



**A G H**

**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,  
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Rejestrator Stacji Bazowych  
BTS Tracker*

Autor:

*Filip Biernat*

Kierunek studiów:

*Automatyka i Robotyka*

Opiekun pracy:

*dr hab. Adam Sędziwy*

Kraków, 2018

*Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpozna bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godność studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*





## **Spis treści**

<b>1. Wprowadzenie .....</b>	7
<b>2. Zarys i analiza problemu.....</b>	9
2.1. Dynamiczny rozwój technologii mobilnych.....	9
2.2. Złożoność sieci komórkowych .....	9
2.3. Rejestrator stacji bazowych .....	10
<b>3. Propozycja rozwiązania.....</b>	13
3.1. Struktura aplikacji .....	13
3.1.1. Wykrywanie stacji bazowych.....	14
3.1.2. Przetwarzanie zebranych informacji .....	15
3.1.3. Przechowywanie danych .....	18
3.2. Graficzny interfejs użytkownika.....	20
3.2.1. Klasa MainActivity .....	20
3.2.2. Klasa FloatingActionButtonOnClickListener.....	22
3.2.3. Klasa PermissionsManager .....	23
3.2.4. Klasy TaskManager i FileManager .....	24
3.2.5. Klasa ListFragment i pakiet list .....	25
3.2.6. Klasa MapFragment.....	28
3.2.7. Klasa TutorialFragment.....	30
<b>4. Rezultaty testów .....</b>	33
4.1. Scenariusze testowe .....	33
4.1.1. Wielogodzinne działanie w tle .....	33
4.1.2. Poruszanie się po mieście .....	34
4.1.3. Przejazd autostradą .....	35
4.1.4. Przemieszczanie się wewnątrz budynku .....	37
4.1.5. Spacer po pagórkowatym terenie .....	38
4.2. Przypadki testowe.....	39
4.2.1. Więcej niż jeden BTS w danej lokalizacji .....	39
4.2.2. Zapis do pliku i odczyt zapisanych informacji .....	41
4.2.3. Wykorzystanie komputera do analizy danych.....	43
4.3. Przypadki szczególne .....	45
4.3.1. Brak informacji o stacji w tablicy danych.....	45

4.3.2. Niepoprawne informacje w pliku wejściowym.....	45
4.4. Szczegóły techniczne.....	46
<b>5. Podsumowanie i wnioski.....</b>	<b>49</b>

# **1. Wprowadzenie**

Celem niniejszej pracy jest zaprojektowanie i realizacja aplikacji mobilnej stworzonej z myślą o użytkownikach telefonów komórkowych. Jej zadaniem jest ciągłe rejestrowanie zmian stacji bazowych, do których urządzenie jest podłączane oraz wizualizacja tych danych.

Głównym założeniem jest dostarczenie funkcjonalnej, estetycznej (zgodnie z Material Design), otwartoźródłowej aplikacji zestawiającej dane pobrane z urządzenia z rekordami prostej bazy danych i przedstawienie ich w formie zarówno tekstowej jak i geograficznej, wykorzystując usługę Google Maps. Oprogramowanie może działać w tle przez dłuższy czas przy bardzo niskim zużyciu energii. Pozwala na bieżącą analizę sygnału z komórki oraz na wizualizację danych dla większego przedziału czasu.

„BTS Tracker” został zaimplementowany w języku Java dla systemu Android. Do stworzenia aplikacji wykorzystano zintegrowane środowisko programistyczne Android Studio.

Rejestrator umożliwia odczytywanie lokalizacji, miejscowości, współrzędnych geograficznych, technologii oraz nazwy operatora sieci. Dodatkowo zaimplementowany mechanizm zegarowy mierzy czas połączenia do konkretnej stacji przekaźnikowej.

Aplikacja wspiera zapis i odczyt z pliku, a także eksport do formatu KML, dzięki któremu możliwy jest odczyt zebranych danych na komputerze.



## **2. Zarys i analiza problemu**

### **2.1. Dynamiczny rozwój technologii mobilnych**

Choć od wiosny roku 1973, kiedy Motorola zaprezentowała swój pierwszy telefon komórkowy [1], do chwili obecnej minęło stosunkowo niewiele lat, ciężko wyobrazić sobie przyszłość bez komunikacji bezprzewodowej. Wykorzystujące ją telefony potrafią już nie tylko zestawiać połączenia głosowe, ale także przesyłać dane z prędkością rzędu setek megabitów na sekundę.

Co więcej nic nie wskazuje na to, żeby ten rozwój technologiczny miał się zatrzymać. Opracowane właśnie oprogramowanie eMTC, NB-IoT oraz Cat-M pozwoli stworzyć niskobudżetowy Internet Rzeczy z prawdziwego zdarzenia [2], co raz na zawsze zmieni świat, jaki znamy, pozwalając komunikować się ze sobą naszym lodówkom, pralkom, czy szczoteczkom do zębów. Równolegle powstaje sieć piątej generacji – sieć ogromnych prędkości, bardzo niskich opóźnień i niezwykłej przepustowości, o infrastrukturze gotowej połączyć setki miliardów użytkowników.

Co ciekawe, w telefonii komórkowej kolejne generacje nie wypierają poprzednich. Niejako obudowują istniejący system dołączając tym samym dodatkowe rozwiązania do już istniejących. Oglądając film w dużym mieście w wysokiej jakości 4K wykorzystamy Internet LTE. Natomiast będąc na wsi czy w górach, gdzie zasięg telefonów czwartej generacji jest nieosiągalny, często musimy zadowolić się połączeniem GSM albo WCDMA. Pędząc koleją z bardzo dużą prędkością również skorzystamy z infrastruktury drugiej lub trzeciej technologii. Ich zaletą jest niewątpliwie szersze pokrycie terenu. Podobnie instalując urządzenie płatnicze z modemem w automacie z napojami, wykorzystamy wolniejszą ale pewniejszą technologię 2G.

### **2.2. Złożoność sieci komórkowych**

Sieci komórkowe różnych generacji współpracują ze sobą. Technologia przesyłania głosu VoLTE wciąż jest nieobecna w katalogu usług wielu operatorów. Dzieje się tak ponieważ jej zalety – szybsze zestawianie połączeń i niższe zużycie baterii – nie są warte kosztów świadczenia takiej usługi. W zamian stacja nadawcza wykonuje tzw. fallback, czyli przekazanie telefonu klienta do sieci niższej generacji, która radzi sobie bez problemu z połączeniem głosowym [3].

Każdy operator komórkowy dysponuje złożoną siecią składającą się z wielu urządzeń dla uproszczenia łącznie nazywanych stacjami bazowymi, czyli BTSami (ang. Base Transceiver Station). Kolejne generacje telefonii komórkowej nadały tym urządzeniom szczegółowe nazwy: Node B (technologia 3G) oraz E-UTRAN Node B (w skrócie eNodeB – technologia 4G) [4].

Stacja bazowa jest co prawda pojedynczym fizycznym urządzeniem, ale swoją pracą może wspierać wiele tzw. komórek (ang. cells). Komórka to obszar geograficzny pokryty przez pojedynczy nadajnik.

Najpopularniejsze są komórki typu makro – są one obsługiwane przez znane nam z wysokich masztów, dachów szkół, czy wież kościołów urządzenia dalekiego zasięgu. W ostatnich kilku latach większą popularność zyskują small cells (pol. małe komórki) – urządzenia typu mikro, piko i femto – będące w stanie pokryć zasięgiem piętro galerii handlowej, kilka pięter wieżowca, trybunę stadionu, podziemną stację metra, blok, czy pojedynczy dom mieszkalny. Inną aberracją są zdalne głowice radiowe (ang. remote radio heads) instalowane w pewnym oddaleniu od BTSA.

To wszystko ma wpływ na złożoność sieci komórkowej. Do jej badania może nas skłonić ludzka ciekawość świata. Pojawiają się pytania: Gdzie jest najbliższa stacja nadawcza? Czy to ta, z której obecnie korzystam? Jaką technologię wykorzystuje mój telefon? Z jakimi stacjami łączyło się moje urządzenie podczas spaceru, a z jakimi podróżą samochodem czy pociągiem?

Oczywiście są też wzgłydy praktyczne: analiza infrastruktury sieci jest niezbędna np. przy instalacji internetu mobilnego w domu. Duże ważniejsze są tu jednak wzgłydy biznesowe. Duże przedsiębiorstwa i korporacje pokładają coraz większe zaufanie w rozwiązańach darmowych. Rozbudowując i dostosowując własnymi środkami oprogramowanie otwartoźródłowe (ang. open-source) zyskują takie same funkcjonalności jak przy zakupie drogiego, dedykowanego oprogramowania.

### 2.3. Rejestrator stacji bazowych

Biorąc pod uwagę opisaną wyżej złożoność infrastruktury połączeń bezprzewodowych i zapotrzebowanie na rozwiązanie analizujące obecność sprzętu użytkownika, zdecydowano się na realizację projektu rejestratora stacji bazowych (ang. BTS Tracker). Uwzględniając możliwości współczesnego smartfona oraz fakt, że nie ma konieczności stosowania żadnych innych narzędzi przejściowych, ustalono, że docelowym urządzeniem będzie telefon komórkowy. Obecnie najpopularniejszym systemem operacyjnym dla tego typu urządzeń jest rozwijany przez firmę Google system Android. Z tego prostego powodu oprogramowanie analizująco-wizualizujące powinno zostać w pierwszej kolejności zaprojektowane właśnie z myślą o tym systemie.

Sprawdzając istnienie tego typu rozwiązań w oficjalnym i dominującym zbiorze aplikacji mobilnych – Sklepie Google [5] – można dostrzec różne programy. Część z nich koncentruje się jedynie na analizie sygnału z aktualnie podłączonej komórki, niektóre umożliwiają lokalizację stacji nadawczych na mapie. Ciężko jednak znaleźć skuteczne rozwiązanie niekomercyjne. Dostępne aplikacje w przytaczającej większości są obarczone reklamami, a w części dostęp jest płatny. Dlatego istnieje zapotrzebowanie na rozwiązanie open-source, które może stanowić zarówno funkcjonalną aplikację, jak i solidną podstawę pod realizację większych projektów.

Zdecydowano się wyróżnić dwa zadania, które powinno realizować tego typu oprogramowanie. Wiążą się one z działającą w tle usługą (ang. service). Sednem tej usługi jest nasłuchiwanie i analiza zmian stanów dostępu telefonu do sieci komórkowej. Zmiana komórki jest natychmiast odnotowywana i przetwarzana, aby później informacje mogły być łatwo wizualizowane. Mogą być one przedstawiane w przejrzystej formie na liście lub w formie geograficznej na mapie.

Część z potrzebnych danych jest przekazywana ze stacji np. w postaci cyklicznie rozgłaszańskich bloków informacyjnych. System Android nie zezwala jednak na dostęp do większości informacji. Metody zawarte w pakietach systemu Android umożliwiają deweloperom jedynie dostęp do globalnego identyfikatora komórki (ang. Global Cell Id), który jednoznacznie identyfikuje urządzenie nadawcze w ogólnoświatowej sieci komórkowej. Jest to jednak uznawane przez twórców systemu za działanie

„niebezpieczne”, dlatego przy pierwszym „sięgnięciu” do tej danej deweloper musi postarać się o zgodę użytkownika aplikacji [6].

Specyfikacja 3GPP [4], konsorcjum odpowiedzialnego za standaryzowanie ogólnoświatowej sieci komórkowej, opisuje Global Cell Id jako złożenie parametrów: MCC (ang. Mobile Country Code, pol. Mobilny Kod Krajowy), MNC (ang. Mobile Network Code, pol. Mobilny Kod Sieci), LAC (ang. Location Area Code, pol. Kod Obszaru Lokalizacji) oraz C-ID (ang. Cell Identifier, pol. identyfikator komórki). Przetwarzając ten zestaw i poszukując odpowiadającego mu rekordu w tablicy danych, można uzyskać szczegółowe informacje.

Chociaż większość operatorów utajnia dane dotyczące swojej infrastruktury, w Internecie funkcjonuje obecnie wiele baz danych zawierających informacje o poszczególnych komórkach. Część z nich jest płatna i realizowana przez prywatne firmy, a część otwarta i tworzona przez społeczność, przy czym generalnie te drugie dotrzymują kroku pierwszym. Oprogramowanie powinno zostać zaprojektowane w ten sposób, aby móc współpracować z dowolną bazą. Najlepszym rozwiązaniem wydaje się wykorzystanie w tym celu pojedynczej tablicy danych zawartej w pliku tekstowym CSV (ang. comma-separated values, pol. wartości rozdzielone przecinkami). Dzięki odpowiedniemu przetworzeniu można będzie dostosować do wymogów programu dowolny zbiór danych.



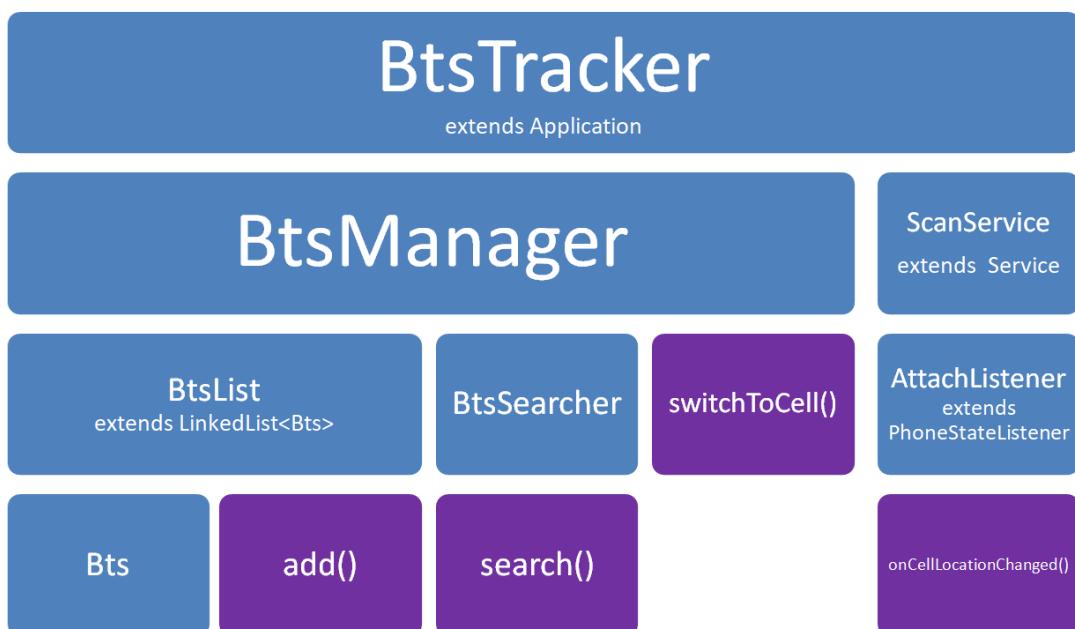
## 3. Propozycja rozwiązania

### 3.1. Struktura aplikacji

Za kolekcjonowanie danych dotyczących stacji bazowych oraz zarządzanie nimi odpowiada obiekt klasy BtsManager. Dane te stanowią trzon aplikacji i muszą być dostępne z poziomu aktywności, fragmentów i usług. W związku z powyższym instancję klasy umieszczono jako pole obiektu BtsTracker, dziedziczącego po Application.

Z kolei jednym z ważniejszych założeń programu jest możliwość działania w tle, dlatego usługa odpowiadająca za nasłuchiwanie zmiany stanu telefonu komórkowego powinna znajdować się w odrębnym wątku. Tą usługą jest ScanService – uruchamia ona obiekt słuchacza AttachListener. Jego nadpisana metoda onCellLocationChanged() odwołuje się do metody switchToCell() obiektu typu BtsManager.

Następuje przeszukanie pliku CSV, za co odpowiada metoda search() z klasy BtsSearcher a następnie, jeżeli konieczne, utworzenie obiektu Bts i dodanie go do listy BtsList.



**Rys. 3.1.** Schemat blokowy ilustrujący klasy i ważniejsze metody biorące udział w procesie wykrywania stacji bazowych. Kolorem niebieskim wyróżniono klasy, a fioletowym metody. Jeżeli klasa dziedziczy po innej, dostępnej w zbiorze pakietów systemu Android, oznaczono to słowem „extends”.

Projekt zrealizowano w języku Java [10].

### 3.1.1. Wykrywanie stacji bazowych

#### 3.1.1.1. Klasa ScanService

W „BTS Tracker” obiekt odpowiedzialny za wykrywanie urządzeń nadawczych jest usługą. ScanService została zaimplementowana jako rozszerzenie usługi (Service) z pakietu dla deweloperów systemu Android. Główną różnicą pomiędzy aktywnością, a usługą jest fakt, że w odróżnieniu od pierwszych, które koncentrują się na graficznym interfejsie użytkownika, te drugie przeznaczone są do działania w tle [8]. Usługa może być aktywna, kiedy aktywność zostanie uśpiona lub usunięta. Również kiedy program pozostaje zminimalizowany, usługa realizuje swoje zadanie.

Ponieważ główny wątek aplikacji odpowiada za szereg zadań, przede wszystkim za wszelkiego rodzaju interakcje z graficznym interfejsem użytkownika, zdecydowano o przesunięciu odpowiedzialności za wypełnianie danych na nowy wątek. Tę funkcjonalność implementuje ScanThread – klasa wewnętrzna z interfejsem Runnable.

ScanService stanowi jedynie pomost pomiędzy aktywnością MainActivity (a dokładniej jej polem typu FloatingActionButtonOnClickListener – słuchaczem dla przycisku włączającego i wyłączającego proces skanowania), a wątkiem ScanThread.

Sam wątek tworzy obiekt TelephonyManager (klasy systemu Android odpowiedzialnej za przechowywanie danych dotyczących połączeń telefonicznych) i podpina do niego obiekt słuchacza AttachListener.

#### 3.1.1.2. Klasa AttachListener

Twórcy systemu Android zadali o to, aby stworzyć praktyczny interfejs pomiędzy warstwą sprzętową telefonu komórkowego, a jego oprogramowaniem. Oczywiście dostęp do wielu informacji, jakimi dysponuje system, jest zabroniony, a do części ograniczony – wymaga specjalnego pozwolenia od użytkownika w czasie działania programu. Jednak dzięki tym danym, które udostępnia klasa PhoneStateListener z pakietu android.telephony można rozpoznać stacje bazowe, z jakimi łączy się telefon. [6]

PhoneStateListener ma charakter słuchacza – jej konkretne metody uruchamiane są w momentach, kiedy mają miejsce konkretne zmiany stanu urządzenia. Z kolei parametrami tych metod są obiekty zawierające bardziej szczegółowe informacje o tym, co się wydarzyło. Należy więc jedynie stworzyć podkласę dziedziczącą po PhoneStateListener i nadpisującą daną metodę.

Taką podklastą jest AttachListener. Jego pola to:

- Referencja do obiektu typu TelephonyManager, wypełnianego przez system Android informacjami dotyczącymi połączeń.
- Referencja do obiektu typu BtsManager, do którego AttachListener powinien przesyłać informacje o stacji BTS, z którą połączył się telefon.

W momencie wywołania metody onCellLocationChanged() (pol. przy zmianie lokalizacji komórki) za pomocą metody getAllCellInfo() obiektu TelephonyManager pobierana jest lista komórek, o których informacje posiada urządzenie. Dane te zebrane są w obiekcie typu CellInfo. Trzeba z nich jeszcze wybrać te, które są istotne. Iterując po liście, wywoływana jest na instancji CellInfo metoda isRegistered() zwracająca prawdę, kiedy telefon jest zarejestrowany w danej komórce.

Wreszcie następuje przekazanie obiektu CellInfo do BtsManagera poprzez metodę switchToCell(). Równolegle przekazywana jest nazwa operatora, którą TelephonyManager przechowuje poza obiektem CellInfo.

### 3.1.2. Przetwarzanie zebranych informacji

#### 3.1.2.1. Przechowywanie danych

Informacje udostępniane deweloperom przez system Android pozwalają na jednoznaczne zidentyfikowanie stacji nadawczej telefonii komórkowej, jednak nie są wystarczające do tego, aby wyświetlić szczegóły dotyczące lokalizacji urządzenia, czy nanieść jego pozycję na mapę.

W związku z powyższym potrzebna jest baza danych, przechowująca zarówno podstawowe informacje, jak i te dodatkowe. Celem „BTS Tracker” jest otwartoźródłowość, dlatego aplikację zaprojektowano tak, żeby mogła współpracować z dowolną zewnętrzną hurtownią danych. Może być to zasób ogólnodostępny lub komercyjny (odpłatny). Dane udostępnione przez zasób należy jedynie sparsować, na przykład z wykorzystaniem wyrażeń regularnych.

Wewnętrzną bazę aplikacji sprowadzono do postaci tablicy CSV (uporządkowanych składowych rozdzielonych za pomocą znaku separatora). Będzie ona działać na zasadzie Look-Up Table. Jeżeli dany rekord zawiera w sobie na początku poszukiwaną frazę, jego dalsza część będzie wykorzystana jako dodatkowe informacje. Dzięki tak prostej realizacji możliwe jest parsowanie dowolnej bazy.

W tym konkretnym przypadku również szybkość przeszukiwania tablicy nie jest kluczowa. Należy pamiętać, że skanowanie odbywa się w tle, często przy zminimalizowanej aplikacji, a dane w graficznym interfejsie użytkownika nie są aktualizowane w czasie rzeczywistym.

Rekord w tablicy CSV składa się z następujących elementów:

- MCC – Mobile Country Code – numer oznaczający kraj, w którym działa operator sieci komórkowej [9].
- Identyfikator technologii: 2, 3 lub 4 – oznaczający odpowiednio generacje: 2G, 3G lub 4G.
- LAC – Location Area Code – numer obszaru, na które podzielona jest sieć komórkowa [9]. Dla technologii LTE ta wartość jest w programie ignorowana.
- ID – identyfikator urządzenia – składa się z dwóch elementów. Opisany dalej.
- Nazwa miejscowości.
- Bardziej szczegółowe informacje o lokalizacji, np.: *ul. Grunwaldzka 32 - Galeria Grunwaldzka, ul. Wiśniowieckiego 56 - komin MPEC - MILLENIUM*.
- Szerokość geograficzna.
- Długość geograficzna.

Przy implementacji wybrano jednolity sposób wyrażania współrzędnych geograficznych. Podczas parsowania danych należy zadbać o to, żeby były one wyrażone w godzinach, z częścią ułamkową rozdzieloną znakiem kropki oraz ze znakiem minus dla półkuli południowej i półkuli zachodniej.

Numer identyfikatora urządzenia jest zależny od technologii. W przypadku czwartej generacji (LTE) tablica powinna zawierać eNodeB ID (numer identyfikacyjny stacji nadawczej E-UTRAN). Tutaj drugi z elementów ID z tablicy CSV nie odgrywa znaczenia. Dla pozostałych technologii numer oznacza Cell Id (pol. identyfikator komórki) i jest dwuczęściowy, przy czym druga część to ostatnia cyfra numeru Cell Id.

Skąd ten rozdział? W technologii mobilnej powszechna jest sytuacja, że jedno urządzenie BTS posiada kilka komórek. Zazwyczaj znajdują się one w tej samej lokalizacji, a ich numery ID rozróżnia

tylko ostatnia cyfra [11]. Stąd, aby uniknąć sytuacji, kiedy kolejne rekordy tablicy CSV różnią się tylko numerem ID, można zastosować inny szkł: pierwszą część wypełnić numerem podzielonym przez 10, a drugą część uzupełnić gwiazdką.

Czasem zdarza się, że operator stosuje odstępstwo od tej reguły i nadaje komórkom w różnych lokalizacjach numery różniące się tylko ostatnią cyfrą. Wówczas w bazie pierwsze cyfry należy wypełnić w ten sam sposób, a wszystkie możliwe kombinacje ostatniej cyfry zebrać w ciąg i wypełnić nimi drugie pole. Ta niezwykle prosta optymalizacja pozwala kilkukrotnie zmniejszyć rozmiar tablicy.

### Pojedynczy rekord z tabeli CSV może opisywać więcej niż jedną komórkę

#### Sytuacja 1

Wszystkie komórki, których numer identyfikacyjny różni się jedynie ostatnią cyfrą, znajdują się w tej samej lokalizacji.

#### Przykład

**2014;**  
\*;  
**Książ Wielki;**  
**maszt T-Mobile;**

Wszystkie komórki z prefiksem 2014 znajdują się w Książu Wielkim na maszcie T-Mobile.

#### Sytuacja 2

Komórki znajdują się w różnych lokalizacjach, chociaż ich numer identyfikacyjny różni się jedynie ostatnią cyfrą.

#### Przykład

**2009;**  
1237;  
**Nowy Targ;**  
**ul. Królowej Jadwigi 17 - biurowiec;**

Komórki 20091, 20092, 20093 oraz 20097 znajdują się w Nowym Targu przy ul. Królowej Jadwigi.

**2009;**  
456;  
**Nowy Targ;**  
**ul. Ceramiczna 10 - komin;**

Komórki 20094, 20095 oraz 20096 znajdują się w Nowym Targu przy ul. Ceramicznej.

### Przykładowy wyciąg z tablicy CSV

260;2;2222;**2014;\***;Książ Wielki;maszt T-Mobile;50.4325;  
20.12972222

260;2;2222;**2009;1237;**Nowy Targ;ul. Królowej Jadwigi 17 - biurowiec;49.478611111;20.026666667

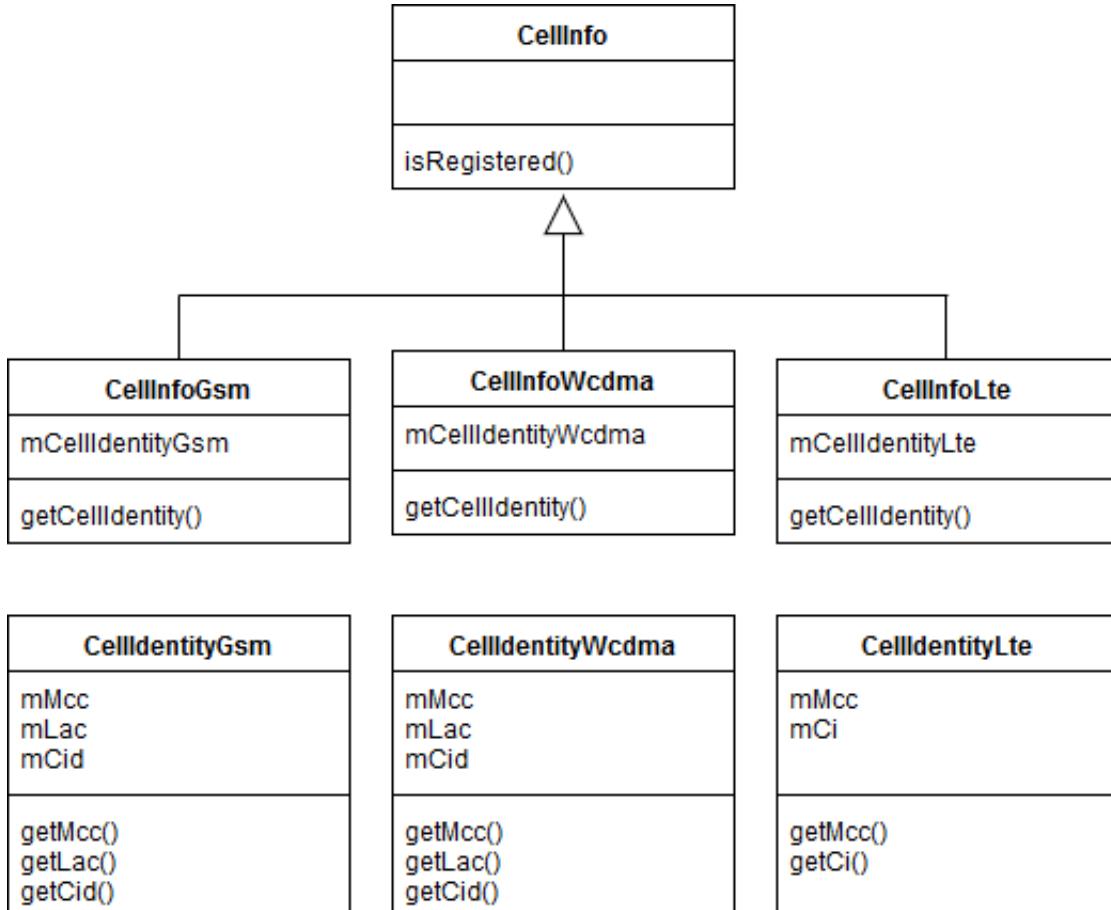
260;2;2222;**2009;456;**Nowy Targ;ul. Ceramiczna 10 - komin;  
49.471111111;20.015277778

**Rys. 3.2.** Schemat optymalizacji rozmiaru bazy danych na przykładzie wyciągu z tablicy CSV. Przy konstruowaniu zbioru danych numer identyfikacyjny komórki należy rozbić na dwie kolumny – w kolumnie czwartej umieścić prefiks (wszystkie cyfry numeru poza cyfrą jedności), a w kolumnie piątej gwiazdkę (symbolizującą wszystkie cyfry) lub ciąg odpowiednich cyfr jedności (nierozdzielonych).

Cała tablica znajduje się w pliku btsData.csv w katalogu assets – domyślnym miejscu przeznaczonym dla dodatkowych zasobów.

### 3.1.2.2. Klasa BtsSearcher

Za przeszukiwanie tablicy danych odpowiada obiekt typu BtsSearcher. Jego zadaniem jest przyjęcie obiektu typu CellInfo i zwrócenie obiektu typu Bts opisującego stację nadawczą. Co ważne, BtsSearcher nie odpowiada za przetwarzanie rekordów z tablicy CSV, a jedynie za ich dostarczenie do obiektu Bts.



Rys. 3.3. W celu pobrania danych komórkowych skorzystano z pakietu android.telephony. Niniejszy diagram przedstawia jedynie wykorzystane klasy, pola i metody.

Klasa ma cztery pola: mcc, networkGeneration, lac, id. Wszystkie typu int. Odpowiadają one pierwszym czterem elementom rekordu danych. Kiedy BtsManager wywoła metodę search() instancji BtsSearcher, w pierwszej kolejności zostanie wykonana metoda processCellInfo(), której zadaniem jest wypełnienie tych pól składowych. Działa ona różnie w zależności od technologii, w jakiej funkcjonuje dana komórka. Zastosowano tu operator instanceof, rzutowanie obiektu CellInfo i wywołanie getCellIdentity().

- Z obiektu CellIdentityGsm (technologia 2G) pobierane są wprost numery MCC, LAC i CID.
- Z obiektu CellIdentityWcdma (technologia 3G) dokonywana jest dodatkowa koniunkcja bitowa numeru CID z liczbą 0xFFFF. W technologii UTRAN na starszych bitach przechowywany jest charakterystyczny dla niej dodatkowy identyfikator RNC (Radio Network Controller) [11].

- Z obiektu CellIdentityLte pobierany jest MCC (wprost) oraz eNodeB ID – stosując przesunięcie bitowe w prawo o osiem na wyniku getCi(). Młodszy bit wykorzystuje się do zapisu CID [9].

Po wypełnieniu pól wartościami pobranymi z CellInfo, zostaje sformułowana fraza, a następnie program wykorzystuje InputStreamReader do odnalezienia w pliku CSV odpowiedniego rekordu. Ostatecznie następuje jeszcze sprawdzenie ostatniej cyfry ID. Jeżeli wartość w kolumnie jest różna od znaku gwiazdki, wartości z kolumny CSV i pola obiektu BtsSearcher muszą się po prostu zgadzać.

### 3.1.3. Przechowywanie danych

#### 3.1.3.1. Klasa Bts

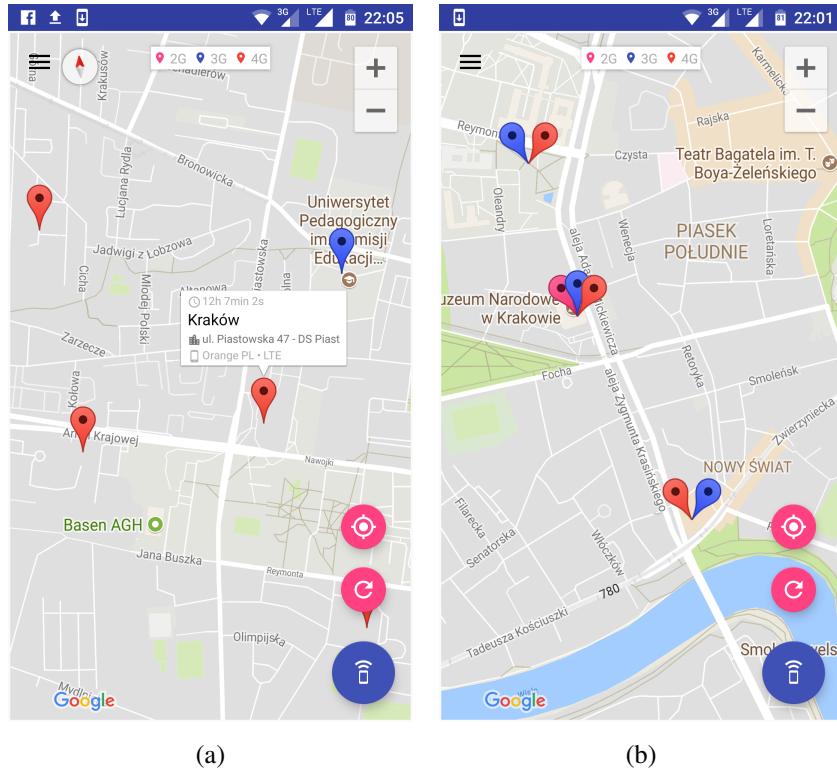
Informacje o pojedynczej stacji przekaźnikowej „BTS Tracker” przechowuje w obiektach typu Bts. Zawierają one pola:

- int networkGeneration – numer generacji sieci telefonicznej: 2G (GSM), 3G (WCDMA) lub 4G (LTE).
- String town – nazwa miejscowości, w której zlokalizowany jest BTS.
- String location – bardziej szczegółowy opis miejsca, w którym zlokalizowany jest BTS.
- String networkType – typ sieci (GSM, WCDMA, LTE) odpowiadający polu networkGeneration.
- LatLng latLng – obiekt opisujący szerokość i długość geograficzną.
- Oraz: long timeAttachedLong, String timeAttached, int rotation – omówione niżej.

Konstruktor klasy otrzymuje jako parametr łańcuch znaków będący pojedynczym wierszem z pliku CSV i wypełnia powyższe pola. Jest też dostępny inny konstruktor, również przyjmujący jako parametr rekord CSV – służy on do wczytywania danych zapisanych wcześniej do pliku. Poza tym klasa Bts implementuje metody zwracające odpowiednie rekordy przy eksportie do pliku CSV (dokładnie BTST) oraz XML (dokładnie KML) oraz metodę zwracającą opcje markera (znacznika) gotowe do naniesienia na mapę.

Obiekty Bts odpowiadają dodatkowo za dwie rzeczy. Pierwszą z nich jest czas podłączenia telefonu do danej stacji nadawczej. Przy odłączeniu telefonu wywoływana jest metoda detach(), która odświeża wartości timeAttachedLong (czas w milisekundach) oraz timeAttached (czas w dniach, godzinach, minutach, sekundach).

Ostatnią funkcją jest możliwość rotacji. Chodzi tu o obrót wskaźnika na mapie. Może się zdarzyć, że w jednej lokalizacji zajdzie konieczność umieszczenia dwóch a nawet trzech znaczników – mamy do czynienia z trzema funkcjonującymi generacjami. Możliwe konfiguracje to -30, 0 oraz +30 stopni.



**Rys. 3.4.** Rysunek (a) zawiera przykład działania timera. W górze okienka przedstawiającego informacje o urządzeniu nadawczym wyświetlona jest informacja o tym jak długo telefon był podłączony. Na rysunku (b) można w praktyce zobaczyć jak działa rotacja. Wskaźniki dla dwóch i trzech stacji znajdujących w jednym punkcie zostały obrócone.

### 3.1.3.2. Klasa BtsManager

Zebrane dane o stacjach przekaźnikowych stanowią trzon aplikacji i muszą być dostępne dla wielu elementów różnych typów: usługi, aktywności, fragmentów. W związku z powyższym dla obiektu tej klasy zrezygnowano z klasycznego sposobu wstrzykiwania referencji pomiędzy obiektami. Jest on tworzony w instancji specjalnej klasy BtsTracker, która dziedziczy po Application. Każdy obiekt, który ma dostęp do Application, może wywołać getBtsManager() i uzyskać referencję do obiektu, który jest odpowiedzialny za przechowywanie, zbieranie i udostępnianie danych.

Informacje są przechowywane w liście BtsList, która jest klasą wewnętrzną, dziedziczącą po liście wiązanej składającej się z elementów typu Bts. Listę może wypełnić tylko BtsManager, wywołując jedyną dodatkową metodę – addBts(). Co ważne, nowy element nie jest tak po prostu dodawany do listy. Bardzo prawdopodobną jest sytuacja, kiedy telefon użytkownika podłącza się ponownie do tej samej stacji bazowej. Wówczas obiekt jest przenoszony na szczyt listy. Dzięki temu w widoku poszczególne stacje wyświetlały się chronologicznie – od aktualnej do najstarszej.

Dodatkowo BtsManager zawiera metody wspomagające obiekt Bts jeżeli chodzi o mierzenie czasu i rotację oraz różnego rodzaju gettery.

## 3.2. Graficzny interfejs użytkownika

### 3.2.1. Klasa MainActivity

Aktywność to fundamentalny blok w architekturze aplikacji mobilnych. Odpowiada za interakcję użytkownika z programem – definiuje jego wygląd i wywołuje odpowiednie reakcje [8]. W kontraste do coraz mniej popularnego podejścia polegającego na przełączaniu kolejnych widoków, w aplikacji zastosowano podejście jednej aktywności, powszechnie w nowoczesnych programach zgodnych z Material Design.

Aktywność sama w sobie odpowiada jedynie za wyświetlanie podstawowych elementów na ekranie. Przede wszystkim jednak spina widoczną dla użytkownika całość. Odpowiada za cztery rzeczy:

- Wyświetlenie po lewej stronie wysuwanego DrawerMenu oraz obsługę wybranych przez użytkownika akcji.
- Wyświetlenie w prawym dolnym rogu przycisku Scanning, za pomocą którego użytkownik może wyłączyć lub włączyć usługę skanowania w tle.
- Wyświetlenie u góry paska narzędzi z przyciskiem „hamburger button” (trzy poziome kreski), zapewniającego w dowolnej chwili możliwość wysunięcia menu.
- Zapewnienie pozostały przestrzeni dla fragmentów, które będą wykorzystywały ten obszar dla interakcji z użytkownikiem oraz wyświetlania i wizualizacji danych.

Dzięki temu, niezależnie od wykonywanej akcji użytkownik może być pewny, że wybierając przycisk „hamburger button” albo przeciągając palcem z lewej strony uzyska dostęp do menu, a także w dowolnym momencie będzie w stanie kontrolować usługę skanowania w tle.

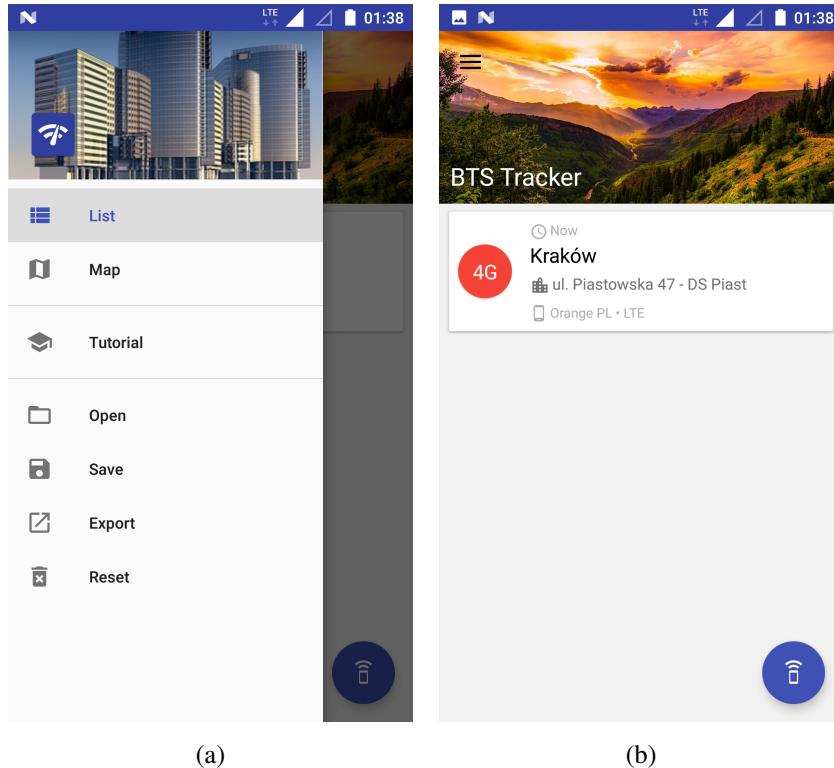
#### 3.2.1.1. Układ elementów

W programowaniu dla systemu Android podstawowym sposobem kontroli wyglądu zewnętrznego aplikacji są dodatkowe zasoby w postaci plików XML. Definiują one tzw. layouts, czyli opis zachowań poszczególnych elementów, ich pozycji, widoczności, rozmiaru, marginesu, paddingu, koloru i kształtu.

Podczas tworzenia aktywności „nadmuchiwany” (ang. inflate) jest DrawerLayout, który spina ze sobą wysuwane menu oraz pozostałe elementy widoku. Elementy menu przedstawione są w sposób uporządkowany. Oddzielono od siebie główne funkcjonalności aplikacji (wyświetlenie danych oraz wizualizacja danych), funkcjonalność pomocniczą (Tutorial) oraz operacje na plikach i danych.

Pozostałe elementy umieszczone są w wydajnym Coordinator Layout. Definiuje on położenie, kształt i kolor przycisku Scanning. Typ tego przycisku to FloatingActionButton. Wykorzystywany jest, aby zakończać ważną lub często wykonywaną akcję. Dzięki takiemu elementowi dostęp do funkcjonalności jest niemal nieogarniczony. Przycisk jest zaprojektowany tak, aby „unosił się” nad pozostałymi elementami.

Umiejscowienie ostatnich dwóch elementów związanych z MainActivity definiuje powszechnie stosowany RelativeLayout. U góry strony tworzy on miejsce na pasek narzędzi (Toolbar), a pozostałe miejsca pozostawia jako puste do wykorzystania przez fragmenty, które będą otwierane (FrameLayout). Sam pasek jest ograniczony do absolutnego minimum, wręcz do jednej ikony (trzech pasków), która odpowiada za otwieranie menu.



**Rys. 3.5.** Po wysunięciu z lewej (a) menu przykrywa pozostałe elementy układu. Po wsunięciu menu (b) użytkownik zawsze ma dostęp do przycisków FAB oraz „hambugera”.

### 3.2.1.2. Implementacja: przy uruchomieniu

Klasa MainActivity dziedziczy po zaimplementowanej w pakietach systemu AppCompatActivity. Umożliwia ona skutecną obsługę opisanego wyżej menu i paska narzędzi. Ponieważ w manifeście zdefiniowano ją jako główną, jest uruchamiana przy starcie aplikacji. Wobec tego w metodzie onCreate() muszą wykonać się wszystkie czynności konieczne po otwarciu programu.

W pierwszej kolejności uruchamiany jest wyżej opisany Layout. Następuje wyświetlenie jednoelementowego paska narzędzi oraz przyporządkowanie obiektów słuchaczy (ang. Listener) odpowiadających za uruchomienie skanowania oraz wysunięcie menu. Można to zrobić na dwa sposoby: przesuwając palcem z lewej lub wybierając przycisk.

Konieczne jest również zagospodarowanie zdefiniowanej w układach XML wolnej przestrzeni dla fragmentów. Domyślnie uruchamiany jest ListFragment – klasa, która ma za zadanie wyświetlenie wszystkich stacji nadawczych, do których podłącza się urządzenie. Wykorzystując menu, użytkownik może reagować dwójako: przełączać fragmenty definiujące zawartość wolnej przestrzeni lub wywoływać natychmiastowe akcje związane na przykład z obsługą plików. Konieczne jest więc zainicjowanie obiektu TaskManager, który odpowiadać będzie za wywoływanie akcji.

W przypadku niektórych czynności, mających wpływ na „otoczenie” aplikacji (np. wykorzystujących zasoby sprzętowe jak GPS, czy zasoby systemu jak pamięć wewnętrzna) system Android wymaga zgody użytkownika w czasie działania aplikacji. Zgody takiej należy udzielić raz przy pierwszym starcie programu. Tworzona jest więc klasa Permission Manager odpowiedzialna za sprawdzenie zezwoleń i – w razie ich braku – odpytanie użytkownika.

Wreszcie uruchamiane jest skanowanie, a następnie po krótkim odstępie czasu widok listy jest odświeżany. Pojawia się na nim pierwsza znaleziona stacja nadawcza.

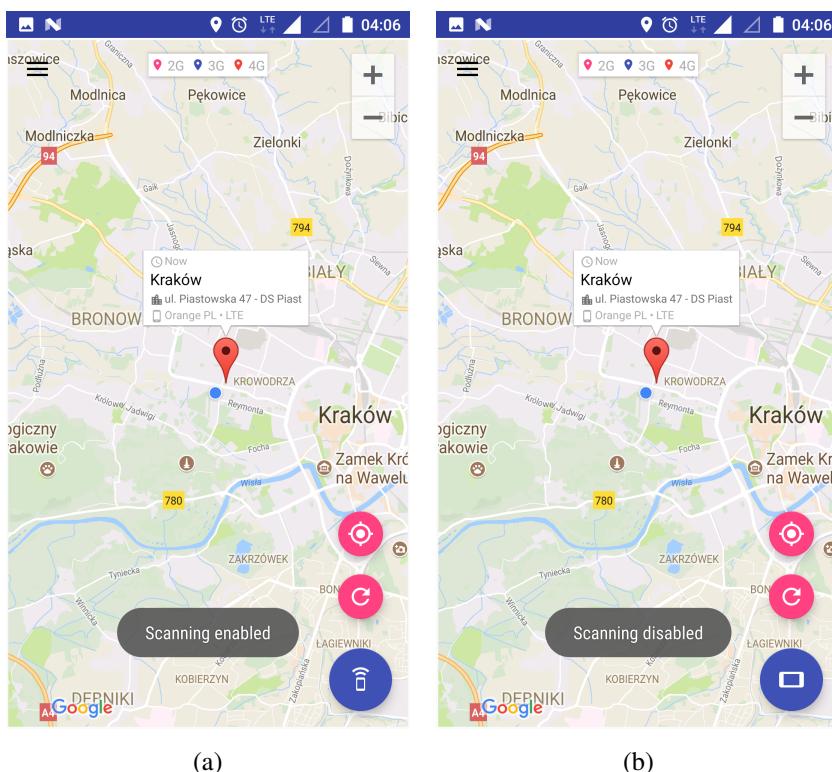
### 3.2.1.3. Implementacja: przy wyborze akcji

Oprócz uruchomienia pozostałych elementów programu, MainActivity odpowiada także za reagowanie na polecenia użytkownika. Metoda onNavigationItemSelected() wywołuje zmianę fragmentu lub podjęcie akcji w klasie TaskManager. Z kolei zwinięcie menu następuje ono po kliknięciu ikony, ale także po użyciu systemowego przycisku wstecz (metoda onBackPressed()).

### 3.2.2. Klasa FloatingActionButtonOnClickListener

Klasa dziedziczy po View.OnClickListener i jest tworzona jedynie w MainActivity. Tylko za pomocą jej metody switchState() możliwe jest włączenie/wyłączenie skanowania. Zatem pomimo niewielkich rozmiarów spełnia ważną rolę – jest mostem pomiędzy główną aktywnością a usługą ScanService.

Metoda może być wywołana przez naciśnięcie przycisku. Zmienia się wówczas ikona skanowania (aktywne/nieaktywne) oraz pojawia się komunikat typu Toast. Istnieje też drugi sposób wywołania metody – poprzez funkcję doubleSwitchFabState() z klasy MainActivity. Funkcja jest skonstruowana tak, że może ją uruchomić tylko obiekt TaskManager (analogia zaprzyjaźnienia z C++). Takie podwójne wirtualne „kliknięcie” przycisku powoduje odświeżenie danych aktywności np. po resecie.



**Rys. 3.6.** Klasa słuchacza odpowiada nie tylko za stan usługi ScanService. Przycisk Scanning (w prawym dolnym rogu) może przyjmować stany enabled (a) i disabled (b). Dodatkowo przy przełączeniu pojawia się komunikat.

### 3.2.3. Klasa PermissionsManager

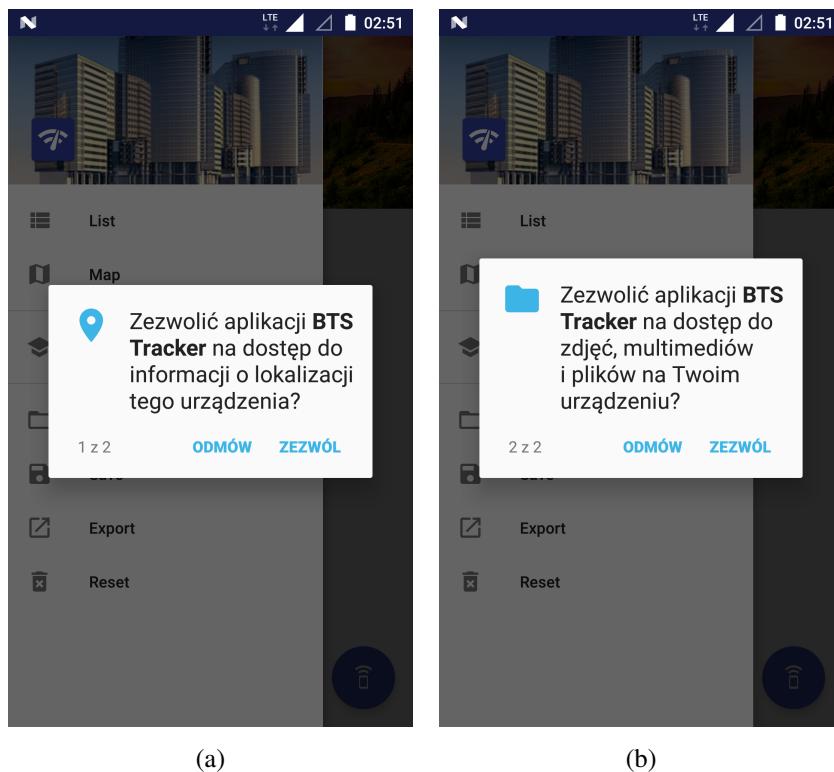
„BTS Tracker” to nie tylko wyświetlanie danych. Główną funkcjonalnością aplikacji jest współpraca ze sprzętem – z telefonem komórkowym, który udostępnia informacje o statusie swojego połączenia z siecią. System Android wymaga wymienienia takich funkcjonalności w Manifeście. Użytkownik, który instaluje aplikację ze sklepu Play [5] może zobaczyć, z jakich elementów systemu korzysta nowo-instalowany program.

Dodatkowo, w przypadku funkcji, gdzie istnieje choć małe podejrzenie, że przy złych zamiarach twórców, mogą one w jakiś sposób wyrobić szkodę, wymagane jest uzyskanie zgody poprzez specjalne odpytanie użytkownika.

**Tabela 3.1.** Pozwolenia, jakie musi uzyskać aplikacja, aby działać poprawnie.

Funkcjonalność	Etykieta
Informacja o lokalizacji urządzenia (GPS)	ACCESS_FINE_LOCATION
Informacja o dostępie do sieci komórkowej	ACCESS_FINE_LOCATION
Odczyt danych z pliku	READ_EXTERNAL_STORAGE
Zapis danych do pliku	WRITE_EXTERNAL_STORAGE

Każdorazowo przy starcie programu klasa MainActivity wywołuje metodę hasPermissions(). W przypadku pozytywnej odpowiedzi następuje uruchomienie skanowania w tle i naniesienie na listę pierwszych wyników. Z kolei w przypadku negatywnej – ponowne odpytanie.



**Rys. 3.7.** Przy pierwszym starcie aplikacji użytkownik zostaje zapytany o zgodę na dostęp do informacji o lokalizacji (a) oraz do plików (b).

### 3.2.4. Klasy TaskManager i FileManager

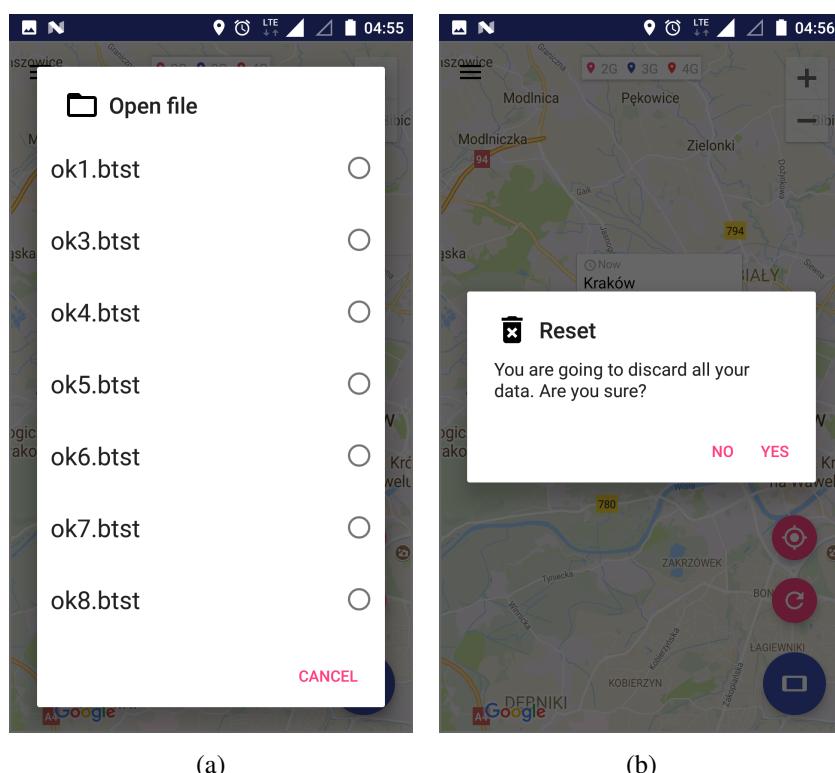
Samo wyświetlenie danych to ważna funkcjonalność, ale w praktyce zebrane dane trzeba często przeanalizować.

- Może być konieczny powrót do danych pobranych wcześniej.
- Użytkownik może chcieć kontynuować zbieranie danych po restarcie telefonu.
- Sieć można skanować z kilku urządzeń na różnych obszarach.
- Może pojawić się potrzeba analizy map na większym ekranie.

W związku z powyższym w „BTS Tracker” wprowadzono możliwość operacji na plikach. Za tę funkcjonalność odpowiada klasa FileManager, która jest polem klasy TaskManager. Umożliwia ona:

- Zapis do pliku BTST (CSV dla stacji nadawczych „BTS Tracker”).
- Pobranie listy plików BTST z pamięci telefonu.
- Odczyt z pliku BTST.
- Eksport do pliku KML (XML dla Google Maps).

Zapis, eksport i odczyt dotyczą oczywiście listy BtsList z klasy BtsManager.



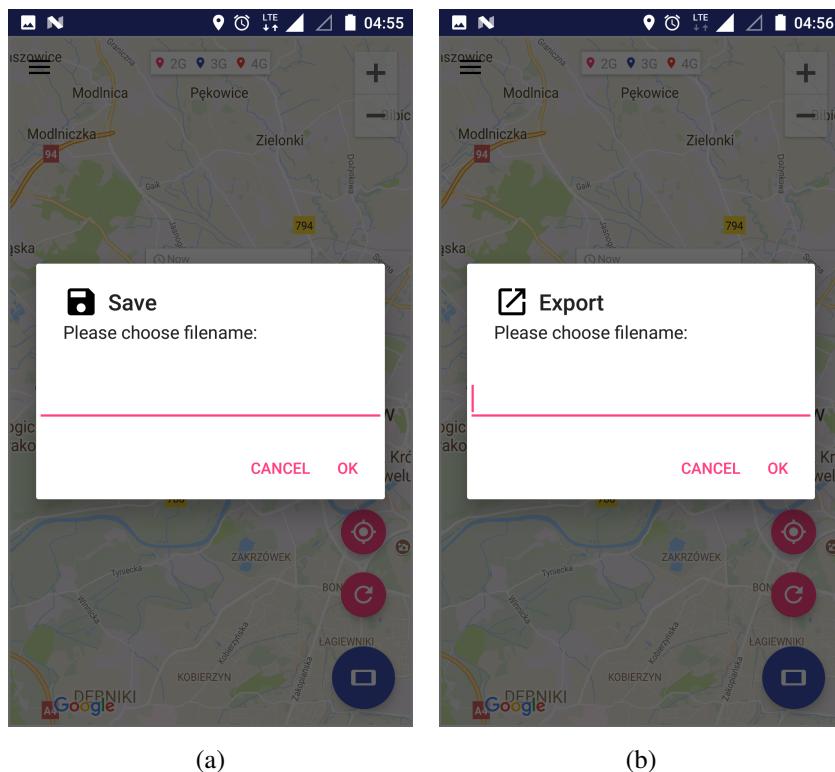
**Rys. 3.8.** Przy otwarciu FileManager znajduje w domyślnej lokalizacji pliki z rozszerzeniem BTST i wyświetla (a), aby użytkownik mógł wybrać, który utworzyć. Przy reseście konieczne jest potwierdzenie (b) zgody na usunięcie danych. Podobna prośba o potwierdzenie pojawia się przy otwarciu przed oknem dialogowym (a).

Keyhole Markup Language to oparty na XML język znaczników z otwartym standardem [7] często wykorzystywany w pracy z mapami, także przez GoogleMaps. Eksport do tego formatu pozwoli na przykład na przeglądanie zebranych danych na większym ekranie. Konieczne jest tutaj dodanie specjalnego nagłówka definiującego m.in. style oraz niedługiej stopki.

Klasa TaskManager odpowiada za interakcje z użytkownikiem po wybraniu zadania z menu (klasa MainActivity). Obsługuje cztery możliwe zadania: open (odczyt z pliku BTST), save (zapis do pliku BTST), export (eksport do pliku KML), reset (wyczyszczenie listy BtsList).

Domyślną lokalizacją plików jest katalog `BTS_Tracker`. Po wybraniu opcji save i export na ekranie pojawia się prośba o podanie nazwy pliku. Rozszerzenie jest dodawane automatycznie. Natomiast po wybraniu open lub reset użytkownik jest proszony o potwierdzenie, ponieważ dane zostaną utracone. Interakcje zrealizowane są za pomocą interfejsu DialogInterface z pakietów systemu Android.

Po wyborze odpowiedniej opcji wywoływane są metody na obiekcie `fileManager`. Aby nie obciążać głównego wątku, który odpowiada za wyświetlanie elementów, otwierany plik zostaje przeczytany w osobnym obiekcie `Runnable`. Na końcu następuje jeszcze odświeżenie widoku przez restart usługi `ScanService`.



Rys. 3.9. Przy zapisie (a) (do otwarcia w telefonie) i eksportie (b) (do otwarcia w komputerze) wyświetla się prośba o podanie nazwy pliku docelowego.

### 3.2.5. Klasa ListFragment i pakiet list

W pakietach systemu Android można znaleźć kontenery, które łatwo wykorzystać do wyświetlenia listy o nieznanej lub dynamicznie zmieniającej się liczbie elementów. Do tego zadania może być użyty

kontener RecyclerView. Jego zaletą jest fakt, że ułatwia on pracę zgodną z wytycznymi budowania interfejsów Material Design [8]. Dobrze radzi sobie również z dużymi ilościami danych, dzięki czemu daje możliwość szybkiego przewijania ekranu.

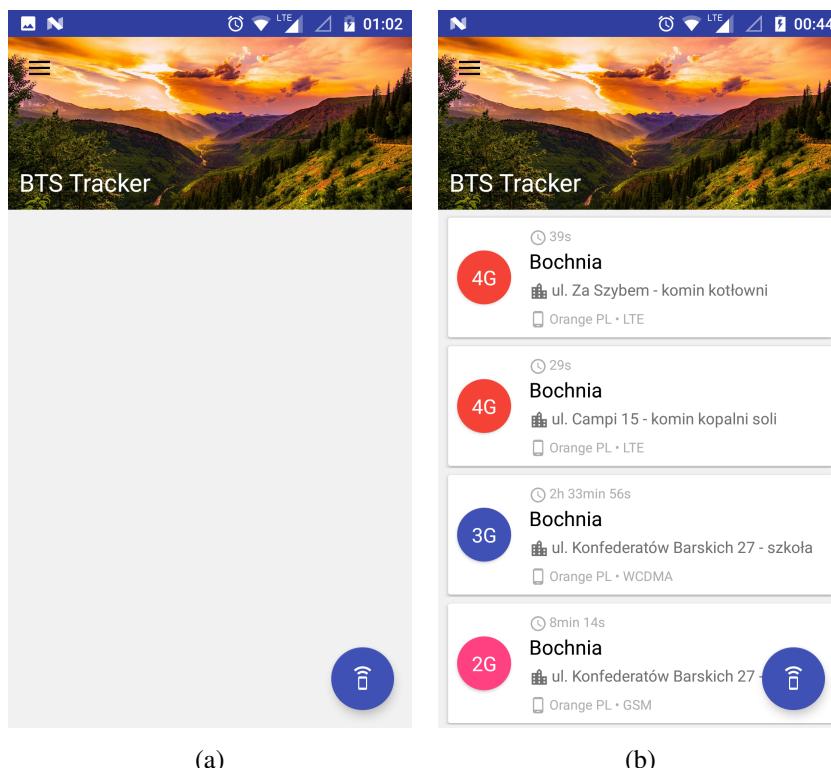
Dodatkowo wydajność działania zwiększa zastosowanie wzorca projektowego ViewHolder. Dworzony zostaje obiekt, który w swoich polach przechowuje referencje do elementów widoku. Raz pobrane – mogą być wykorzystane wielokrotnie.

Z implementacją kontenera wiążą się kolejne zadania zadania:

- Utworzenie pliku XML układu fragmentu z miejscem przeznaczonym na widok kontenera.
- Utworzenie pliku XML pojedynczego elementu listy.
- Implementacja odpowiedniego kodu fragmentu, który będzie się odwoływał do kontenera RecyclerView.
- Implementacja adaptera RecyclerView.Adapter i RecyclerView.ViewHolder – klasy z referencjami.

### 3.2.5.1. Układ elementów

Ponieważ w jednym z fragmentów mamy do czynienia z mapami, rozmiar ekranu odgrywa wówczas niemałe znaczenie. Dlatego w widoku nadziednym związanym z MainActivity zdecydowano się na ograniczenie do absolutnego minimum paska narzędzi w górnej części widoku aplikacji.



**Rys. 3.10.** Domyślnie widok ListFragment (a) składa się z podstawowych elementów widoku nadziednego oraz z rozszerzonego paska (grafika + nazwa aplikacji). Dopiero w miarę wykrywania kolejnych urządzeń, widok listy wypełnia się pojedynczymi elementami (b) z drugiego układu. Dodatkowo przy przepelnieniu widok staje się przewijalny.

Z tego powodu konieczność uzupełnienia tego elementu pojawia się w przypadku ListFragment oraz TutorialFragment. Stąd w układzie XML w Relative Layout w górnej części okna umieszczono element ImageView (zawierający grafikę) oraz TextView (zawierający nazwę aplikacji). Pozostałe miejsce przeznaczono na kontener RecyclerView.

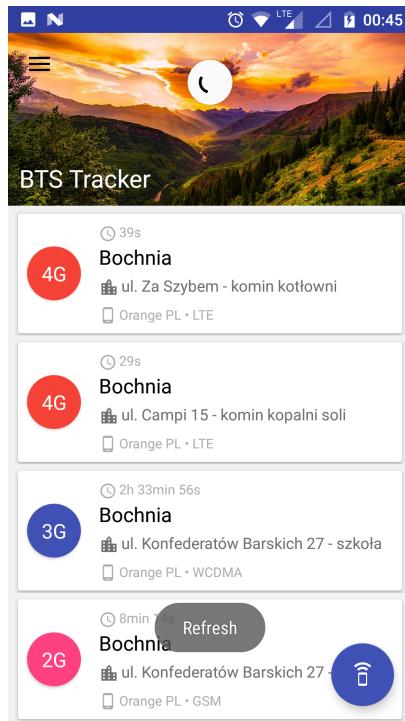
Całość opakowano jeszcze w SwipeRefreshLayout, który pozwoli na sprawne odświeżanie danych, polegające klasycznie na przeciągnięciu palcem z góry widoku.

Miejsce przeznaczone na kontener wypełniane będzie elementami list\_view\_item, przedstawiającymi dane dotyczące poszczególnych BTSów. Najbardziej wyróżniono technologię, z jakiej dana infrastruktura korzysta. W kolejnym (charakterystycznym dla nowoczesnych aplikacji) zgodnych z wytycznymi Material Design) umieszczono tekst: 2G, 3G lub 4G. Dodatkowo, kolor wypełnienia kół różni się w zależności od technologii.

Z prawej strony elementu graficznego wypisano dane. Uznano, że na wyróżnienie zasługuje nazwa miejscowości, w której zlokalizowana jest stacja nadawcza. Powyżej jest wyświetlany sumaryczny czas podłączenia telefonu komórkowego do stacji, a poniżej szczegółowa lokalizacja (np. nazwa ulicy, czy budynku) oraz nazwa operatora komórkowego i technologii. Elementy opatrzone prostymi pictogramami.

### 3.2.5.2. ListFragment – Implementacja

Po uruchomieniu fragmentu na ekranie pojawia się opisany wyżej układ elementów. Następnie definiowana jest referencja do kontenera RecyclerView i jest on konfigurowany. W szczególności przypisywany jest adapter widoku listy definiowany jako klasa dziedzicząca po RecyclerView.Adapter.



Rys. 3.11. Widok listy można intuicyjnie odświeżyć klasycznym ruchem przeciagnięcia palcem z góry ekranu.

Obiekty SwipeRefreshLayout odpowiadającemu za odświeżenie układu w momencie przeciagnięcia palcem z góry przypisywany jest obiekt słuchacza, którym jest właśnie instancja klasy ListFragment.

Klasa implementuje interfejs SwipeRefreshLayout.OnRefreshListener. Odświeżenie powoduje oczywiście wyświetlenie nowowczytanych elementów z listy. Wyświetlana jest także informacja „Refresh” i przewijające się kółko systemu Android.

Przy projektowaniu aplikacji zdecydowano się zrezygnować z dynamicznego odświeżania elementów, ponieważ to nie przeglądanie widoków, a usługa działająca w tle stanowi główną funkcjonalność programu. Samo dynamiczne odświeżanie mogłoby przeszkadzać podczas analizy tekstu czy mapy, dla tego to użytkownik decyduje, czy chce zobaczyć nowe dane, a czynność, jaką musi w tym celu wykonać jest intuicyjna i podstawowa. Oczywiście podczas przełączania fragmentu dane wczytują się automatycznie.

### 3.2.5.3. Pakiet list: klasy ListAdapter i ViewHolder

Klasa ViewHolder dziedziczy po RecyclerView.ViewHolder z Android Support Library. Przechowuje referencje do obiektów z układu list\_view\_item: do przycisku (Button) obudowującego koło z tekstem symbolizującą daną technologię oraz do obiektów TextView opisujących: miejscowości, lokalizację, czas połączenia do stacji nadawczej, nazwę operatora i nazwę technologii. Dodatkowa metoda setNetworkMode() odpowiada za zmianę koloru i tekstu przycisku.

Klasa ListAdapter dziedziczy po generycznej kolekcji RecyclerView.Adapter<>, a parametrem tej kolekcji może być tylko obiekt ViewHolder. Dzięki temu zabiegowi od strony implementacji wystarcza nadpisanie metody onBindViewHolder(), która jako parametry przyjmuje właśnie obiekt ViewHolder i jego pozycję. Obiekt ListAdapter ma dostęp do listy BtsList poprzezinstancję BtsManager. Zadaniem onBindViewHolder() jest pobranie danych z elementu na odpowiedniej pozycji listy i przypisanie ich do referencji przechowywanych w obiekcie wzorca ViewHolder.

### 3.2.6. Klasa MapFragment

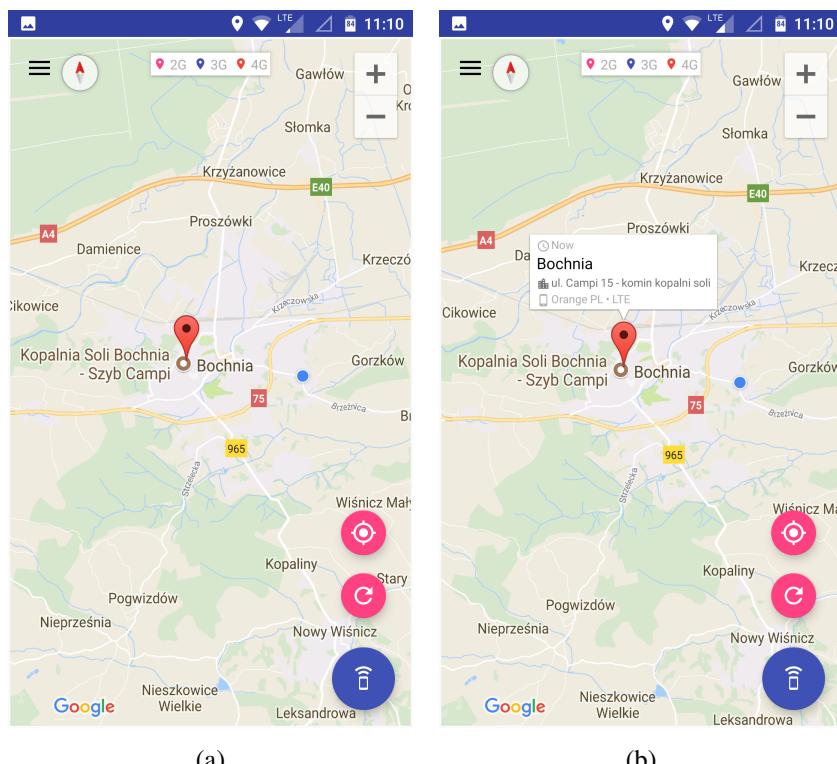
Widok mapy jest ważnym elementem programu „BTS Tracker”. Zadaniem GUI aplikacji jest naniesienie na mapę lokalizacji, do których urządzenie było podłączone i jeżeli dostępna jest także informacja o lokalizacji telefonu komórkowego (uruchomiony nadajnik GPS lub inna opcja lokalizacji), dodatkowe zaznaczenie tej pozycji.

Przy realizacji zadania wykorzystano pakiet com.google.android.gms.maps ze zbioru „Google APIs for Android”, a więc dostarczony przez producenta systemu operacyjnego [6]. Dzięki temu fragment wymaga tylko niewielkiej konfiguracji ze strony dewelopera. Przyjęto następujące założenia:

- Stacje nadawcze mają być oznaczane za pomocą klasycznych markerów (znaczników) Google Maps. Powinny się różnić jedynie kolorami, zależnie od technologii: 2G – różowy, 3G – niebieski, 4G – czerwony.
- Może zdarzyć się sytuacja, że kilka stacji nadawczych pojawi się w tej samej lub znacznie zblżonej lokalizacji. Taka sytuacja nie jest rzadka i dotyczy BTSów różnych technologii. Aby znaczniki się nie pokrywały, markery mogą być obrócone o +/- 30 stopni.
- U góry widoku powinien „unosić się” nad powierzchnią mapy niewielki układ będący legendą – opisujący znaczenie kolorów.
- Lokalizacja telefonu komórkowego ma być oznaczona charakterystyczną dla systemu Android niebieską kropką, a jeżeli znana jest tylko lokalizacja przybliżona – okręgiem.

- Powinny zostać zachowane funkcjonalności pakietu maps takie jak przybliżanie, oddalanie, obracanie z wykorzystaniem dwóch palców. W prawym górnym rogu należy umieścić przyciski plus i minus odpowiedzialne za zoom, a lewej niewielki kompas (widoczny tylko po obróceniu).
- Należy nanieść dwa dodatkowe przyciski: Spot oraz Refresh. Przycisk Spot powinien umożliwiać animowane przejście tak, aby wyśrodkować widok na stacji nadawczej, do której obecnie podłączony jest telefon komórkowy. Z kolei zadaniem przycisku Refresh ma być odświeżenie widoku – pobranie nowych danych. Aby nie rozpraszać użytkownika i nie zaciemniać widoku, nowe markery nie są dodawane automatycznie, a jedynie przy otwarciu fragmentu i odświeżeniu układu. Zgodnie z wytycznymi Material Design ilość obiektów typu FloatingActionButton może być większa niż jeden, ale dodatkowe powinny odróżniać się od głównego (Scanning) na przykład rozmiarem i kolorem.

### 3.2.6.1. Układ elementów



**Rys. 3.12.** Rysunek (a) przedstawia podstawowy widok mapy z naniesionym jednym elementem nadawczym i niebieską kropką oznaczającą pozycję telefonu komórkowego. Widać kompas, legendę, przyciski regulacji przybliżenia oraz przyciski Spot i Refresh. Na rysunku (b) po kliknięciu markera pojawia się jego opis. Układ opisu jest analogiczny do tego z ListFragment.

W pliku XML układ elementów fragment\_map składa się z elementów:

- Fragmentu SupportMapFragment, który rozciąga się na całą długość i szerokość widoku.
- Układu RelativeLayout spinającego legendę mapy. Układ jest umieszczony w górnej części mapy. Składa się z trzech zestawów: Button (w tym przypadku: ikona) + TextView (pole tekstowe).

- Dwóch przycisków typu FloatingActionButton. Rozmiar przycisków to mini.

Całość spięta jest układem RelativeLayout.

Plik XML układu map\_view\_item służy do wyświetlenia informacji o stacji nadawczej. Jest analogiczny do pojedynczego elementu listy z ListFragment (list\_view\_item). Podobnie jak wcześniej wyróżniono nazwę miejscowości, w której zlokalizowana jest stacja nadawcza. Wyświetlono także sumaryczny czas podłączenia telefonu komórkowego do stacji, szczegółową lokalizację oraz nazwę operatora komórkowego i nazwę technologii. Tutaj również elementy opatrzone prostymi piktogramami. Całość (tekst i grafiki) jest nieco mniejsza.

### 3.2.6.2. Implementacja

W metodzie onViewCreated() następuje przypisanie referencji do obiektu SupportMapFragment odpowiadającego widokowi z pakietu maps oraz referencji do obiektu BtsManager zawierającego listę stacji nadawczych. Definiowane są także obiekty słuchacza (ang. listener) do przycisków Spot (animowane przejście do aktualnie podłączonej stacji nadawczej) i Refresh (ponowne załadowanie markerów i wyświetlenie mapy).

Za detale odpowiada metoda onMapReady() z interfejsu OnMapReadyCallback, który klasa MapFragment implementuje. Konfiguruje ona położenie kompasa i przycisków przyplążenia (plus i minus) oraz obecność niebieskiej kropki wskazującej aktualną lokalizację. Następnie definiuje w klasie anonimowej adapter okna informacji (GoogleMap.InfoWindowAdapter), czyli okienka, które pojawia się nad markerem i przedstawia informacje o BTSie. Do elementów widoku (TextView) przypisywane są informacje z klasy Bts przekazywane przez tzw. snippet (pol. skrawek). Do okna przypisywany jest również słuchacz, który powoduje, że dymek po naciśnięciu znika.

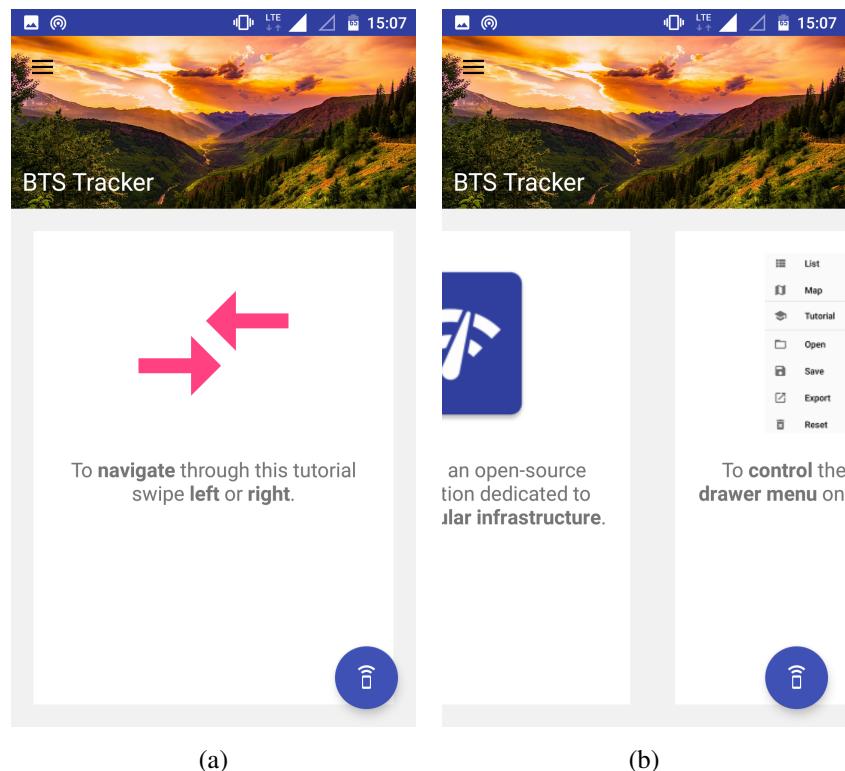
Wreszcie z klasy BtsManager pobierany jest obiekt ArrayList<MarkerOptions>. Program iterując po tej kolekcji umieszcza na mapie kolejne znaczniki wyświetlając okno informacji dla najbardziej aktualnej stacji badawczej.

### 3.2.7. Klasa TutorialFragment

Przy realizacji zadania dołożono wszelkich starań, aby aplikacja była przejrzysta i intuicyjna. W celu szybkiego przedstawienia użytkownikowi możliwości programu, przygotowano samouczek (tutorial). Aby nie przytłoczyć nadmiarem informacji, zdecydowano się je stopniować i łączyć z ilustracjami stąd TutorialFragment składa się z wielu mniejszych widoków: obiektów TutorialElemFragment. Każdy taki element to jedno-dwa zdania tekstu (sformatowanego za pomocą znaczników HTML) oraz odpowiednia grafika (o stałym rozmiarze).

Sam układ XML jest analogiczny do tego z ListFragment. Kontener RecyclerView zastąpiono przewijalnym elementem ViewPager. Wewnątrz niego wyświetlane są elementy Button (grafika) i TextView (tekst sformatowany), opakowane w RelativeLayout.

W implementacji wykorzystano klasę wewnętrzną rozszerzającą FragmentPagerAdapter z pakietów systemu Android. Jego metoda getItem() określa, że przewijalnymi elementami są obiekty klasy TutorialElemFragment dziedziczącej po klasie Fragment. Te obiekty w metodzie onCreateView() przypisują tekst i grafiki do elementów widoku.



**Rys. 3.13.** Rysunek (a) przedstawia typowy widok TutorialFragment. Zrzut ekranu (b) został wykonany w momencie przesunięcia palcem, czyli podczas przełączania elementów tutorialu.

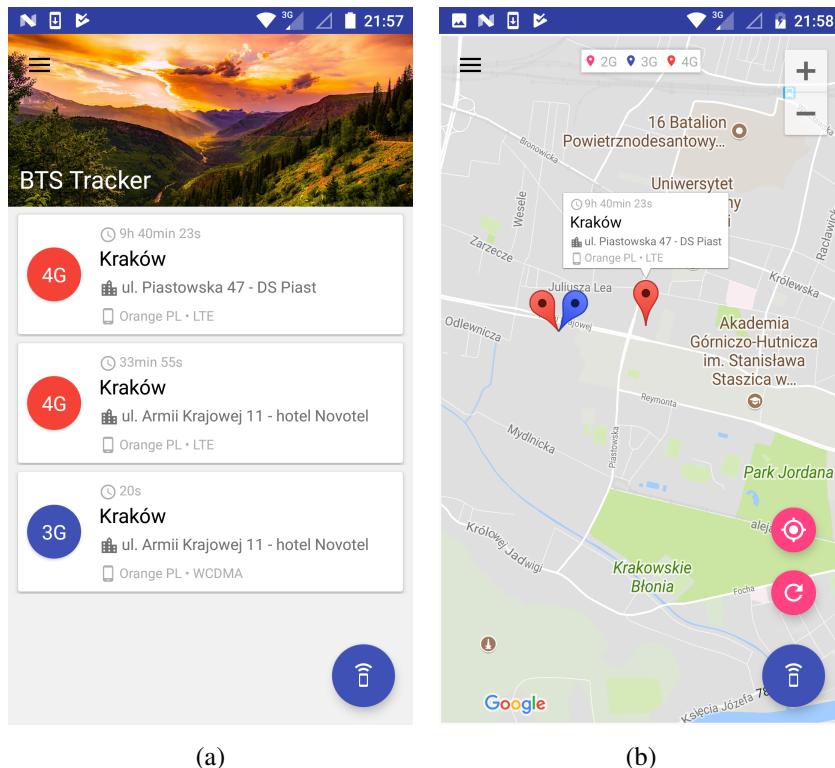


## 4. Rezultaty testów

### 4.1. Scenariusze testowe

#### 4.1.1. Wielogodzinne działanie w tle

Założono, że czas trwania pierwszego z testów wyniesie około 10 godzin. Po uruchomieniu skanowania w aplikacji „BTS Tracker” telefon komórkowy został zablokowany i położony w jednym miejscu wewnątrz budynku.



Rys. 4.1. Rezultaty testu według scenariusza: wielogodzinne działanie w tle. Widok listy (a) i mapy (b).

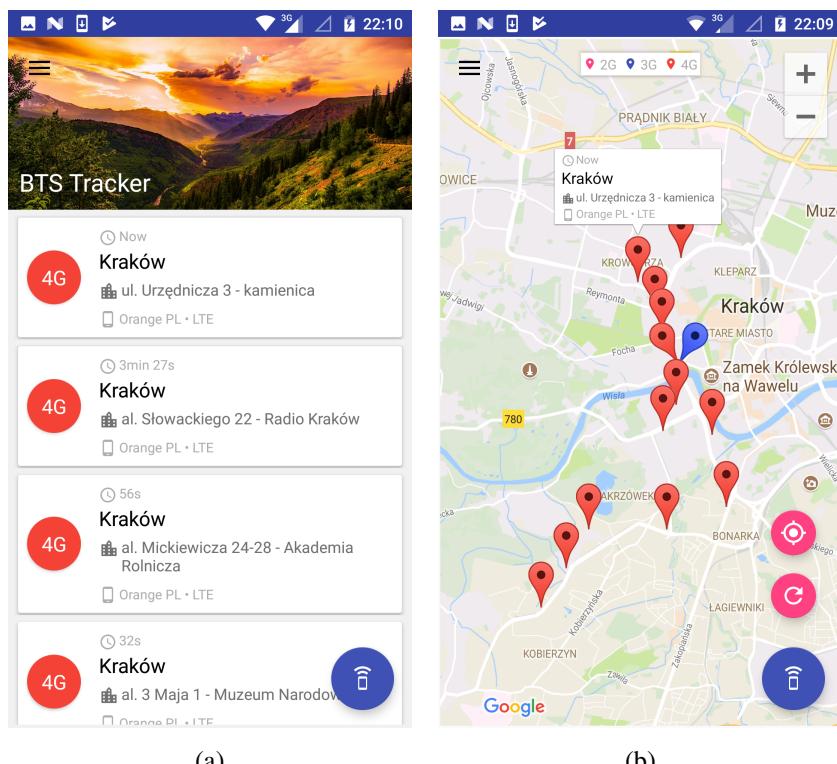
Przez cały okres pracy w tle urządzenie podłączyło się do trzech stacji bazowych, przy czym przez niemal 95% czasu pozostawało połączone z najbliższą stacją przekaźnikową. Można również zauważyć, że miało miejsce przynajmniej jedno przejście do sieci niższej generacji.

**Tabela 4.1.** Zebrane informacje. Scenariusz: wielogodzinne działanie w tle.

Miejscowość	Lokalizacja	Technologia	Czas połącz.
Kraków	ul. Piastowska 47 - DS Piast	LTE	9h 40min 23s
Kraków	ul. Armii Krajowej 11 - hotel Novotel	LTE	33min 55
Kraków	ul. Armii Krajowej 11 - hotel Novotel	WCDMA	20s

#### 4.1.2. Poruszanie się po mieście

Pomiaru dokonano w Krakowie na trasie: Pętla Czerwone Maki – Centrum Kongresowe ICE – Budynek C1 AGH. Telefon komórkowy został umieszczony w samochodzie osobowym, po czym uruchomiono skanowanie i zminimalizowano aplikację. Równolegle na pierwszym planie działała nawigacja Google Maps.



**Rys. 4.2.** Rezultaty testu według scenariusza: poruszanie się po mieście. Widok listy (a) i mapy (b).

Przejazd trwał łącznie 37 minut. Przez ten czas telefon połączył się z czternastoma stacjami przekaźnikowymi. Czas podłączenia wahał się w przedziale od kilkudziesięciu sekund do kilku minut. W mieście, gdzie infrastruktura sieci LTE jest gęsta i zwarta, transfer danych (m.in. dla aplikacji nawigacji) przebiegał niemal wyłącznie z wykorzystaniem sieci 4G. Właśnie dzięki dużej gęstości elementów sieci z dużym prawdopodobieństwem można odtworzyć trasę podróży dzięki informacjom zarejestrowanym przez „BTS Tracker”.

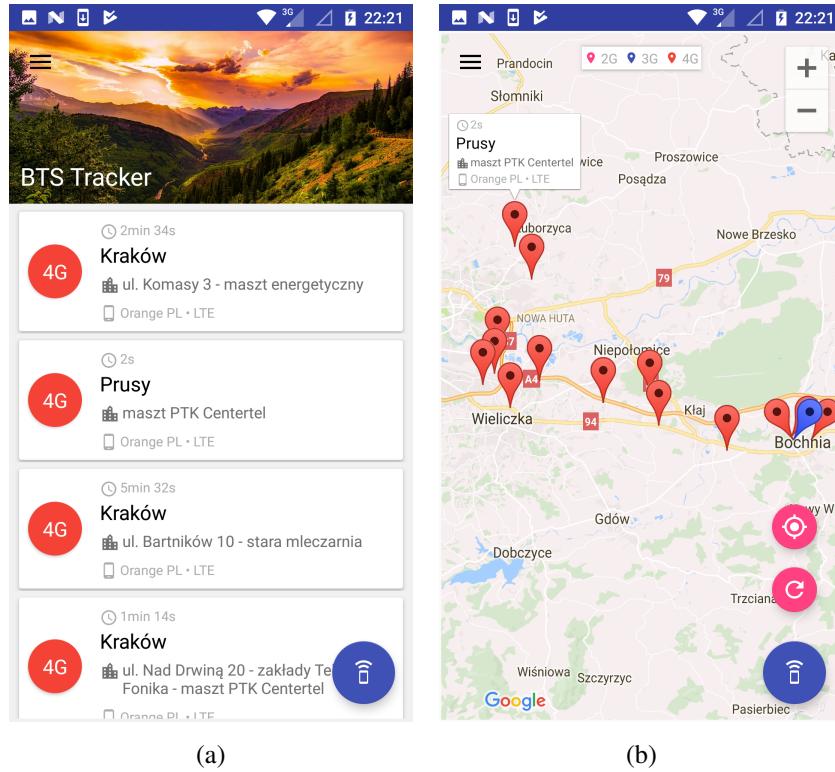
**Tabela 4.2.** Zebrane informacje. Scenariusz: poruszanie się po mieście.

Miejscowość	Lokalizacja	Technologia	Czas połącz.
Kraków	ul. Urzędnicza 3 - kamienica	LTE	Now
Kraków	al. Słowackiego 22 - Radio Kraków	LTE	3min 27s
Kraków	al. Mickiewicza 24-28 - Akademia Rolnicza	LTE	56s
Kraków	al. 3 Maja 1 - Muzeum Narodowe	LTE	32s
Kraków	ul. Konopnickiej 28 - hotel Sofitel	LTE	4min 23s
Kraków	al. Krasińskiego 1/3 - DH Jubilat	LTE	5min 3s
Kraków	al. Krasińskiego 1/3 - DH Jubilat	WCDMA	42s
Kraków	ul. Konfederacka 6 - kościół	LTE	4min 20s
Kraków	ul. Szwedzka 27 - Zakład Opieki Zdrowotnej	LTE	1min 12s
Kraków	ul. Wadowicka 3 - Przedsiębiorstwo Hydrogeologiczne Hydropol	LTE	3min 42s
Kraków	ul. Kapelanka 60 - ośrodek rec.-szkol. szpitala im. J. Dietla	LTE	26s
Kraków	ul. Gronostajowa 7 - Wydział Biotechnologii UJ	LTE	4min 17s
Kraków	ul. Łojasiewicza 6 - budynek Wydziału Matematyki i Informatyki UJ	LTE	54s
Kraków	ul. Bobrzyńskiego 14 - budynek Jagiellońskiego Centrum Innowacji	LTE	6min 48s

#### 4.1.3. Przejazd autostradą

W kolejnym scenariuszu również wykorzystano samochód osobowy. Pomiar przeprowadzono na trasie Bochnia – Kraków Bieżanów – Kraków Nowa Huta poruszając się z prędkością autostradową drogami A4 oraz S7. W tym czasie skanowanie aplikacji „BTS Tracker” działało w tle.

Można zauważyć, że stacje bazowe nie są rozmieszczone wzdłuż stosunkowo nowych dróg, jakimi są A4 i S7, lecz znajdują się w okolicznych miejscowościach i są rozlokowane rzadziej niż w dużym mieście. Co ciekawe, aplikacja zarejestrowała połączenie do masztu w Prusach, który znajduje się w dużej odległości od trasy przejazdu. Trwało ono jednak jedynie dwie sekundy – musiało być spowodowane chwilową zmianą warunków radiowych.



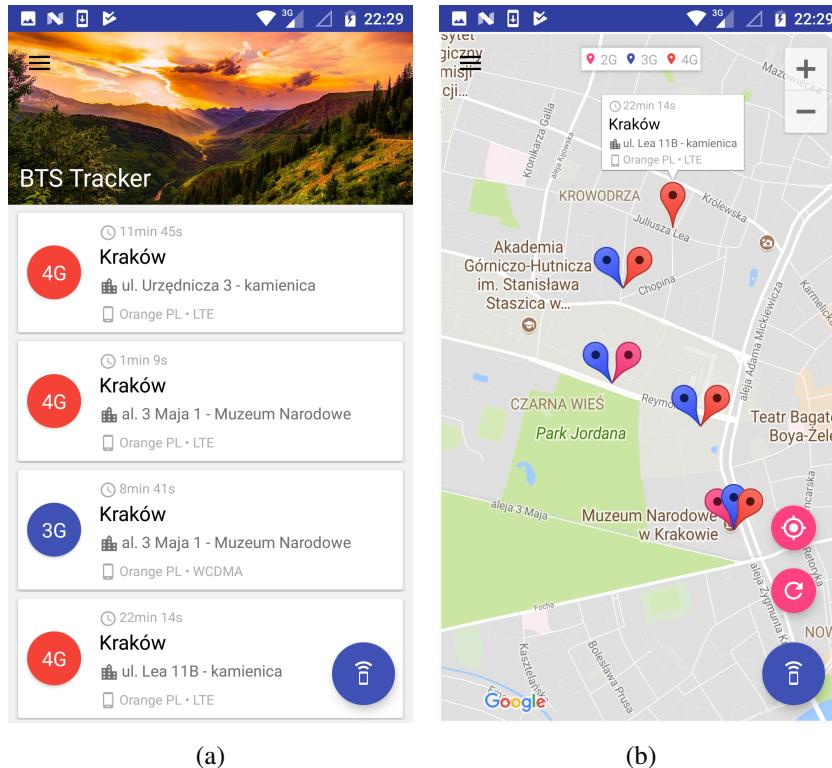
**Rys. 4.3.** Rezultaty testu według scenariusza: przejazd autostradą. Widok listy (a) i mapy (b).

**Tabela 4.3.** Zebrane informacje. Scenariusz: przejazd autostradą.

Miejscowość	Lokalizacja	Technologia	Czas połącz.
Kraków	ul. Komasy 3 - maszt energetyczny	LTE	2min 34s
Prusy	maszt PTK Centertel	LTE	2s
Kraków	ul. Bartników 10 - stara mleczarnia	LTE	5min 32s
Kraków	ul. Nad Drwiną 20 - zakłady Tele-Fonika - maszt PTK Centertel	LTE	1min 14s
Zagórze	maszt Emitel	LTE	1min 30s
Kraków	ul. Drożdżowa 5 - zakłady drożdżowe - komin	LTE	1min 53s
Wieliczka	ul. Grottgera 58 - HERZ	LTE	29s
Węgrzce Wielkie	maszt P4	LTE	1min 10s
Zagórze	maszt Emitel	LTE	3min 3s
Dąbrowa	wieża obserwacyjna przeciwpożarowa	LTE	15s
Grodkowice	maszt Plusa	LTE	12min 15s
Moszczenica	maszt PTK Centertel	LTE	4min 8s
Bochnia	ul. Za Szybem - komin kotłowni	LTE	39s
Bochnia	ul. Campi 15 - komin kopalni soli	LTE	20s
Bochnia	ul. Campi 15 - komin kopalni soli	WCDMA	3min 24s
Bochnia	ul. Konfederatów Barskich 27 - szkoła	LTE	7min 40s

#### 4.1.4. Przemieszczanie się wewnątrz budynku

Test zrealizowano wewnątrz zespołu budynków C1 – C2 – C3 Akademii Górnictwo-Hutniczej w Krakowie. Trwał 2h 15 minut, z czego przez 90 minut telefon pozostawał niemal w tym samym miejscu.



Rys. 4.4. Rezultaty testu według scenariusza: przemieszczanie się wewnątrz budynku.  
Widok listy (a) i mapy (b).

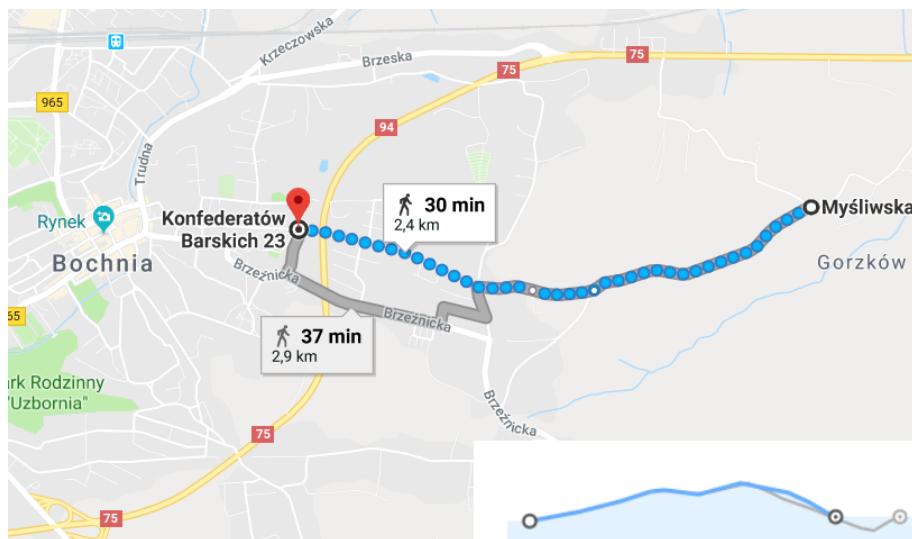
Mimo tego aplikacja zarejestrowała połączenia z dziesięcioma okolicznymi stacjami BTS. Z powodu gorszych warunków radiowych, jakie panują wewnątrz, urządzenie łączyło się z nadajnikami wszystkich dostępnych generacji.

Tabela 4.4. Zebrane informacje. Scenariusz: przemieszczanie się wewnątrz budynku.

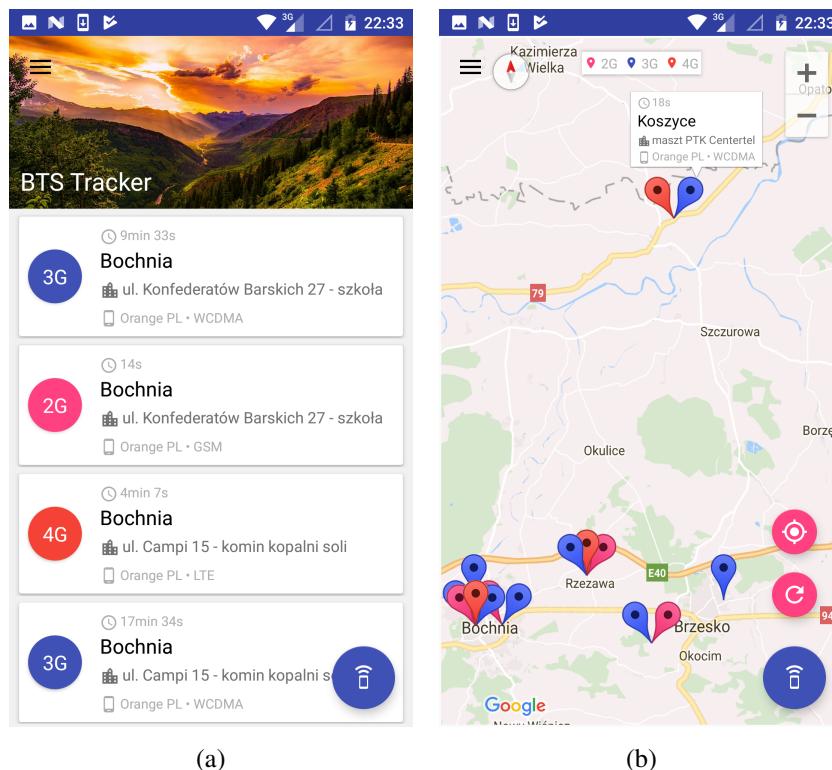
Miejscowość	Lokalizacja	Technologia	Czas połącz.
Kraków	ul. Urzędnicza 3 - kamienica	LTE	11min 45s
Kraków	al. 3 Maja 1 - Muzeum Narodowe	LTE	1min 9s
Kraków	al. 3 Maja 1 - Muzeum Narodowe	WCDMA	8min 41s
Kraków	ul. Lea 11B - kamienica	LTE	22min 14s
Kraków	ul. Urzędnicza 3 - kamienica	WCDMA	1min 3s
Kraków	al. Mickiewicza 24-28 - Akademia Rolnicza	LTE	1h 11min 55s
Kraków	al. Mickiewicza 24-28 - Akademia Rolnicza	WCDMA	17min 20s
Kraków	ul. Reymonta 11 - akademik Nawojka	WCDMA	2min 22s
Kraków	al. 3 Maja 1 - Muzeum Narodowe	GSM	9s
Kraków	ul. Reymonta 11 - akademik Nawojka	GSM	8s

#### 4.1.5. Spacer po pagórkowatym terenie

Ostatni scenariusz testowy zrealizowano poza dużym miastem. Telefon komórkowy znajdował się w kieszeni użytkownika, który przez godzinę spacerował na drodze z Bochni do sąsiedniego Gorzkowa i z powrotem.



Rys. 4.5. Trasa testu według scenariusza: spacer po pagórkowatym terenie. W prawym dolnym rogu przedstawiono różnicę wzniesień.



Rys. 4.6. Rezultaty testu według scenariusza: spacer po pagórkowatym terenie. Widok listy (a) i mapy (b).

Infrastruktura telekomunikacyjna w górzystym krajobrazie stanowi dużo większe wyzwanie, dlatego trasę wybrano tak, aby dwukrotnie pokonać wznieśenie, które znajduje się między miejscowościami.

Test potwierdził, że wznieśenie jest przeszkodą dla sygnału telekomunikacyjnego. Pomimo że miejscowości Gorzków sąsiaduje z Bochnią, telefon użytkownika podłączał się do stacji przekaźnikowych położonych między innymi w odległym Brzesku i w jeszcze bardziej oddalonych Koszycach.

Podczas spaceru urządzenie wykorzystywało infrastrukturę wielu możliwych generacji – warunki radiowe musiały być trudne. W tej sytuacji niemożliwe jest wprost odtworzenie trasy, jaką poruszał użytkownik.

**Tabela 4.5.** Zebrane informacje. Scenariusz: spacer po pagórkowatym terenie.

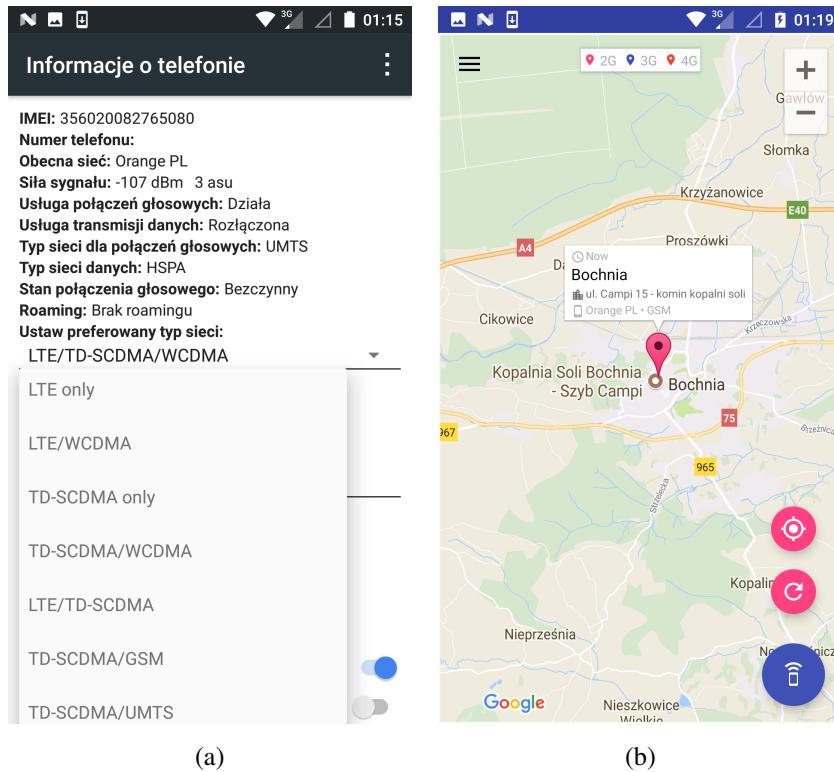
Miejscowość	Lokalizacja	Technologia	Czas połącz.
Bochnia	ul. Konfederatów Barskich 27 - szkoła	WCDMA	9min 33s
Bochnia	ul. Konfederatów Barskich 27 - szkoła	GSM	14s
Bochnia	ul. Campi 15 - komin kopalni soli	LTE	4min 7s
Bochnia	ul. Campi 15 - komin kopalni soli	WCDMA	17min 34s
Bochnia	ul. Campi 15 - komin kopalni soli	GSM	52s
Bochnia	ul. Za Szybem - komin kotłowni	GSM	55s
Bochnia	ul. Za Szybem - komin kotłowni	WCDMA	16s
Rzezawa	maszt Plusa	WCDMA	13min 42s
Jasień - Granice	maszt PTK CenterTel	WCDMA	5min 50s
Bochnia	ul. Partyzantów - maszt T-Mobile	WCDMA	2s
Rzezawa	maszt Plusa	LTE	54s
Koszyce	maszt PTK CenterTel	LTE	6s
Rzezawa	maszt Plusa	GSM	54s
Koszyce	maszt PTK CenterTel	WCDMA	18s
Brzesko	ul. Starowiejska 28 - CAN-PACK	WCDMA	5min 36s
Łopoń	Łopoń 46 - maszt T-Mobile	WCDMA	1min 35s
Jasień - Granice	maszt PTK CenterTel	GSM	18s

## 4.2. Przypadki testowe

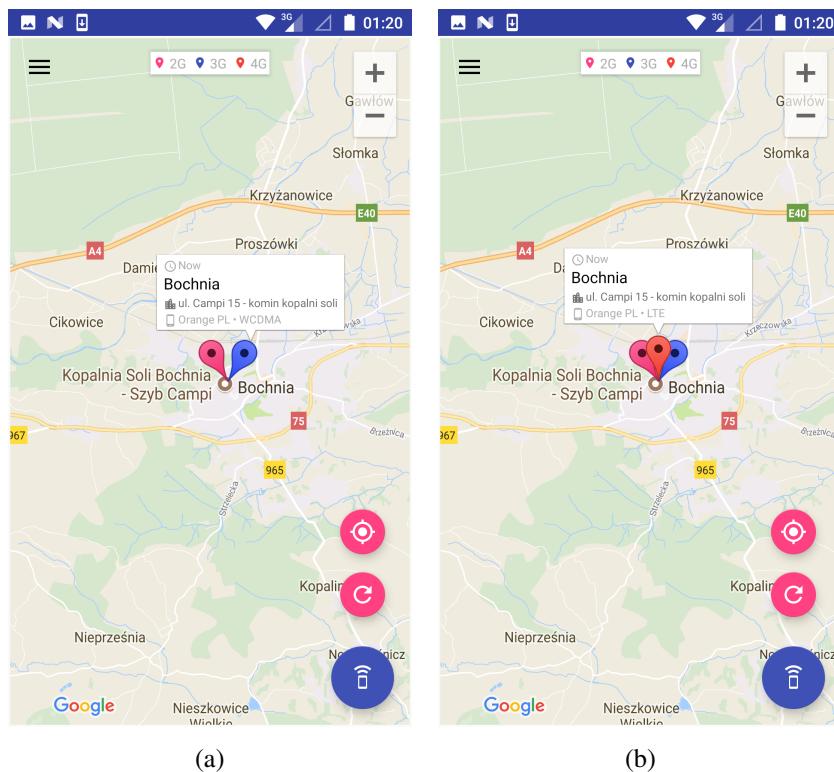
### 4.2.1. Więcej niż jeden BTS w danej lokalizacji

W rzeczywistości nie mamy do czynienia z jedną, ani dwoma, ale właściwie z trzema generacjami sieci telekomunikacyjnych, które się wzajemnie uzupełniają. Większość współczesnych telefonów jest przystosowana do odbierania sygnału na częstotliwościach wszystkich trzech technologii, dlatego bardzo prawdopodobną jest sytuacja, kiedy więcej niż jeden nadajnik znajduje się w tej samej lokalizacji.

Aby uniknąć niepraktycznego efektu pokrywania się markerów mapy wprowadzono dodatkową rotację o 30 stopni – odpowiada za nią klasa BtsManager. Niniejszy test ma za zadanie sprawdzić działanie tego mechanizmu w praktyce.



Rys. 4.7. Menu trybu testowania w telefonie z systemem Android (a) i pojedynczy BTS w danej lokalizacji zaznaczony na mapie rejestratora stacji bazowych (b).



Rys. 4.8. Obrót o 30 stopni markerów na mapie w sytuacji, gdy pojawia się drugi (a) i trzeci (b) BTS.

Do zarządzania telefonem wykorzystano ukryte menu „Testowanie” przygotowane przez projektantów systemu Android z myślą o serwisantach i deweloperach. Wykorzystując jedną z opcji menu, które pojawia się po wpisaniu kodu \*#\*#4636#\*#\*, zmuszono telefon komórkowy do podłączenia się kolejno do sieci: GSM, WCDMA i LTE.

**Tabela 4.6.** Zebrane informacje. Przypadek testowy: więcej niż jeden BTS w danej lokalizacji.

Miejscowość	Lokalizacja	Technologia	Czas połącz.
Bochnia	ul. Campi 15 - komin kopalni soli	LTE	Now
Bochnia	ul. Campi 15 - komin kopalni soli	WCDMA	48s
Bochnia	ul. Campi 15 - komin kopalni soli	GSM	41s

#### 4.2.2. Zapis do pliku i odczyt zapisanych informacji

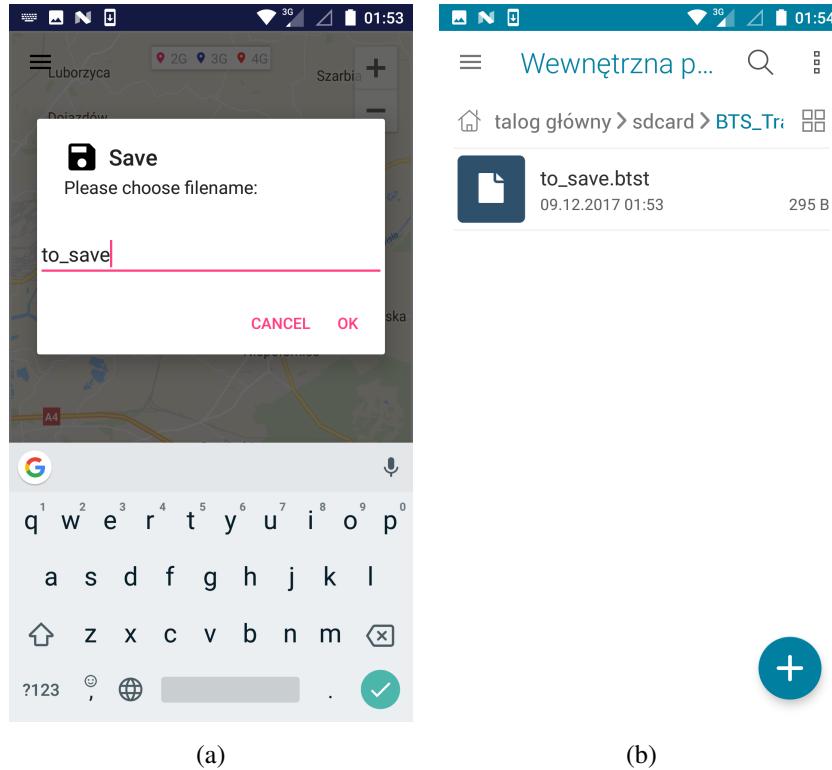
Po wielogodzinnej akwizycji danych postanowiono użyć opcji Save zapisując zebrane informacje do pliku BTST rozdzielające je separatorami.

Plik z zapisanymi danymi ma postać:

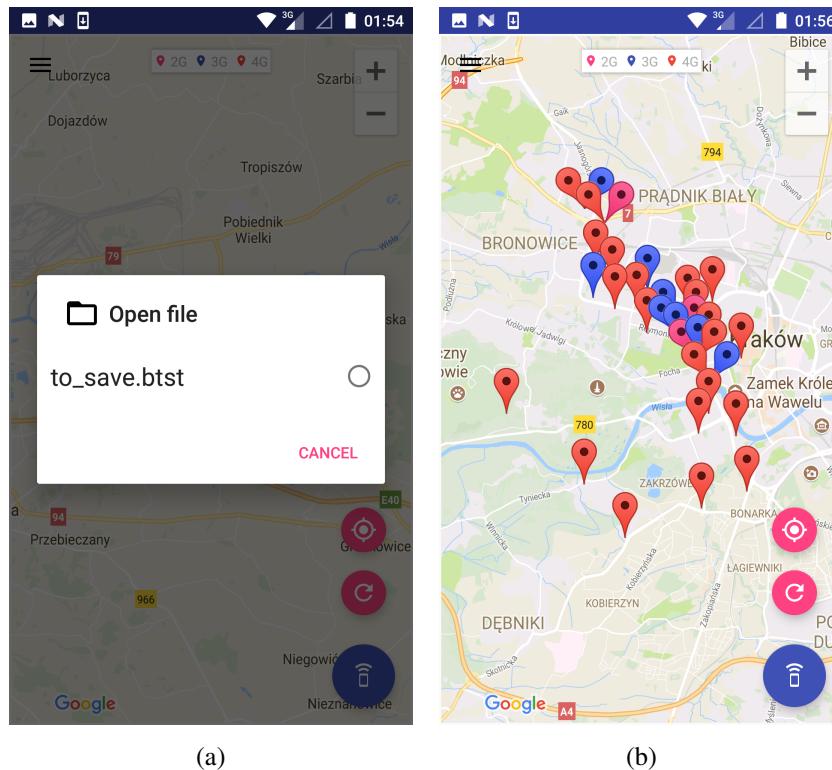
```
4; Kraków; ul. Wadowicka 3 – Przedsiębiorstwo Hydrogeologiczne Hydropol; Orange PL;LTE;50.034444444;19.938888889;222552;3 min 42 s ;0
4; Kraków; ul. Szwedzka 27 – Zakład Opieki Zdrowotnej; Orange PL;LTE;50.045833333;19.924166667;72661;1 min 12 s ;0
4; Kraków; ul. Konfederacka 6 – kościół; Orange PL;LTE;50.049722222;19.927222222;260214;4 min 20 s ;0
3; Kraków; al. Krasińskiego 1/3 – DH Jubilat; Orange PL;WCDMA;50.055833333;19.927777778;42261;42 s ;30
4; Kraków; al. Krasińskiego 1/3 – DH Jubilat; Orange PL;LTE;50.055833333;19.927777778;30395;5 min 3 s ;-30
4; Kraków; al. Słowackiego 22 – Radio Kraków; Orange PL;LTE;50.071666667;19.928333333;207170;3 min 27 s ;0
2; Kraków; ul. Reymonta 11 – akademik Nawojska; Orange PL;GSM;50.065;19.917777778;8231;8 s ;30
2; Kraków; al. 3 Maja 1 – Muzeum Narodowe; Orange PL;GSM;50.060277778;19.923888889;9175;9 s ;-30
3; Kraków; ul. Reymonta 11 – akademik Nawojska; Orange PL;WCDMA;50.065;19.917777778;142214;2 min 22 s ;-30
3; Kraków; al. Mickiewicza 24–28 – Akademia Rolnicza; Orange PL;WCDMA;50.063611111;19.922222222;1040659;17 min 20 s ;-30
3; Kraków; ul. Urzędnicza 3 – kamienica; Orange PL;WCDMA;50.068055556;19.918333333;63892;1 min 3 s ;-30
4; Kraków; ul. Lea 11B – kamienica; Orange PL;LTE;50.07;19.920833333;1334615;22 min 14 s ;0
3; Kraków; al. 3 Maja 1 – Muzeum Narodowe; Orange PL;WCDMA;50.060277778;19.923888889;521440;8 min 41 s ;0
4; Kraków; al. 3 Maja 1 – Muzeum Narodowe; Orange PL;LTE;50.060277778;19.923888889;69537;1 min 9 s ;30
4; Kraków; ul. Urzędnicza 3 – kamienica; Orange PL;LTE;50.068055556;19.918333333;705668;11 min 45 s ;30
3; Kraków; ul. Podchorążych 2 – WSP; Orange PL;WCDMA;50.073888889;19.908611111;22173;22 s ;0
4; Kraków; ul. Bandtkiego 19 – budynek poradni; Orange PL;LTE;50.075555556;19.897777778;41023;41 s ;0
2; Kraków; ul. Conrada 63 – budynek biurowy; Orange PL;GSM;50.087222222;19.895555556;9682;9 s ;30
4; Kraków; ul. Radzikowskiego 152 – Instytut Fizyki; Orange PL;LTE;50.09;19.889444444;1715650;28 min 35 s ;-30
3; Kraków; ul. Radzikowskiego 152 – Instytut Fizyki; Orange PL;WCDMA;50.09;19.889444444;175617;2 min 55 s ;30
4; Kraków; ul. Zapolskiej 38 – Ahold; Orange PL;LTE;50.078888889;19.892777778;253076;4 min 13 s ;0
3; Kraków; ul. Lea 210 – Hydrokop – nowy biurowiec; Orange PL;WCDMA;50.0725;19.891944444;37172;37 s ;0
4; Kraków; ul. Armii Krajowej 11 – hotel Novotel; Orange PL;LTE;50.070277778;19.898611111;852460;14 min 12 s ;0
4; Kraków; ul. Conrada 63 – budynek biurowy; Orange PL;LTE;50.087222222;19.895555556;1397432;23 min 17 s ;-30
4; Kraków; ul. Piastowska 47 – DS Piast; Orange PL;LTE;50.070555556;19.905277778;43622680;12 h 7 min 2 s ;0
4; Kraków; ul. Reymonta 22 – żółty treningowy budynek Wisły; Orange PL;LTE;50.065555556;19.908333333;229418;3 min 49 s ;0
4; Kraków; al. Mickiewicza 24–28 – Akademia Rolnicza; Orange PL;LTE;50.063611111;19.922222222;4315394;1 h 11 min 55 s ;30
4; Kraków; ul. Konopnickiej 28 – hotel Sofitel; Orange PL;LTE;50.045277778;19.935555556;338096;5 min 38 s ;0
4; Kraków; ul. Kapelanka 60 – ośrodek rec.–szkol. szpitala im. J. Dietla; Orange PL;LTE;50.031111111;19.925;95988;1 min 35 s ;0
4; Kraków; Rynek Główny 15–16 – restauracja Wierzynek; Orange PL;LTE;50.060555556;19.937222222;20997;20 s ;0
4; Kraków; ul. Jodłowa 13 – Instytut Studiów Polonijnych i Etnicznych UJ; Orange PL;LTE;50.049722222;19.865555556;18183;18 s ;0
4; Kraków; ul. Ćwikliowa 1 – szkoła; Orange PL;LTE;50.035833333;19.889166667;143459;2 min 23 s ;0
4; Kraków; ul. Gronostajowa 7 – Wydział Biotechnologii UJ; Orange PL;LTE;50.025277778;19.901666667;261164;4 min 21 s ;0
```

Zbiera kolejno: numer technologii, miejscowości, szczegóły lokalizacji, nazwę operatora, szerokość i długość geograficzną, wartość timera wyrażoną zarówno liczbowo jak i w postaci łańcucha znaków oraz kąt rotacji.

Po powrocie do programu wybrano opcję Open i widoki z powrotem z powodzeniem wypełniły się danymi.



**Rys. 4.9.** Po zapisie (a) w aplikacji plik to\_save.btst można odnaleźć w systemie plików telefonu w katalogu BTS\_Tracker (b).



**Rys. 4.10.** Po otwarciu (a) wczytanie informacji do listy i mapy (b) przebiegło pomyslnie.

### 4.2.3. Wykorzystanie komputera do analizy danych

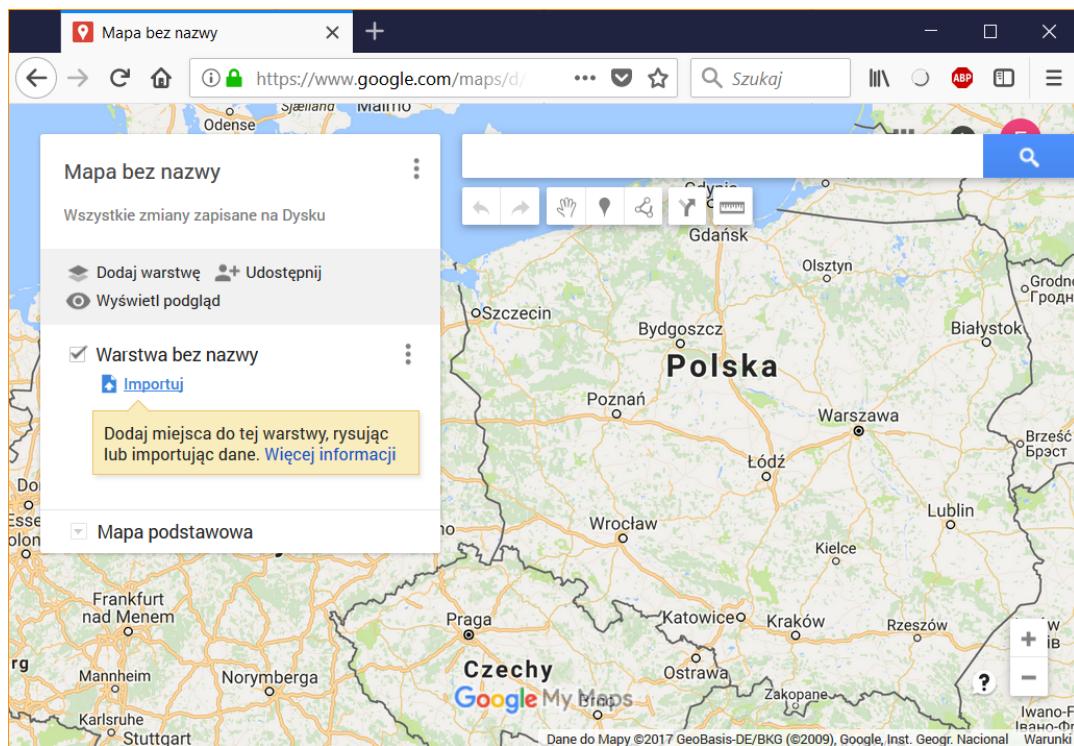
Informacje o krakowskiej infrastrukturze zebrane w poprzednim przypadku wyeksportowano do pliku KML, a następnie zapisano na Dysku Google. Plik zawiera wszystkie najważniejsze informacje o stracjach przekaźnikowych i ma postać:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name/>
    <description/>
    <Style id="bts2G">
      <IconStyle>
        <color>ff8140ff </color>
        <Icon>
          <href>http://www.gstatic.com/mapspro/images/stock/503-wht-blank_maps.png </href>
        </Icon>
        <hotSpot x="16" xunits="pixels" y="32" yunits="insetPixels"/>
      </IconStyle>
    </Style>
    <Style id="bts3G">
      <IconStyle>
        <color>ffb5513f </color>
        <Icon>
          <href>http://www.gstatic.com/mapspro/images/stock/503-wht-blank_maps.png </href>
        </Icon>
        <hotSpot x="16" xunits="pixels" y="32" yunits="insetPixels"/>
      </IconStyle>
    </Style>
    <Style id="bts4G">
      <IconStyle>
        <color>ff3643f4 </color>
        <Icon>
          <href>http://www.gstatic.com/mapspro/images/stock/503-wht-blank_maps.png </href>
        </Icon>
        <hotSpot x="16" xunits="pixels" y="32" yunits="insetPixels"/>
      </IconStyle>
    </Style>
    <Folder>
      <name>BTS_Tracker_bts_export.kml</name>
      <Placemark>
        <name>4G: Kraków</name>
        <description><![CDATA[ ul. Wadowicka 3 – Przedsiębiorstwo Hydrogeologiczne  
Hydropol<br>Orange PL * LTE<br>Attached: 3min 42s]]></description>
        <styleUrl>#bts4G</styleUrl>
        <Point>
          <coordinates>19.938888889,50.034444444,0</coordinates>
        </Point>
      </Placemark>
      <!-- dane o kolejnych BTSach -->
    </Folder>
  </Document>
</kml>
```

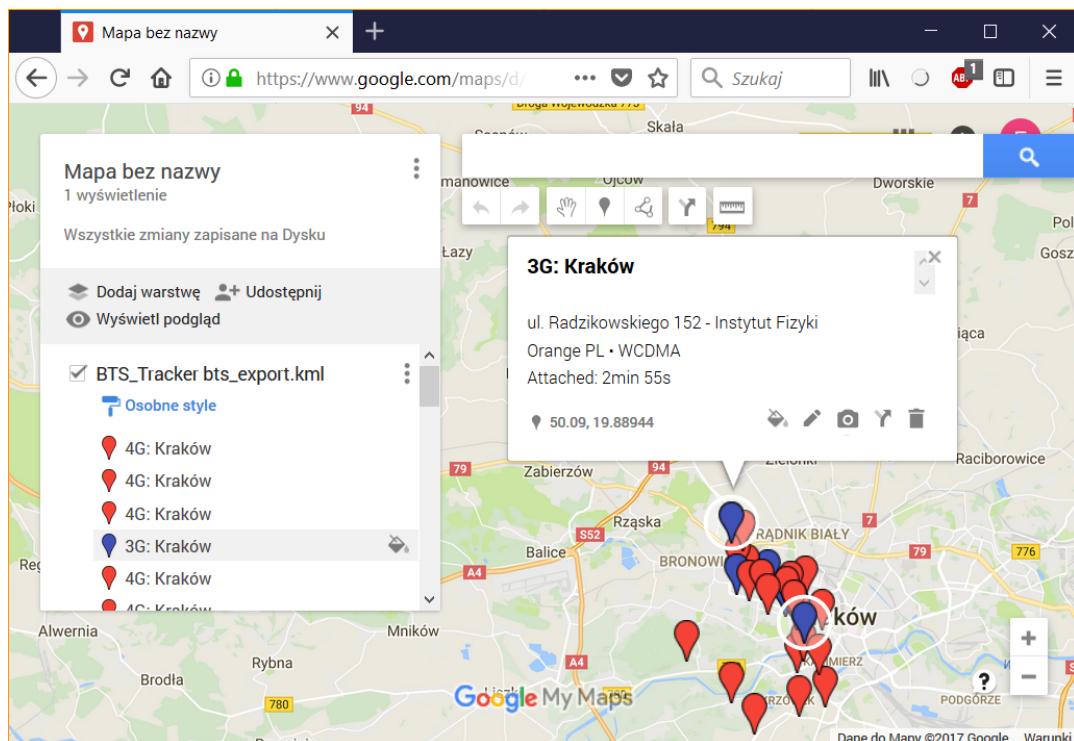
W kolejnym kroku otworzono serwis <http://google.com/mymaps> i wybrano opcję „Utwórz nową mapę”. Po kliknięciu Importuj i zalogowaniu się do Dysku Google wybrano zapisany plik KML z aplikacji „BTS Tracker”.

Na mapie Google Maps pojawiła się dodatkowa warstwa markerów w kolorach: różowym (infrastruktura 2G), niebieskim (3G) i czerwonym (4G). Po kliknięciu na marker pojawiały się informacje z aplikacji.

Sprawdzono także pozostałe funkcjonalności MyMaps. Narzędzie pozwala na wydruk, łączenie wielu warstw, rysowanie tras, wyszukiwanie i zaznaczanie miejsc oraz mierzenie odległości.



Rys. 4.11. Opcja Importuj w usłudze Google MyMaps.

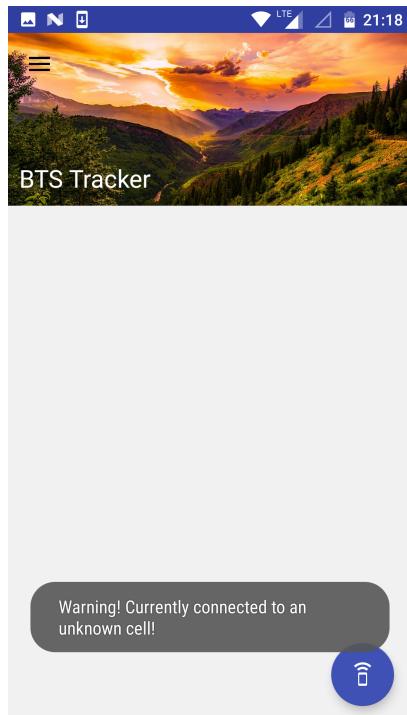


Rys. 4.12. Stacje przekaźnikowe z aplikacji „BTS Tracker” w usłudze Google MyMaps.

## 4.3. Przypadki szczególne

### 4.3.1. Brak informacji o stacji w tablicy danych

Test przeprowadzono tymczasowo modyfikując tablicę danych btsData.csv, aby odtworzyć sytuację, która może się zdarzyć w czasie codziennego wykorzystywania programu. Z bazy usunięto rekord identyfikujący stację bazową, do której był podłączony telefon komórkowy, przeprowadzono proces komplikacji projektu, a następnie ponownie zainstalowano i uruchomiono aplikację.



Rys. 4.13. Przypadek szczególny: brak informacji o stacji w tablicy danych.

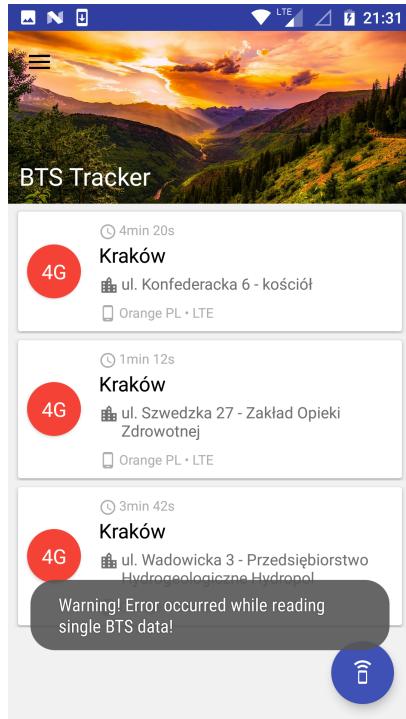
Po uruchomieniu skanowania „BTS Tracker” wyświetlił komunikat informujący użytkownika o tym, że urządzenie podłączone jest do stacji, która jest nieznana. Nowy element nie pojawił się w widoku listy, ani w widoku mapy.

### 4.3.2. Niepoprawne informacje w pliku wejściowym

W przypadku testowym wykorzystano plik BTST zapisany podczas działania programu. Plik otwarto w edytorze tekstu i z pierwszego wersu usunięto część tekstu sprowadzając go do następującej postaci:

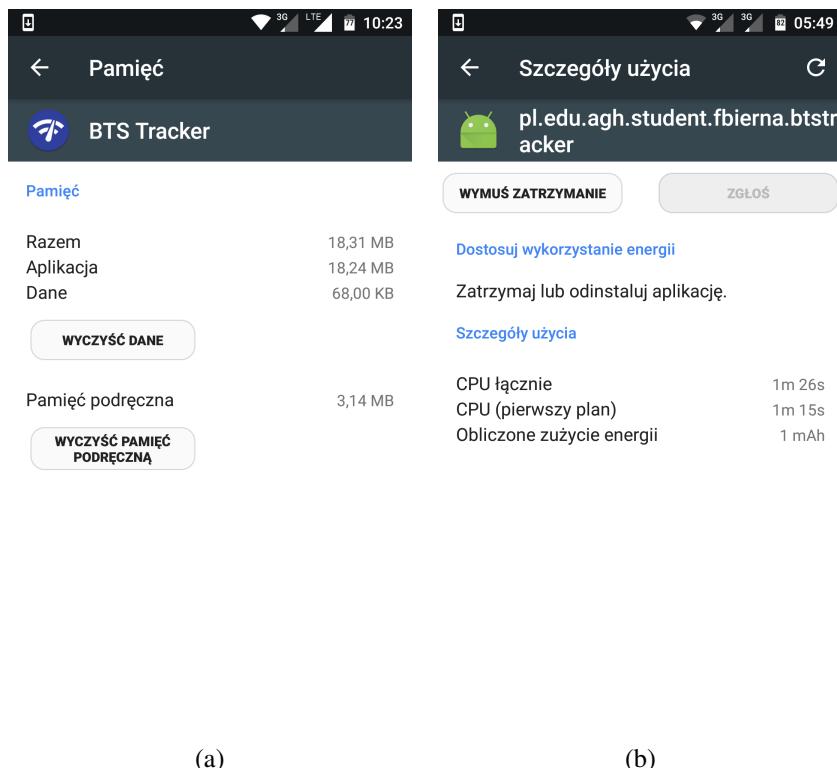
```
4;Kraków;ul. Bobrzańskiego 14 – budynek Jagiellońskiego Centrum Innowacji  
4;Kraków;ul. Wadowicka 3 – Przedsiębiorstwo Hydrogeologiczne Hydropol;Orange PL;LTE;50.034444444;19.938888889;222552;3 min 42 s;0  
4;Kraków;ul. Szwedzka 27 – Zakład Opieki Zdrowotnej;Orange PL;LTE;50.045833333;19.924166667;72661;1 min 12 s;0  
4;Kraków;ul. Konfederacka 6 – kościół;Orange PL;LTE;50.049722222;19.927222222;260214;4 min 20 s;0
```

Ostatecznie zaobserwowano, że program poprawnie poradził sobie z wyłapaniem wyjątku przy przetwarzaniu informacji wejściowych. Do widoków listy i mapy zostały dodane elementy od 2 do 4, a niepoprawny pierwszy element został pominięty. Dodatkowo aplikacja wyświetliła ostrzeżenie o błędzie, który pojawił się podczas odczytu danych pojedynczej stacji bazowej.



Rys. 4.14. Przypadek szczególny: niepoprawne informacje w pliku wejściowym.

## 4.4. Szczegóły techniczne



Rys. 4.15. Wykorzystanie pamięci (a) oraz energii (b) w aplikacji.

Rozmiar programu w porównaniu do popularnych aplikacji mobilnych okazuje się być nieduży. W systemie Android 7 Marshmallow „BTS Tracker” zajmuje mniej niż 20 MB pamięci. Kilkadziesiąt kilobajtów w pamięci danych to przede wszystkim zapamiętane uzyskane zezwolenia na dostęp do lokalizacji oraz na zapis i odczyt z plików. Dodatkowo, aby przyspieszyć działanie, program przechowuje m.in. grafiki w pamięci cache.

Dzięki zastosowanemu rozwiązaniu – implementacji usługi ScanService oraz obiektu słuchacza AttachListener, zużycie baterii okazało się być minimalne. Pamiętając o fakcie, że głównym zadaniem programu jest długoterminowe działanie w tle, przeprowadzono test uruchamiając aplikację, a następnie sprawdzając zużycie energii po piętnastu godzinach. Wyniosło ono około 1 mAh, czyli mniej niż jeden promil pojemności standardowej baterii.



## 5. Podsumowanie i wnioski

W ramach pracy inżynierskiej dyplomowej zrealizowano projekt rejestratora stacji bazowych. Aplikacja „BTS Tracker” spełnia założenia dotyczące funkcjonalności, estetyki i otwartoźródłowości.

Program w pierwszej kolejności realizuje identyfikację stacji przekaźnikowej. Przetwarzając dane z systemu Android otrzymuje ciąg liczb, które w jednoznaczny sposób wyróżniają dane urządzenie. Następnie z wykorzystaniem tablicy CSV przetwarza te dane na zasadzie look-up table otrzymując informacje, które może przekazać użytkownikowi. Ostatnim krokiem jest wizualizacja tych danych – w formie listy oraz mapy.



Rys. 5.1. Wyróżnione kolejne kroki w procesie działania aplikacji.

Rejestrator stacji bazowych sprawdził się podczas:

- wielogodzinnego działania w tle.
- poruszania się po mieście.
- przejazdu autostradą
- przemieszczania się wewnątrz budynku
- oraz spaceru po pagórkowatym terenie.

Równie dobrze poradził sobie również z odczytem i zapisem do pliku oraz eksportem danych do komputera.



## Bibliografia

- [1] Tom Farley. „Mobile telephone history”. W: *Teletronikk* 101.3/4 (2005), s. 22.
- [2] Sérgio Barros i in. „Evolution of long term narrowband-IoT”. W: *Electronics, Electrical Engineering and Computing (INTERCON), 2017 IEEE XXIV International Conference on*. IEEE. 2017, s. 1–4.
- [3] Assist Prof Spiros Louvros i Angelina Gkioni. „Voice Over LTE (VoLTE): Service Implementation and Cell Planning Perspective”. W: *System-Level Design Methodologies for Telecommunication*. Springer, 2014, s. 43–62.
- [4] Specyfikacja 3GPP.  
<http://www.3gpp.org/specifications/67-releases>.
- [5] Sklep Play.  
<https://play.google.com/store?hl=pl>.
- [6] Dokumentacja pakietów systemu Android.  
<https://developer.android.com/reference/packages.html>.
- [7] Keyhole Markup Language Reference.  
<https://developers.google.com/kml/documentation/kmlreference>.
- [8] Conder S. I. Delessio C. Darcey L. *Android Studio w 24 godziny. Wygodne programowanie dla platformy Android*. Wydanie IV. Wydawnictwo Helion, 2016.
- [9] Sploty. *E-UTRAN/LTE Signalling*. Wydawnictwo Sploty.
- [10] Cornell G. Horstmann C. S. *Java. Podstawy*. Wydanie IX. Wydawnictwo Helion, 2014.
- [11] BTSearch.  
<http://btsearch.pl>.

# Spis rysunków

3.1	Schemat blokowy ilustrujący klasy i ważniejsze metody biorące udział w procesie wykrywania stacji bazowych. Kolorem niebieskim wyróżniono klasy, a fioletowym metody. Jeżeli klasa dziedziczy po innej, dostępnej w zbiorze pakietów systemu Android, oznaczono to słowem „extends” . . . . .	13
3.2	Schemat optymalizacji rozmiaru bazy danych na przykładzie wyciągu z tablicy CSV. Przy konstruowaniu zbioru danych numer identyfikacyjny komórki należy rozbić na dwie kolumny – w kolumnie czwartej umieścić prefiks (wszystkie cyfry numeru poza cyfrą jedności), a w kolumnie piątej gwiazdkę (symbolizującą wszystkie cyfry) lub ciąg odpowiednich cyfr jedności (nierozdzielonych). . . . .	16
3.3	W celu pobrania danych komórkowych skorzystano z pakietu android.telephony. Niniejszy diagram przedstawia jedynie wykorzystane klasy, pola i metody. . . . .	17
3.4	Rysunek (a) zawiera przykład działania timera. W górze okienka przedstawiającego informacje o urządzeniu nadawczym wyświetlona jest informacja o tym jak długo telefon był podłączony. Na rysunku (b) można w praktyce zobaczyć jak działa rotacja. Wskaźniki dla dwóch i trzech stacji znajdujących w jednym punkcie zostały obrócone. . . . .	19
3.5	Po wysunięciu z lewej (a) menu przykrywa pozostałe elementy układu. Po wsunięciu menu (b) użytkownik zawsze ma dostęp do przycisków FAB oraz „hamburgera”. . . . .	21
3.6	Klasa słuchacza odpowiada nie tylko za stan usługi ScanService. Przycisk Scanning (w prawym dolnym rogu) może przyjmować stany enabled (a) i disabled (b). Dodatkowo przy przełączeniu pojawia się komunikat. . . . .	22
3.7	Przy pierwszym starcie aplikacji użytkownik zostaje zapytany o zgodę na dostęp do informacji o lokalizacji (a) oraz do plików (b). . . . .	23
3.8	Przy otwarciu FileManager znajduje w domyślnej lokalizacji pliki z rozszerzeniem BTST i wyświetla (a), aby użytkownik mógł wybrać, który utworzyć. Przy resecie konieczne jest potwierdzenie (b) zgody na usunięcie danych. Podobna prośba o potwierdzenie pojawia się przy otwarciu przed oknem dialogowym (a). . . . .	24
3.9	Przy zapisie (a) (do otwarcia w telefonie) i eksportie (b) (do otwarcia w komputerze) wyświetla się prośba o podanie nazwy pliku docelowego. . . . .	25
3.10	Domyślnie widok ListFragment (a) składa się z podstawowych elementów widoku nadzielnego oraz z rozszerzonego paska (grafika + nazwa aplikacji). Dopiero w miarę wykrywania kolejnych urządzeń, widok listy wypełnia się pojedynczymi elementami (b) z drugiego układu. Dodatkowo przy przepełnieniu widok staje się przewijalny. . . . .	26
3.11	Widok listy można intuicyjnie odświeżyć klasycznym ruchem przeciągnięcia palcem z góry ekranu. . . . .	27

3.12 Rysunek (a) przedstawia podstawowy widok mapy z naniesionym jednym elementem nadawczym i niebieską kropką oznaczającą pozycję telefonu komórkowego. Widać kompas, legende, przyciski regulacji przybliżenia oraz przyciski Spot i Refresh. Na rysunku (b) po kliknięciu markera pojawi się jego opis. Układ opisu jest analogiczny do tego z ListFragment. . . . .	29
3.13 Rysunek (a) przedstawia typowy widok TutorialFragment. Zrzut ekranu (b) został wykonyany w momencie przesunięcia palcem, czyli podczas przełączania elementów tutorialu. . . . .	31
4.1 Rezultaty testu według scenariusza: wielogodzinne działanie w tle. Widok listy (a) i mapy (b). . . . .	33
4.2 Rezultaty testu według scenariusza: poruszanie się po mieście. Widok listy (a) i mapy (b). . . . .	34
4.3 Rezultaty testu według scenariusza: przejazd autostradą. Widok listy (a) i mapy (b). . . . .	36
4.4 Rezultaty testu według scenariusza: przemieszczanie się wewnątrz budynku. Widok listy (a) i mapy (b). . . . .	37
4.5 Trasa testu według scenariusza: spacer po pagórkowatym terenie. W prawym dolnym rogu przedstawiono różnicę wzniesień. . . . .	38
4.6 Rezultaty testu według scenariusza: spacer po pagórkowatym terenie. Widok listy (a) i mapy (b). . . . .	38
4.7 Menu trybu testowania w telefonie z systemem Android (a) i pojedynczy BTS w danej lokalizacji zaznaczony na mapie rejestratora stacji bazowych (b). . . . .	40
4.8 Obrót o 30 stopni markerów na mapie w sytuacji, gdy pojawia się drugi (a) i trzeci (b) BTS. . . . .	40
4.9 Po zapisie (a) w aplikacji plik to_save.btst można odnaleźć w systemie plików telefonu w katalogu BTS_Tracker (b). . . . .	42
4.10 Po otwarciu (a) wczytanie informacji do listy i mapy (b) przebiegło pomyślnie. . . . .	42
4.11 Opcja Importuj w usłudze Google MyMaps. . . . .	44
4.12 Stacje przekaźnikowe z aplikacji „BTS Tracker” w usłudze Google MyMaps. . . . .	44
4.13 Przypadek szczególny: brak informacji o stacji w tablicy danych. . . . .	45
4.14 Przypadek szczególny: niepoprawne informacje w pliku wejściowym. . . . .	46
4.15 Wykorzystanie pamięci (a) oraz energii (b) w aplikacji. . . . .	46
5.1 Wyróżnione kolejne kroki w procesie działania aplikacji. . . . .	49

# **Spis tabelic**

3.1	Pozwolenia, jakie musi uzyskać aplikacja, aby działać poprawnie. . . . .	23
4.1	Zebrane informacje. Scenariusz: wielogodzinne działanie w tle. . . . .	34
4.2	Zebrane informacje. Scenariusz: poruszanie się po mieście. . . . .	35
4.3	Zebrane informacje. Scenariusz: przejazd autostradą. . . . .	36
4.4	Zebrane informacje. Scenariusz: przemieszczanie się wewnątrz budynku. . . . .	37
4.5	Zebrane informacje. Scenariusz: spacer po pagórkowatym terenie. . . . .	39
4.6	Zebrane informacje. Przypadek testowy: więcej niż jeden BTS w danej lokalizacji. . . . .	41