu przypisane zostaną mu pierwotne wartości zainicjowane na początku metody. Na końcu z gotowych wartości zmiennych tworzony jest obiekt "Device" i dodawany do listy urządzeń "devices". Nazwę urządzenia oraz typ będzie można edytować w zakładce urządzenia. Na zakończenie powiadamiany jest interfejs (View), który oczekuje na zmiany w modelu wyglądu (View Model), poprzez polecenie "notifyListeners".

```
Future<String> createDevice(String ip, int port) async {
    SocketConnection socketConnection = SocketConnection(ip, port);
    Completer completer = Completer();
    String name = "Unknown";
    String manufacturer = "Unknown";
    String model = "Unknown";
    String status = "Offline";
    String serial = "Unknown";
    String type = "Unknown";
    Socket socket;
    try {
      if (!isValidHost(ip)) {
        return "Zly format IP!";
      } else if (!comparedByIP(ip)) {
        return "Urzadzenie znajduje sie juz na liscie.";
      }
      socket =
          await Socket.connect(ip, port, timeout: const Duration(seconds:
3));
      socket.listen((List<int> event) async {
        List<String> message = utf8.decode(event).split(',');
        if (message.isNotEmpty) {
          manufacturer = message.elementAt(0).trim();
          model = message.elementAt(1).trim();
          type = detectDeviceType(model).toString();
          name = "$type $model";
          serial = message.elementAt(2).trim();
        }
        status = "dostepny";
        completer.complete(event);
        completer = Completer();
      });
      socket.add(utf8.encode('*IDN?\n'));
      final timeoutTimer = Timer(const Duration(seconds: 4), () {
        completer.complete();
      });
      await completer.future;
      timeoutTimer.cancel();
      socket.close();
    } catch (ex) {
      return "Nie udalo sie nawiazac polaczenia z urzadzeniem!";
    }
     Device device = Device(key: UniqueKey(), name: name, type: type, ip: ip,
port: port, manufacturer: manufacturer, model: model, serial: serial, status:
status, connection: socketConnection);
    devices.add(device);
    notifyListeners();
    return "Nawiazano polaczenie z urzadzeniem!";
  }
```

List. 4.9. Kod metody „createDevice”

Należy w tym miejscu omówić klasę SocketConnection. Jak już wspomniano wykorzystano fragmenty kodu z pakietu "tcp_socket_connection". Do tego dodano swoje metody pełniące kluczowe funkcje. Na listingu 4.10 pokazano fragment tej klasy. Asynchroniczna metoda "connect" przyjmująca za argumenty czas na odpowiedź, funkcję zwrotną obsługującą odebraną wiadomość oraz niewymagany parametr liczbę prób połączenia.

```
Future<void> connect(int timeOut, Function callback,{int attempts = 1}) async
{
    int k = 1;
    while (k <= attempts) {
      try {
        _server = await Socket.connect(_ipAddress, _portAddress,
            timeout: Duration(milliseconds: timeOut));
        _connected = true;
        _printData("Socket successfully connected");
        _server!.listen(
          (List<int> event) {
            String received = (utf8.decode(event));
            callback(received);
          },
          cancelOnError: true,
          onError: (e) {
            debugPrint("EXCEPTION ERROR ");
          },
        );
        break;
      } catch (ex) {
        _printData("$k attempt: Socket not connected (Timeout reached)");
        if (k == attempts) {
          return;
        }
      }
      k++;
    }
  }

void disconnect() {
    if (_server != null) {
      try {
        _server!.close();
        _printData("Socket disconnected successfully");
      } catch (exception) {
      }
    }
    _connected = false;
  }

void sendMessageEOM(String message, String eom) {
    try {
      if (_server != null && _connected) {
        _server!.add(utf8.encode(message + eom));
      }
    } catch (e) {
      _printData("ERROR: $e");
    }
  }
```

List. 4.10. Kod metod „connect”, „disconnect” oraz „sendMessageEOM”

Na listingu 4.11 pokazano kolejny fragment klasy SocketConnection, a właściwie kluczową metodę wysyłającą zadeklarowane polecenia dla danego kanału danego urządzenia. Po odebraniu wiadomości jest ona przypisywana do listy list zmiennych typu "String" tj. list kanałów. Na końcu lista ta jest zwracana. Przedstawiono jedynie przykładową metodę dla urządzenia typu oscyloskop ze względu na podobną budowę dla każdego z urządzeń.

```
Future<List<List<String>>> getOscilloscopeValues(
      Oscilloscope oscilloscope) async {
    List<List<String>> oscilloscopeChannelsRawResponses = [[], [], [], []];
    int i = 0;
    try {
      for (OscilloscopeChannel channel in oscilloscope.channels) {
        for (Command command in channel.commands) {
          if (command.type == "READ") {
            sendMessageEOM(command.query, '\n');
            final timeoutTimer = Timer(const Duration(milliseconds: 1500), ()
{
              completer.complete();
              message = "-";
            });
            await completer.future;
            timeoutTimer.cancel();
            completer = Completer();
            oscilloscopeChannelsRawResponses[i].add(message);
          }
        }
        i++;
      }
      return oscilloscopeChannelsRawResponses;
    } catch (exception) {
      debugPrint("$exception");
      return List.empty();
    }
  }
```

List. 4.11. Kod metody „getOscilloscopeValues" – dla przykładu urządzenie oscyloskop

Po pomyślnym dodaniu urządzenia do listy urządzeń, należy go dodać do listy urządzeń danego stanowiska. Umożliwia to metoda asynchroniczna "addDeviceToStation" (List. 4.12). Ze względu na przejrzystość pracy usunięto fragment kodu dla zasilacza. Jako argumenty przyjmuje ona indeks docelowego stanowiska oraz obiekt dodawanego urządzenia. Metoda ta, po wcześniejszym sprawdzeniu czy numer serial urządzenia znajduje się już w liście urządzeń stanowiska, dodaje obiekt klasy adekwatny do typu podanego obiektu "Device". Po utworzeniu danego obiektu urządzenia z wartości pierwotnego obiektu nawiązywane jest połączenie, a następnie nowo utworzony obiekt dodawany jest do listy urządzeń danego stanowiska. Stanowiska oraz urządzenia w stanowisku mogą być zamieniane miejscami poprzez przytrzymanie danego kontenera i przeniesienie go w oczekiwane miejsce. Usunięcie zaimplementowano poprzez przesunięcie danego elementu od prawej do lewej.

```
Future<void> addDeviceToStation(int indexStation, Device device) async {
    if (comparedBySerialInStations(device)) {
      switch (device.type) {
        case "Multimetr":
          Multimeter multimeter = Multimeter(
              key: UniqueKey(),
              name: device.name,
              displayON: true,
              ip: device.ip,
              port: device.port,
              manufacturer: device.manufacturer,
              model: device.model,
              serial: device.serial,
              status: device.status,
              connection: device.connection);
          stations[indexStation].devices.add(multimeter);
          await multimeter.connection.startConnection();
          break;
        case "Generator":
          Generator generator = Generator(
              key: UniqueKey(),
              name: device.name,
              displayON: true,
              ip: device.ip,
              port: device.port,
              manufacturer: device.manufacturer,
              model: device.model,
              serial: device.serial,
              status: device.status,
              connection: device.connection);
          stations[indexStation].devices.add(generator);
          await generator.connection.startConnection();
          break;
        case "Oscyloskop":
          Oscilloscope oscilloscope = Oscilloscope(
              key: UniqueKey(),
              name: device.name,
              displayON: true,
              ip: device.ip,
              port: device.port,
              manufacturer: device.manufacturer,
              model: device.model,
              serial: device.serial,
              status: device.status,
              connection: device.connection);
          stations[indexStation].devices.add(oscilloscope);
          await oscilloscope.connection.startConnection();
          break;
        default:
      }
      notifyListeners();
    }
}
```

List. 4.12. Kod metody „addDeviceToStation”

Kolejnym etapem po dodaniu urządzenia do stanowiska jest przeprowadzanie pomiarów. Przycisk "Pomiar" umożliwia wykonaniu pomiaru jednorazowego oraz ciągłego. Na listingu 4.13 przedstawiono metody odpowiadające za wykonanie tych instrukcji.

Metoda "measureContinously" wykonuje pomiary w sposób ciągły dla każdego urządzenie w każdym stanowisku, aż do zatrzymania przyciskiem "Pomiar". To za sprawą metody "refreshDeviceValues", do której przekazywany jest obiekt danego urządzenia. Dodatkowo pobierana jest ustawiona wartość opóźnienia z ustawień w klasie "SettingsViewModel". Po zakończeniu wykonywania pomiaru do każdego z urządzeń wysyłane jest polecenie "SYSTem:LOCal", które zmienia tryb pracy urządzenia ze zdalnego do lokalnego odblokowując panel. Natomiast metoda "measureOnce" przeprowadza powyższe operacje jednorazowo bez żadnego opóźnienia.

```
void measureContinously() async {
    SettingsViewModel settingsViewModel = getSettingsViewModel();
    while (!isStopped) {
      for (Station station in stations) {
        for (var device in station.devices) {
          if (isStopped) break;
          await refreshDeviceValues(device);
          notifyListeners();
        }
      }
      //delay
      await Future.delayed(Duration(milliseconds:
settingsViewModel.timeout));
    }
    if (isStopped) {
      for (Station station in stations) {
        for (var device in station.devices) {
          device.connection.sendMessageEOM("SYSTem:LOCal", '\n');
        }
      }
    }
  }

  void measureOnce() async {
    for (Station station in stations) {
      for (var device in station.devices) {
        await refreshDeviceValues(device);
        notifyListeners();
        device.connection.sendMessageEOM("SYSTem:LOCal", '\n');
      }
    }
  }
```

List. 4.13. Kod metod „measureContinously" oraz „measureOnce"

Na poniższym listingu 4.14 zawarto metodę "refreshDeviceValues" wykorzystaną w powyższym listingu 4.13. Ze względu na przejszystość pracy nie zamieszczono fragmentu kodu dla zasilacza. Dla danego typu urządzenia pobiera ona do listy list wartości z metody klasy SocketConnection (np. "getMultimeterValues"). Po pobraniu listy z wartościami są one przypisywane bezpośrednio do danego kanału danego obiektu urządzenia.

```
Future<void> refreshDeviceValues(var device) async {
  switch (device.toString()) {
    case "Multimeter":
      try {
        List<List<String>> rawChannelsValues = await
device.connection.getMultimeterValues(device);
        int i = 0;
        for (MultimeterChannel channel in device.channels) {
          channel.unit = rawChannelsValues[i].elementAt(0).trim();
          channel.value = rawChannelsValues[i].elementAt(1).trim();
          i++;
        }
      } catch (e) {
      }
      break;
    case "Generator":
      try {
        List<List<String>> rawChannelsValues = await
device.connection.getGeneratorValues(device);
        int i = 0;
        for (GeneratorChannel channel in device.channels) {
          channel.function = rawChannelsValues[i].elementAt(0).trim();
          channel.voltageValue = rawChannelsValues[i].elementAt(1).trim();
          channel.voltageOffset = rawChannelsValues[i].elementAt(2).trim();
          channel.frequencyValue = rawChannelsValues[i].elementAt(3).trim();
          i++;
        }
      } catch (e) {
      }
      break;
    case "Oscilloscope":
      try {
        List<List<String>> rawChannelsValues = await
device.connection.getOscilloscopeValues(device);
        int i = 0;
        for (OscilloscopeChannel channel in device.channels) {
          channel.vpp = rawChannelsValues[i].elementAt(0).trim();
          channel.vrms = rawChannelsValues[i].elementAt(1).trim();
          channel.vaverage = rawChannelsValues[i].elementAt(2).trim();
          channel.frequency = rawChannelsValues[i].elementAt(3).trim();
          i++;
        }
      } catch (e) {
      }
      break;
    default:
  }
}
```

List. 4.14. Kod metody „refreshDeviceValues”

W ostatnim listingu 4.15 zawarto kod metody asynchronicznej "findDevicesInNetwork" typu "Future<String>". Pobiera ona wartości z klasy "SettingsViewModel" do lokalnych pól: adres sieci, adres urządzenia i adres rozgłoszeniowy (ostatni). Następnie, jeśli jest połączenie z siecią Wi-Fi, dla każdego adresu w sieci lokalnej nie będącego na liście urządzeń, aż do osiągnięcia adresu rozgłoszeniowego, sprawdzane jest czy możliwe jest nawiązanie połączenia. Dalej tworzone jest urządzenie metodą "createDevice".

Na końcu inkremetujemy aktualny adres IP. Metoda zwiększająca adres IP "incrementIP" przyjmuje argument typu String tj. adres IP. Jest on rozdzielany na 4 oktety, które następnie zamienione są na liczby dziesiętne typu int. Dalej następuje odpowiednie przesunięcie bitowe dla każdego z oktetów oraz wykonywana operacja OR. Następnie otrzymaną wartość inkrementujemy dodając 1 i zwracamy zamieniony ponownie na ciąg znaków zwiększony adres IP.

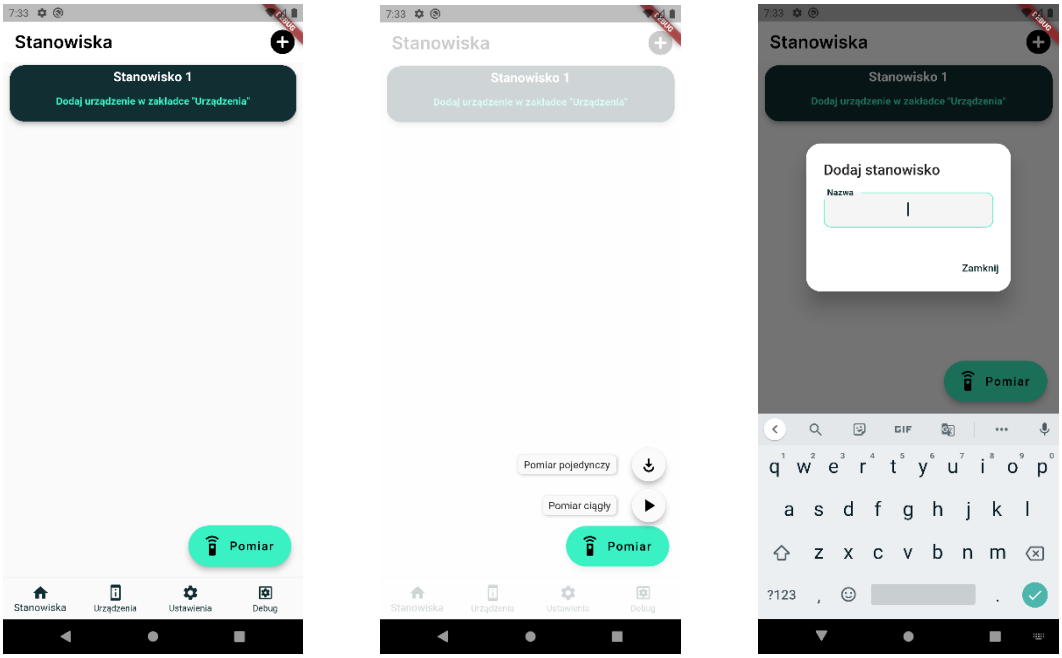
```
Future<String> findDevicesInNetwork() async {
    findDevicesInNetworkBreak = false;
    int newCount = 0;
    SettingsViewModel settingsViewModel = getSettingsViewModel();
    String? networkIP = settingsViewModel.ipRange;
    String deviceIP = networkIP;
    String? broadcast = settingsViewModel.broadcast;
    if (await checkConnectivityToWifi()) {
      while (deviceIP != broadcast) {
        if (!findDevicesInNetworkBreak) {
          try {
            if (comparedByIP(deviceIP)) {
              SocketConnection socketConnection =
                  SocketConnection(deviceIP, 5025);
              if (await socketConnection.canConnect(1000)) {
                await createDevice(deviceIP, 5025);
                newCount++;
              }
            }
            deviceIP = incrementIP(deviceIP);
          } catch (e) {
            debugPrint(e.toString());
          }
        } else {
          break;
        }
      }
    } else {
      return "Brak połączenia z siecią Wi-Fi! Połacz się z siecią i pobierz
jej adres w zakładce Ustawienia.";
    }
    return 'Zakończono skanowanie. Dodano $newCount nowe urządzenia.';
  }

String incrementIP(String input) {
  List<String> inputIPString = input.split(".");
  List<int> inputIP = inputIPString.map((e) => int.parse(e)).toList();
  var ip = (inputIP[0] << 24) |
      (inputIP[1] << 16) |
      (inputIP[2] << 8) |
      (inputIP[3] << 0);
  ip++;
  return "${ip >> 24 & 0xff}.${ip >> 16 & 0xff}.${ip >> 8 & 0xff}.${ip >> 0 &
0xff}"; //0xff = 255
}
```

List. 4.15. Kod metod „findDevicesInNetwork” oraz „incrementIP”

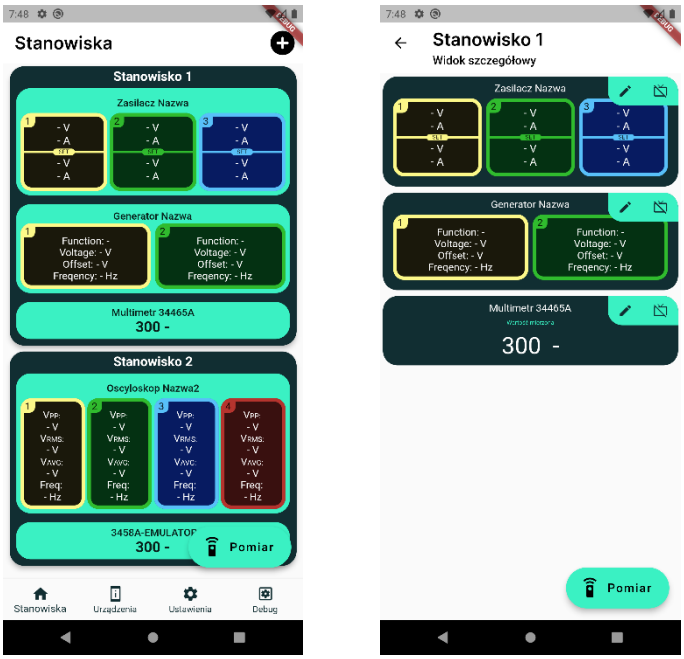
4.2. Interfejs

Starano się jak najdokładniej odwzorować zaprojektowany interfejs aplikacji. Drobne różnice mogą wynikać z kompromisów pomiędzy wyglądem a funkcjonalnością aplikacji. Na rysunku 4.1 przedstawiono widoki zakładki „Stanowiska” na emulowanym urządzeniu Android.



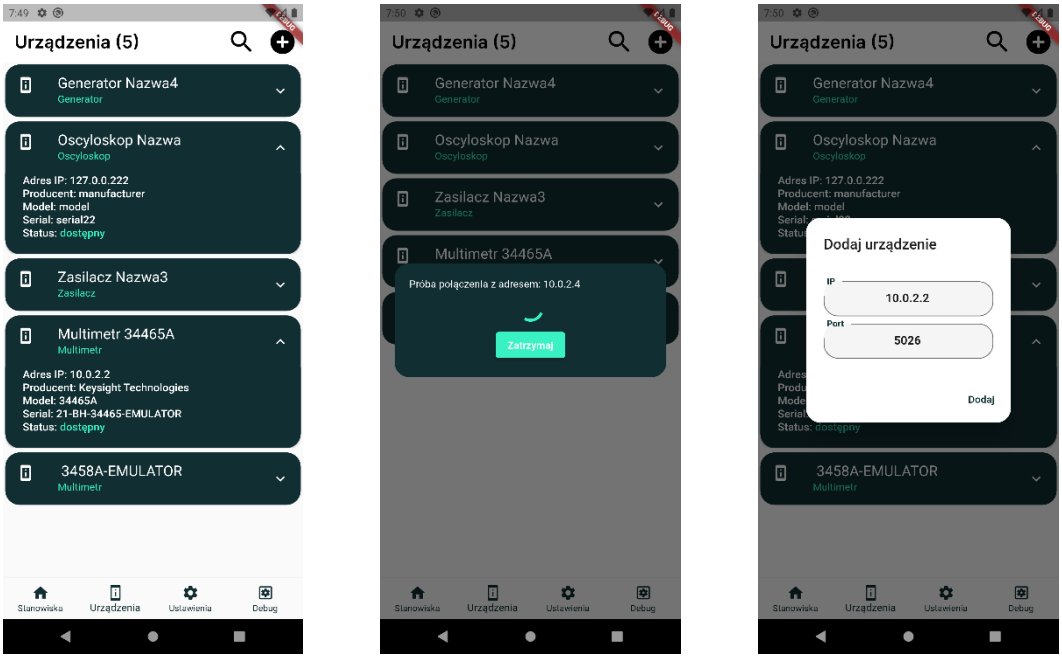
Rys. 4.1. Widok zakładki „Stanowiska” bez urządzeń

Na rysunku 4.2 pokazano widok zakładki „Stanowiska” z dodanymi urządzeniami.



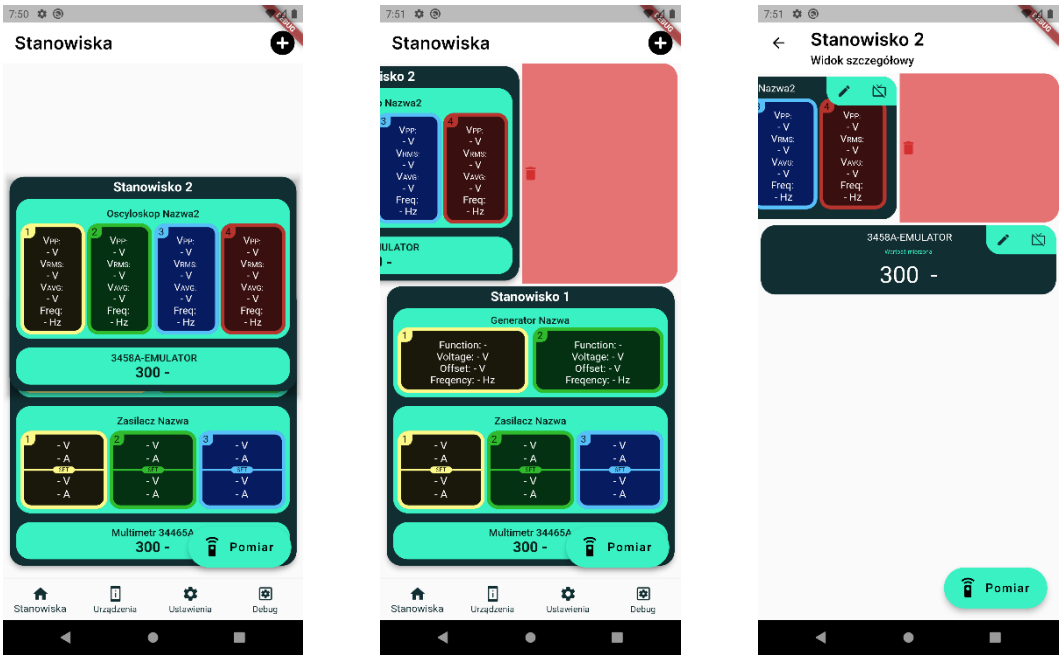
Rys. 4.2. Widok zakładki „Stanowiska” z urządzeniami oraz widok szczegółowy stanowiska

Na rysunku 4.3 ukazano widok zakładki „Urządzenia” z dodanymi urządzeniami oraz okna dialogowe przycisków nawigacji.



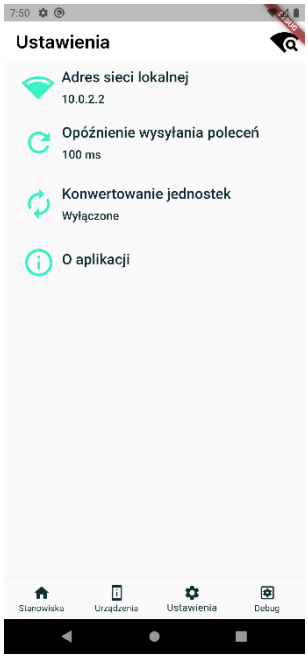
Rys.4.3. Widok zakładki „Urządzenia”

Na rysunku 4.4 przedstawiono zaprogramowane gesty usprawniające przemieszczenie stanowisk i urządzeń w stanowisku.



Rys. 4.4. Działanie gestów

Na rysunku 4.5 zamieszczono widok zakładki „Ustawienia”.



Rys. 4.5. Widok zakładki „Ustawienia”

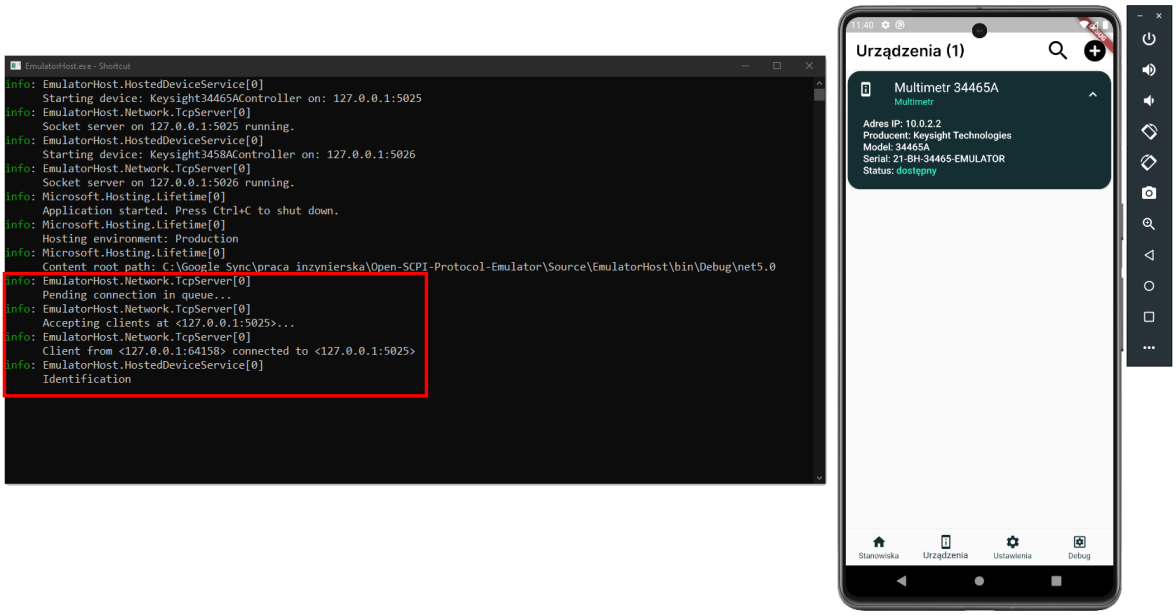
5. Testowanie

5.1. Symulowane urządzenia pomiarowe

Głównym problemem podczas wykonywania tego projektu było debugowanie i sprawdzanie kodu pod względem działania z fizycznym, rzeczywistym urządzeniem pomiarowym. Urządzenia te nie cechują się łatwą dostępnością do użytku domowego ze względu na swoją cenę. Dlatego też poszukiwano czegoś w rodzaju emulatora urządzenia pomiarowego.

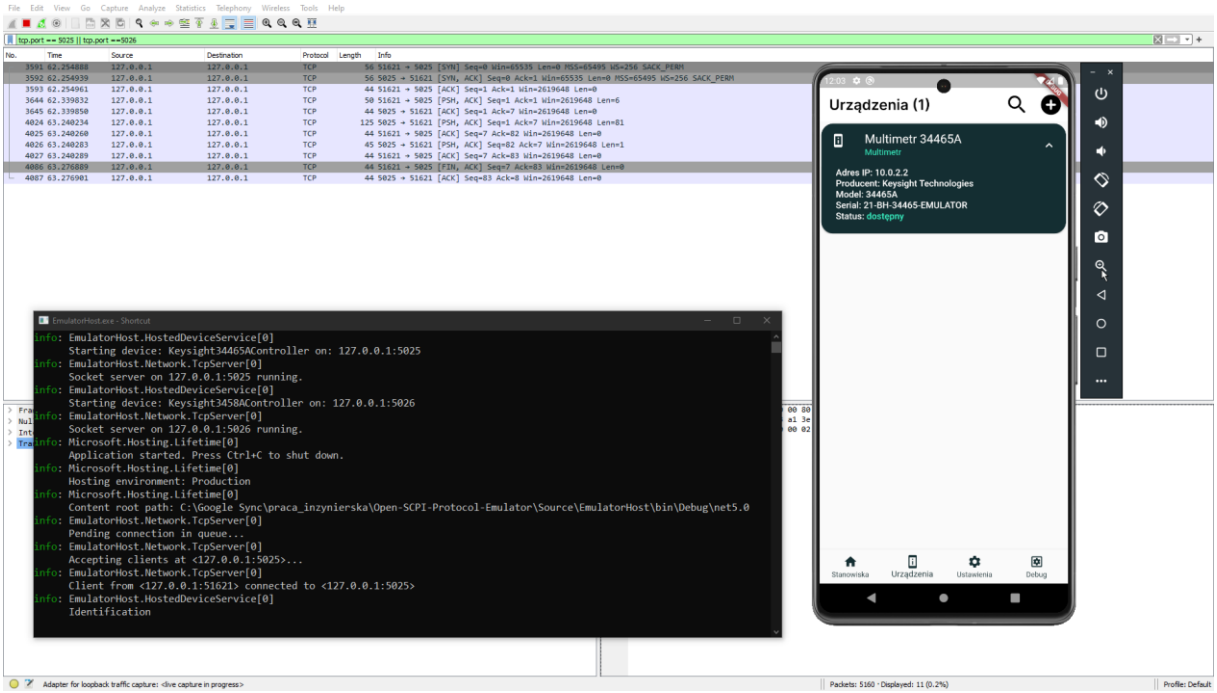
Udało się znaleźć repozytorium spełniające w pewnym stopniu wymagania. Repozytorium Open-SCPI-Protocol-Emulator użyto do emulacji urządzeń pomiarowych Keysight poprzez implementację protokołu SCPI przy użyciu połączenia TCP. Umożliwiło to testy integracyjne w potokach ciągłej integracji i umożliwiło pisanie oprogramowania dla tych urządzeń bez posiadania drogiego urządzenia. Program działa w formie konsolowej. Domyślnie wbudowane zostały dwa urządzenia typu multimetr. Obsługiwały one niektóre z poleceń SCPI jak: *IDN?, READ?, ABORt, CONFigure:CURRent:AC, CONFigure:CURRent:DC, MEASure:CURRent:AC?, MEASure:CURRent:DC?, CONFigure:VOLTage:AC, CONFigure:VOLTage:DC, MEASure:VOLTage:AC?, MEASure:VOLTage:DC?, DISPlay:TEXT, DISPlay:TEXT:CLEar, SENSe:VOLTage:IMPedance:AUTO. Pomimo braku wszystkich instrukcji, obecne w dużym stopniu pozwoliły na zaprogramowanie modelu komunikacji z urządzeniem [15].

Urządzenia emulowane przez ten program działały na adresie pętli zwrotnej (localhost:127.0.0.1) komputera na portach 5025 oraz 5026. Natomiast emulowane urządzenie Android do komunikacji z pętlą zwrotną komputera wykorzystuje adres 10.0.2.2, ponieważ adres 127.0.0.1 oznacza adres pętli zwrotnej samego emulatora urządzenia Android, co jest kluczową informacją. Na poniższym rysunku 5.1 pokazano zrzut programu emulującego urządzenia pomiarowe. Aplikację przetestowano dla emulatorów z wersją Android 10.0 oraz najnowszą Android 14.0.



Rys. 5.1. Widok konsoli z działającym programem emulującym urządzenia pomiarowe

Proces połączenia zdecydowano się podejrzeć w programie WireShark. Wynik przedstawiono na poniższym rysunku 5.2. Połączenie jest bez problemu nawiązywane i wymieniane są informacje.



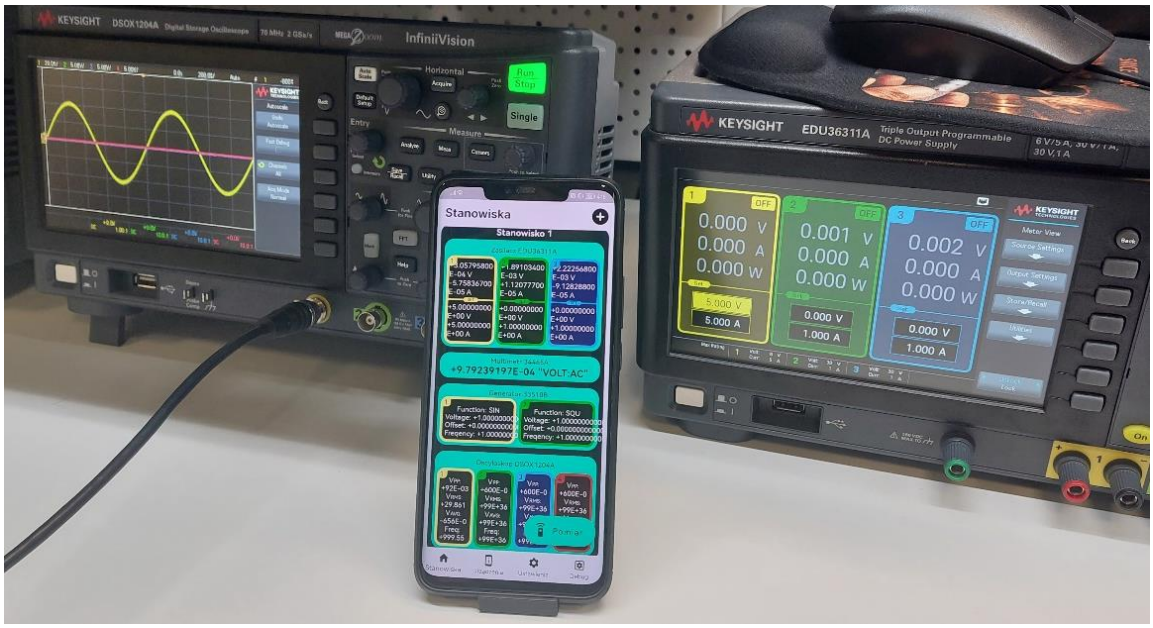
Rys. 5.2. Widok pomyślnego nawiązania połączenia w programie WireShark

5.2. Rzeczywiste urządzenia pomiarowe

Symulacja nie odzwierciedla rzeczywistości. Dlatego też przystąpiono do sprawdzenia działania aplikacji w warunkach laboratoryjnych z fizyczymi urządzeniami na stanowisku przedstawionym na rysunku 5.3. Schemat połączeniowy wykonano zgodnie ze schematem przedstawionym na rysunku 1.1. Urządzenia kablem Ethernet połączono do przełącznika (ang. switch). Przełącznik podłączono do trasownika (ang. router) z włączonym serwerem DHCP. Przed uruchomieniem aplikacji, na smartfonie z wersją Android 10.0, ustanowiono połączenie z siecią Wi-Fi. Aplikacja pomyślnie pobiera jej adres aktualnie połączonej sieci Wi-Fi. Nawiązywane jest połączenie z urządzeniami. Zostają pobrane informacje o urządzeniu i wykrywany jest typ. Po pogrupowaniu urządzeń do stanowisk, pomyślnie wykonywane są pomiary i odczytanie wartości mierzonych przez urządzenia.

Rys. 5.3. Stanowisko testowe

Niestety z powodu błędnej implementacji opcji formatującej otrzymane dane, po jej włączeniu, nie są one prezentowane w przejrzystym formacie. Na rysunku 5.3 oraz 5.4 ustawienie to jest wyłączone.



Rys. 5.4. Widok aplikacji z pomyślnie odczytanymi wartościami

Na rysunku 5.4 zawarto widok aplikacji ze zmierzonymi wartościami. Zauważalny jest m.in. pomiar wartości częstotliwości przebiegu podanego na kanał 1 oscyloskopu przez kanał 1 generatora oraz wartość napięcia i prądu zasilania kanału 1 zasilacza.

6. Podsumowanie

Celem projektu inżynierskiego było stworzenie aplikacji mobilnej, która oferuje możliwość zdalnego monitoringu urządzeń pomiarowych. Proces tworzenia aplikacji składał się z etapów:

1) projekt graficzny i funkcjonalny aplikacji,
2) programowanie aplikacji,
3) testowanie aplikacji.

Napisano aplikację w technologii wieloplatformowej Flutter na system Android. Dzięki temu, po niedużej modyfikacji, możliwe jest działanie tej aplikacji na systemach iOS czy Windows. Główna funkcjonalność aplikacji została osiągnięta. Aplikacja wykrywa działające w sieci cyfrowe urządzenia producenta Keysight typu multimetry, oscyloskopy serii InfiniiVision, zasilacze, generatory przebiegów i funkcji. Nawiązuje z nimi niezależne połączenia, pobiera informacje, wykonuje pomiary i prezentuje otrzymane zmierzone wartości. Wszystko to wykorzystując metody asynchroniczne zapobiegające wystąpieniu błędów czy zapętlenia wysyłania i odbierania poleceń. Program prezentuje stanowiska pogrupowane w jednym widoku oraz pojedyncze stanowisko w widoku szczegółowym. Możliwa jest edycja urządzeń w aplikacji poprzez zmianę ich nazwy oraz typu. Stanowiska oraz urządzenia w stanowisku mogą być usunięte poprzez wykonanie gestu przesunięcia elementu. To samo dotyczy kolejności stanowisk oraz urządzeń w stanowiskach, które może być zmieniana wykorzystując gest przytrzymania elementu.

Z powodu ograniczonego i skróconego czasu na wykonanie projektu nie wykonano szeregu usprawnień, optymalizacji kodu, funkcjonalności, ustawień dostępności czy wyglądu interfejsu. Występuje problem z obróbką i wyświetleniem otrzymywanych danych w kompaktowy i przejrzysty sposób.

W efekcie końcowym uzyskano działający produkt, spełniający założenia projektowe. Stworzona aplikacja stanowi innowacyjne i skuteczne narzędzie, które znacząco ułatwia monitorowanie oraz analizę danych pomiarowych. Dzięki swojej funkcjonalności i intuicyjnemu interfejsowi użytkownika, umożliwia precyzyjny odczyt i rejestrację pomiarów, eliminując przy tym konieczność bezpośredniego dostępu do urządzeń pomiarowych.

Wykonanie tego projektu znacząco wpłynęło na umiejętność analitycznego myślenia, umiejętność projektowania graficznego oraz programowania aplikacji moblinych z wymianą danych.

6.1. Plany rozwoju

W aplikacji drzemie potencjał. Będzie ona dalej rozwijana. Oto propozycje oraz cele do osiągnięcia w najbliższym czasie:

1) optymalizacja kodu – usprawnienie asynchronicznej metody pobierania wartości tak, aby niezależność pobierania informacji między każdym urządzeniem była jeszcze większa,
2) usprawnienie komunikacji z urządzeniem,
3) poprawiona obsługa wyjątków i błędów możliwych do wystąpienia podczas działania aplikacji,
4) poprawa sposobu formatowania otrzymywanych danych,
5) poprawa responsywności interfejsu użytkownika (większe ekrany),
6) przechowywanie informacji o urządzeniach i ustawień aplikacji lokalnie na urządzeniu bądź na serwerze zdalnym
7) język angielski aplikacji,
8) ciemny motyw aplikacji.

Autor za wkład własny pracy uważa:

1) przegląd rozwiązań,
2) zapoznanie z dostępną literaturą i dokumentacjami,
3) projekt graficzny i funkcjonalny aplikacji,
4) zaprogramowanie aplikacji,
5) przeprowadzenie testów aplikacji,
6) opracowanie i sformułowanie wniosków.

LITERATURA

[1] Gilski, Przemyslaw, and Jacek Stefanski. "Android os: A review." Tem Journal 4.1 (2015): 116.

[2] https://opensource.org/osd/. Dostęp 15.10.2023.

[3] https://code.visualstudio.com/docs. Dostęp 15.10.2023.

[4] https://developer.android.com/studio/run/emulator. Dostęp 15.10.2023.

[5] Lou, Tian. "A comparison of Android native app architecture MVC, MVP and MVVM." Eindhoven University of Technology (2016).

[6] Bracha, Gilad. The Dart programming language. Addison-Wesley Professional, 2015.

[7] Rus, Mateusz. "Tworzenie nowoczesnych aplikacji wieloplatformowych w technologii Flutter i Dart." (2023).

[8] Scrimger, Rob, et al. "TCP/IP a Bíblia." Rio de Janeiro: Campus (2002).

[9] https://www.wireshark.org/docs/wsug_html_chunked/. Dostęp 17.10.2023.

[10] https://www.keysight.com/zz/en/assets/9921-01870/miscellaneous/SCPI-99.pdf. Dostęp 10.12.2023.

[11] https://help.figma.com/hc/en-us. Dostęp 17.10.2023.

[12] Niezgoda, Małgorzata Agata. Material Design-examining the quality of graphical interface. Diss. Instytut Telekomunikacji, 2016.

[13] Blischak, John D., Emily R. Davenport, and Greg Wilson. "A quick introduction to version control with Git and GitHub." PLoS computational biology 12.1 (2016): e1004668.

[14] Miyashita, Yutsuki, et al. "Design of the inspection process using the GitHub flow in project based learning for software engineering and its practice." arXiv preprint arXiv:2002.02056 (2020).

[15] https://github.com/bluehands/Open-SCPI-Protocol-Emulator/. Dostęp 11.06.2023.

Sygnatura:

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza Rzeszów, 2023
Wydział Elektrotechniki i Informatyki

STRESZCZENIE PROJEKTU INŻYNIERSKIEGO

ZDALNY MONITORING PRZYRZĄDÓW POMIAROWYCH

Autor: Karol Bulanowski, nr albumu: ET-DI-167684
Opiekun: dr inż. Jakub Wojturski
Słowa kluczowe: pomiary, scpi, flutter, android, zdalne

Tematem projektu inżynierskiego jest stworzenie aplikacji umożliwiającej nawiązywanie połączenia i komunikację z urządzeniami pomiarowymi w celu pobierania oraz wyświetlania aktualnie mierzonych i nastawionych wartości. Przedstawiono cały proces tworzenia programu: od projektu do kodu.

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, 2023
Faculty of Electrical and Computer Engineering

DIPLOMA THESIS (BS) ABSTRACT

REMOTE MONITORING OF MEASURING INTRUMENTS

Author: Karol Bulanowski, code: ET-DI -167684
Supervisor: Jakub Wojturski, PhD, Eng.
Key words: measurements, scpi, flutter, android, remote

The topic of this engineering project is the development of an application that allows connection and communication with measuring devices to retrieve and display of currently measured and set values. The entire process of creating the program is presented from design to coding.