

POLITECHNIKA WROCŁAWSKA  
WYDZIAŁ ELEKTRONIKI

---

KIERUNEK: INFORMATYKA (INF)

SPECJALNOSC: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH (INS)

PRACA DYPLOMOWA  
INŻYNIERSKA

Kieszonkowy trener personalny wspomagający  
trening siłowy lub wytrzymałościowy.

Pocket personal trainer for strength or  
endurance sports.

AUTOR:

Wojciech Kolendo

PROWADZĄCY PRACĘ:

Dr inż. Tomasz Babczyński

OCENA PRACY:

---

WROCŁAW, 2016

# Spis treści

Spis rysunków .....	4
Spis tabel .....	5
Spis listingów .....	6
Skróty .....	8
1. Wstęp.....	9
1.1. Wprowadzenie .....	9
1.2. Cel i zakres pracy .....	9
2. Przegląd istniejących rozwiązań .....	11
3. Analiza wymagań .....	13
3.1. Opis działania i schemat logiczny systemu .....	13
3.2. Wymagania funkcjonalne .....	13
3.2.1. Diagram przypadków użycia.....	14
3.2.2. Scenariusze przypadków użycia.....	15
3.3. Wymagania niefunkcjonalne .....	20
3.4. Używane technologie .....	20
3.4.1. Android.....	20
3.4.2. Pozostałe technologie .....	23
4. Projekt systemu .....	24
4.1. Architektura aplikacji .....	24
4.2. Mechanizm łączenia z bazą danych.....	24
4.3. Mechanizm lokalizowania użytkownika .....	25
4.4. Projekt widoków interfejsu.....	25
4.5. Projekt bazy danych.....	27
4.5.1. Model koncepcyjny .....	27

4.5.2.	Model fizyczny .....	28
4.5.3.	Ograniczenia integralnościowe .....	31
5.	Implementacja systemu .....	32
5.1.	Implementacja bazy danych .....	32
5.1.1.	Modele encji .....	32
5.1.2.	Mapowanie obiektowo-relacyjne .....	38
5.2.	Implementacja modułów aplikacji.....	41
5.2.1.	Definicje komponentów aplikacji .....	41
5.2.2.	Definicje komponentów interfejsu .....	43
5.2.3.	Moduł głównego menu.....	44
5.2.4.	Moduł planowania treningów.....	48
5.2.5.	Moduł śledzenia użytkownika.....	55
5.2.6.	Moduł pomocniczych kalkulatorów .....	70
5.3.	Instalacja i konfigurowanie systemu .....	76
5.4.	Wykorzystane narzędzia i biblioteki .....	77
5.5.	Testy aplikacji.....	77
6.	Podsumowanie .....	78
	Bibliografia.....	80
	Dodatek A: .....	81

# Spis rysunków

Rys. 1 - Logo aplikacji Endomondo.....	11
Rys. 2 - Logo aplikacji Total Fitness .....	11
Rys. 3 - Diagram przypadków użycia .....	14
Rys. 4 - Logo systemu Android .....	20
Rys. 5 - Wykres liniowy systemów mobilnych i ich udziałów .....	21
Rys. 6 - Wykres kołowy wersji Androida oraz udziałów w dystrybucji.....	23
Rys. 7 - Podstawowe komponenty Material Design .....	26
Rys. 8 - Model konceptualny bazy danych – notacja Chena .....	27
Rys. 9 - Model fizyczny treningu i jego ćwiczeń – notacja kurzej łapki .....	28
Rys. 10 - Model fizyczny aktywności fizycznej – notacja kurzej łapki.....	29
Rys. 11 - Logo aplikacji Pocket Trainer.....	32
Rys. 12 - Cykl życia aktywności.....	41
Rys. 13 - Cykl życia fragmentu.....	42
Rys. 14 - Zrzut ekranu głównego menu .....	47
Rys. 15 - Zrzut ekranu dodawania nowego treningu .....	51
Rys. 16 - Zrzut ekranu listy ćwiczeń danego treningu .....	54
Rys. 17 - Zrzut ekranu rozpoczynania nowej aktywności.....	56
Rys. 18 - Zrzut ekranu widoku mapy .....	63
Rys. 19 - Zrzut ekranu szczegółów treningu.....	66
Rys. 20 - Zrzut ekranu historii aktywności .....	69
Rys. 21 - Zrzut ekranu kalkulatora BMI .....	72
Rys. 22 - Zrzut ekranu rezultatu BMI .....	73
Rys. 23 - Zrzut ekranu kalkulatora BFP.....	74
Rys. 24 - Zrzut ekranu rezultatu BFP.....	76

# Spis tabel

Tab. 1 - Udział rynkowy systemów mobilnych .....	21
Tab. 2 - Wersje Androida oraz udziały w dystrybucji .....	22
Tab. 3 - Opis atrybutów encji Training .....	28
Tab. 4 - Opis atrybutów encji Training Activity .....	28
Tab. 5 - Opis atrybutów encji Workout.....	30
Tab. 6 - Opis atrybutów encji Position.....	30
Tab. 7 - Opis atrybutów encji Speed .....	30
Tab. 8 - Opis atrybutów encji Distance .....	31

# Spis listingów

Listing 1 - Model encji Training – definicja klasy .....	32
Listing 2 - Model encji Training - definicje pól .....	33
Listing 3 - Model encji Training - definicja kolekcji .....	33
Listing 4 - Model encji Training - metody .....	33
Listing 5 - Model encji Training Activity .....	34
Listing 6 - Model encji JoinTrainingsWithTrainingActivities .....	34
Listing 7 - Model encji Workout - pola klasy standardowego typu .....	35
Listing 8 - Model encji Workout - pola klasy niestandardowego typu .....	35
Listing 9 - Model encji Workout - kolekcje klasy .....	36
Listing 10 - Model encji Position .....	36
Listing 11 - Model encji JoinWorkoutWithPositions .....	37
Listing 12 - Model encji Speed .....	37
Listing 13 - Model encji JoinWorkoutWithSpeeds .....	37
Listing 14 - Model encji Distance .....	38
Listing 15 - Model encji JoinWorkoutWithDistances .....	38
Listing 16 - Transformer typów enum - WorkoutType .....	39
Listing 17 - Transformer typów enum - Units .....	40
Listing 18 - Transformer typu niestandardowego - Calendar .....	40
Listing 19 - Moduł głównego menu - zmiana fragmentów .....	44
Listing 20 - Moduł głównego menu - listener elementu menu .....	45
Listing 21 - Moduł głównego menu - sprawdzenie wersji systemu .....	46
Listing 22 - Moduł głównego menu - obsługa kliknięcia strzałki powrotu .....	46
Listing 23 - Moduł planowania treningów - pobieranie listy z bazy danych .....	48
Listing 24 - Moduł planowania treningów - ustawianie elementu listy .....	48
Listing 25 - Moduł planowania treningów - dodawanie treningu do bazy danych .....	49
Listing 26 - Moduł planowania treningów - usuwanie treningu z bazy danych .....	49
Listing 27 - Moduł planowania treningów - okno dialogowe dodawania .....	50
Listing 28 - Moduł planowania treningów - okno dialogowe usuwania .....	50
Listing 29 - Moduł planowania treningów - przejście do nowej aktywności .....	52
Listing 30 - Moduł planowania treningów - pobieranie ćwiczeń z bazy danych .....	52
Listing 31 - Moduł planowania treningów - dodawanie nowego ćwiczenia .....	52

Listing 32 - Moduł planowania treningów - usuwanie ćwiczenia .....	53
Listing 33 - Moduł śledzenia użytkownika - listener zmiany jednostek.....	55
Listing 34 - Moduł śledzenia użytkownika - sprawdzenie uprawnień GPS .....	55
Listing 35 - Moduł śledzenia użytkownika - tworzenie aktywności.....	57
Listing 36 - Moduł śledzenia użytkownika - tworzenie serwisu.....	57
Listing 37 - Moduł śledzenia użytkownika - start API lokalizowania .....	58
Listing 38 - Moduł śledzenia użytkownika - listener nowych lokalizacji.....	58
Listing 39 - Moduł śledzenia użytkownika - obliczanie danych w m/s .....	59
Listing 40 - Moduł śledzenia użytkownika - obliczanie danych w km/h.....	59
Listing 41 - Moduł śledzenia użytkownika - kończenie aktywności fizycznej.....	60
Listing 42 - Moduł śledzenia użytkownika - zapisywanie do lokalnej bazy danych.....	61
Listing 43 - Moduł śledzenia użytkownika - pobieranie adresu lokalizacji.....	62
Listing 44 - Moduł śledzenia użytkownika - pobieranie aktywności z bazy danych.....	64
Listing 45 - Moduł śledzenia użytkownika - wyświetlenie szczegółów aktywności.....	64
Listing 46 - Moduł śledzenia użytkownika - usuwanie aktywności fizycznej.....	67
Listing 47 - Moduł pomocniczych kalkulatorów - adapter View Pager .....	70
Listing 48 - Moduł pomocniczych kalkulatorów - obliczanie BMI.....	71
Listing 49 - Moduł pomocniczych kalkulatorów - przełączanie zagnieżdżonych fragmentów.....	71

# Skróty

**GPS** (ang. *Global Positioning System*)

**GUI** (ang. *Graphical User Interface*)

**BMI** (ang. *Body Mass Index*)

**BFP** (ang. *Body Fat Percentage*)

**SDK** (ang. *Software Development Kit*)

**API** (ang. *Application Programming Interface*)

**URI** (ang. *Uniform Resource Identifier*)

**LTE** (ang. *Long Term Evolution*)

**Wi-Fi** (ang. *Wireless Fidelity*)

**XML** (ang. *Extensible Markup Language*)

**IDE** (ang. *Integrated Development Environment*)

**APK** (ang. *Android Application Package*)

**JDK** (ang. *Java Development Kit*)

**ADB** (ang. *Android Debug Bridge*)

**USB** (ang. *Universal Serial Bus*)

**ORM** (ang. *Object-Relational Mapping*)

**DAO** (ang. *Data Access Object*)



# 1. Wstęp

## 1.1. Wprowadzenie

Rozwój, postęp i udział urządzeń mobilnych na rynku dały ludziom możliwość powszechnego korzystania ze smartfonów w roli asystenta w codziennych sprawach. Dzisiaj jesteśmy w stanie trzymać w jednym miejscu w kieszeni sprzęt umożliwiający cały wachlarz multimedialny: aparat, kamerę, nawigację GPS, telefonię GSM/LTE, bezprzewodowy dostęp do Internetu pozwalający na komunikatory tekstowe i wideo rozmowy, a także swobodne korzystanie ze stron internetowych, które jest w stanie umożliwić niektórym ludziom całkowite zrezygnowanie z komputera osobistego. Aplikacje codziennego użytku stały się popularne z uwagi na możliwości jakie oferują, prostotę użytkowania oraz przede wszystkim mobilność idąca w parze z niewielkimi rozmiarami sprzętu mieszczącego się w każdej kieszeni.

Dynamiczny wzrost udziału aplikacji mobilnych w branży oprogramowania dał ludziom amatorsko uprawiającym sporty możliwość korzystania z aplikacji wspierających aktywności fizyczne. Najpopularniejsze aplikacje skupiają się mocno na konkretnych dyscyplinach, na przykład wyłącznie sportach wytrzymałościowych, omijając pozostałe, co sprowadza się do dwóch wniosków:

- Istnieje nisza wśród aplikacji mobilnych skupiających sporty siłowe i wytrzymałościowe w jednym miejscu.
- Aplikacje skupiające się na jednej dyscyplinie oferują wiele profesjonalnych rozwiązań nieużywanych przez przeciętnego użytkownika.

## 1.2. Cel i zakres pracy

W niniejszej pracy zostanie podjęta próba stworzenia aplikacji mobilnej wspierającej uprawianie sportów zarówno siłowych i wytrzymałościowych. Z uwagi na największą popularność w gałęzi systemów mobilnych oraz własne doświadczenie zawodowe aplikacja będzie od początku analizowana i implementowana na system Android. Głównym celem pracy będzie stworzenie oprogramowania wspierającego aktywność fizyczną taką jak bieganie czy jazda na rowerze, a także wspierającą treningi na siłowni.

Zakres pracy będzie obejmował zaprojektowanie i implementację aplikacji posiadającej trzy wyróżniające się moduły:

- Śledzenie aktywności fizycznej użytkownika – biegania lub jazdy na rowerze poprzez moduł GPS urządzenia oraz zbieranie statystyk.
- Możliwość budowania własnych planów treningowych.
- Dodatki, np. kalkulator BMI, BFP.

W projektowaniu istotną rolę będzie spełniał wygląd interfejsu użytkownika, który powinien być jak najbardziej zrozumiały dla przeciętnego użytkownika uruchamiającego aplikację po raz pierwszy.

## 2. Przegląd istniejących rozwiązań

Biorąc pod lupę najpopularniejsze aplikacje dostępne w sklepie Google Play, których funkcjonalności częściowo pokrywają się z założeniami niniejszej pracy można wyróżnić dwie aplikacje:

- Endomondo



Rys. 1 - Logo aplikacji Endomondo

Aplikacja umożliwia śledzenie użytkownika poprzez sygnał GPS, gromadzenie statystyk, a także dzielenie własnej aktywności z innymi zarejestrowanymi użytkownikami.

Jedną z inspiracji jest sposób zaprojektowania interfejsu użytkownika, który przejrzysto oddziela główne funkcjonalności od mniej istotnych. Drugim elementem wartym uwagi jest sposób wyświetlania kolejnych widoków odpowiedzialnych za rozpoczęcie treningu, wyświetlanie głównej mapy i prezentacja wyników oraz statystyk.

- Total Fitness



Rys. 2 - Logo aplikacji Total Fitness

Aplikacja oferuje możliwość budowania własnych planów treningowych, przewodnik po ćwiczeniach oraz kalkulatory umożliwiające obliczenie własnego zapotrzebowania kalorycznego lub wskaźnika BMI.

Pomimo bogatej listy funkcjonalności, prezentacja interfejsu użytkownika w aplikacji nie została zaprojektowana w oparciu o Material Design (opisywany w 4.4), czyli zbiór wskazówek i wytycznych opracowanych przez firmę Google. Posiadając własne grafiki i solidnie zaprojektowany projekt designu można się starać implementować własne GUI. Niestety w tym przypadku interfejs jest chaotyczny oraz trudny do zrozumienia, co jest głównym czynnikiem wpływającym na negatywny odbiór przez użytkownika. Jest to jeden z powodów, dla których w niniejszej pracy zostanie użyty wspomniany Material Design.

# 3. Analiza wymagań

## 3.1. Opis działania i schemat logiczny systemu

Aplikacja Pocket Trainer umożliwia użytkownikowi gromadzenie danych własnych aktywności fizycznych. Sporty wytrzymałościowe takie jak bieganie lub jazda na rowerze są analizowane przez dane dostarczane przez sygnał GPS. Są one odbierane w postaci współrzędnych lokalizacji użytkownika, więc pozwalają na obliczanie statystyk na bieżąco podczas treningu, np. aktualna prędkość, spalane kalorie lub dystans. Cała aktywność jest trzymana jako jeden model, dzięki czemu użytkownik ma wgląd na pełne statystyki po zakończeniu przebiegu trasy, np. wykresy drogi od czasu. Możliwy jest także dostęp do historii aktywności trzymany w lokalnej bazie danych aplikacji. Użytkownik ma także możliwość budowania własnych planów treningowych wraz z ich ćwiczeniami, które oba są trzymane w lokalnej bazie danych. Dostępny jest także kalkulator BMI pozwalający na obliczenie własnego indeksu masy ciała, który informuje użytkownika o jego stanie fizycznym oraz kalkulator BFP obliczający udział tkanki tłuszczowej w ciele człowieka.

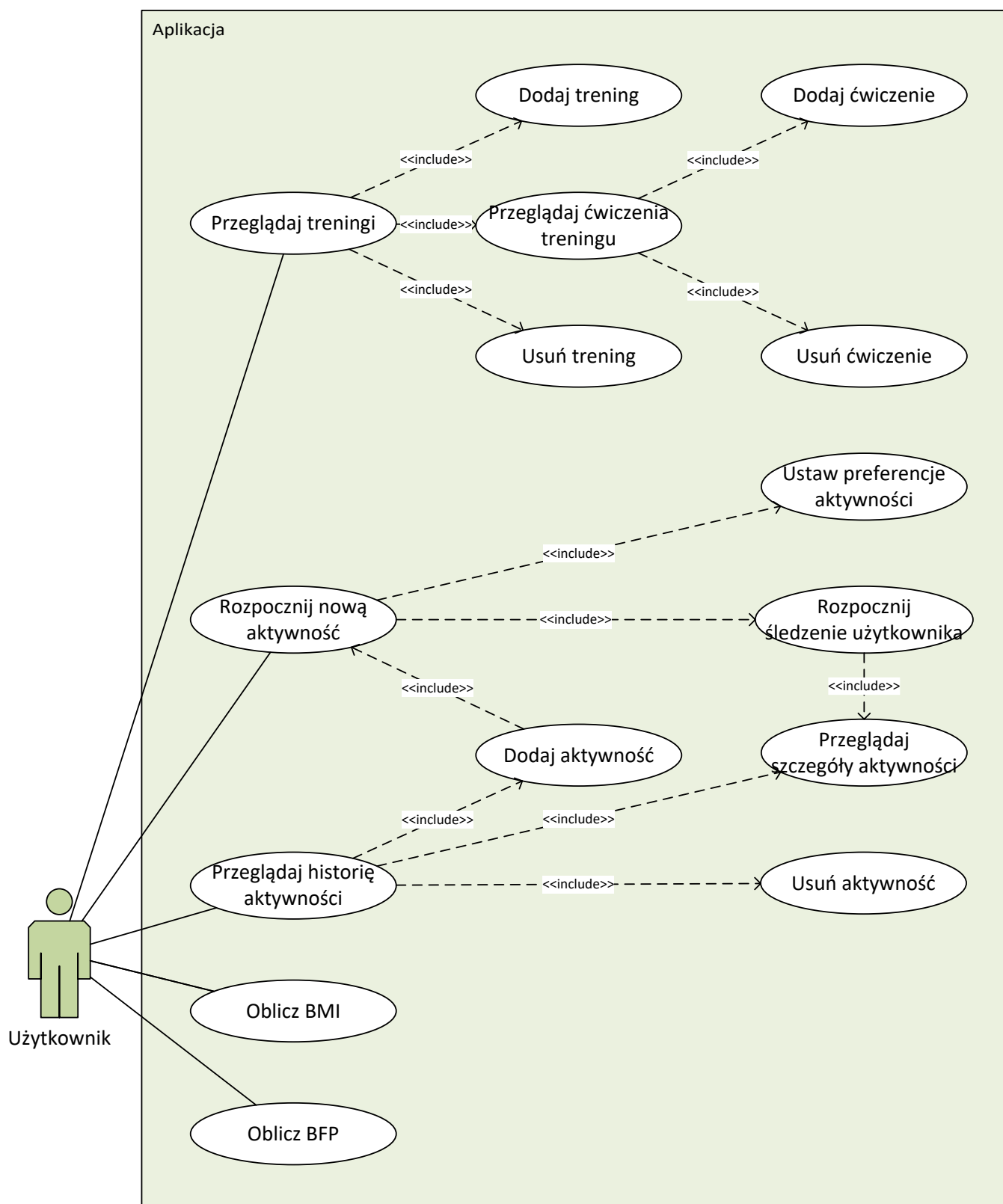
W niniejszej pracy będą powszechnie używane dwa podobne pojęcia:

- Trening – związany z treningiem siłowym i planami treningowymi.
- Aktywność fizyczna – związana z bieganiem, jazdą na rowerze oraz śledzeniem lokalizacji.

## 3.2. Wymagania funkcjonalne

- Dodawanie, usuwanie, edytowanie własnego planu treningowego.
- Dodawanie, usuwanie, edytowanie ćwiczeń do istniejącego planu treningowego.
- Śledzenie aktywności fizycznej używając modułu GPS oraz gromadzenie statystyk.
- Przegląd historii skończonych aktywności oraz ich statystyk.
- Kalkulator BMI.
- Kalkulator BFP

### 3.2.1. Diagram przypadków użycia



Rys. 3 - Diagram przypadków użycia

### 3.2.2. Scenariusze przypadków użycia

**Przypadek użycia:** Przeglądaj treningi.

**Cel:** Wyświetlenie listy treningów dostępnych w lokalnej bazie danych.

**Przebieg PU:**

1. Użytkownik wybiera pozycję „Training Planner” z podręcznego menu.
2. Wyświetlenie widoku listy.
3. Pobranie kolekcji treningów z lokalnej bazy danych.
4. Wyświetlenie treningów na liście.

**Przypadek użycia:** Dodaj trening.

**Cel:** Dodanie nowego treningu zdefiniowanego przez użytkownika do lokalnej bazy danych.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy treningów.

**Przebieg PU:**

1. Użytkownik klika na ikonę dodawania i uzupełnia dane treningu.
2. Użytkownik klika na ikonę „Add”.
3. Zapisanie treningu do bazy danych.
4. Odświeżenie wyświetlanej listy treningów.

**Przypadek użycia:** Usuń trening.

**Cel:** Usunięcie treningu wskazanego przez użytkownika z lokalnej bazy danych.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy treningów oraz baza nie jest pusta.

**Przebieg PU:**

1. Użytkownik klika na ikonę usuwania przy wybranym, konkretnym treningu.
2. Usunięcie treningu z bazy danych.
3. Odświeżenie wyświetlanej listy treningów.

**Przypadek użycia:** Przeglądaj ćwiczenia treningu.

**Cel:** Wyświetlenie listy ćwiczeń dostępnych w lokalnej bazie danych dla wybranego treningu.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy treningów.

**Przebieg PU:**

1. Użytkownik klika na wybrany trening na liście treningów.
2. Wyświetlenie widoku listy.
3. Pobranie kolekcji ćwiczeń odpowiadających treningowi z lokalnej bazy danych.
4. Wyświetlenie ćwiczeń na liście.

**Przypadek użycia:** Dodaj ćwiczenie.

**Cel:** Dodanie nowego ćwiczenia dla treningu zdefiniowanego przez użytkownika do lokalnej bazy danych.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy ćwiczeń.

**Przebieg PU:**

1. Użytkownik klika na ikonę dodawania i uzupełnia dane ćwiczenia.
2. Użytkownik klika na ikonę „Add”.
3. Zapisanie ćwiczenia do bazy danych dla konkretnego treningu.
4. Odświeżenie wyświetlanej listy ćwiczeń.

**Przypadek użycia:** Usuń ćwiczenie.

**Cel:** Usunięcie ćwiczenia treningu wskazanego przez użytkownika z lokalnej bazy danych.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy ćwiczeń oraz baza ćwiczeń dla wybranego treningu nie jest pusta.

**Przebieg PU:**

1. Użytkownik klika na ikonę usuwania przy wybranym, konkretnym ćwiczeniu.
2. Usunięcie ćwiczenia z bazy danych dla konkretnego treningu.
3. Odświeżenie wyświetlanej listy treningów.



**Przypadek użycia:** Rozpocznij nową aktywność.

**Cel:** Wyświetlenie widoku rozpoczynającego nową aktywność.

**Przebieg PU:**

1. Użytkownik wybiera pozycję „New workout” z podręcznego menu.
2. Wyświetlenie widoku nowej aktywności.

**Przypadek użycia:** Ustaw preferencje aktywności.

**Cel:** Ustawienie preferencji dla nowej aktywności.

**Warunki wstępne:** Aktualnie wyświetlony jest widok nowej aktywności.

**Przebieg PU:**

1. Użytkownik opcjonalnie wybiera:
  - Dla nowej aktywności:
    - Typ aktywności.
  - Preferencje zapisywane w cache:
    - Tryb mniejszej dokładności lokalizowania GPS.
    - Jednostki metryczne/imperialne.

**Przypadek użycia:** Rozpocznij śledzenie użytkownika.

**Cel:** Wyświetlenie widoku oraz serwisu śledzenia użytkownika.

**Warunki wstępne:** Aktualnie wyświetlony jest widok nowej aktywności.

**Przebieg PU:**

1. Użytkownik klika na przycisk „Start”.
2. Wyświetlenie widoku śledzenia oraz uruchomienie serwisu.
3. Obliczanie i zapisywanie statystyk aktywności w tle.

**Przypadek użycia:** Przeglądaj szczegóły aktywności.

**Cel:** Wyświetlenie widoku statystyk zakończonej aktywności.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy z historią aktywności lub śledzenia użytkownika.

**Przebieg PU:**

1. Użytkownik kończy śledzenie lub klika na wybraną aktywności z listy.
2. Wyświetlenie widoku statystyk aktywności.

**Przypadek użycia:** Przeglądaj historię aktywności.

**Cel:** Wyświetlenie listy zakończonych aktywności dostępnych w lokalnej bazie danych.

**Przebieg PU:**

1. Użytkownik wybiera pozycję „History” z podręcznego menu.
2. Wyświetlenie widoku listy.
3. Pobranie kolekcji zakończonych aktywności z lokalnej bazy danych.
4. Wyświetlenie aktywności na liście.

**Przypadek użycia:** Dodaj aktywność.

**Cel:** Dodanie nowej aktywności do lokalnej bazy danych.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy aktywności.

**Przebieg PU:**

1. Użytkownik klika na ikonę dodawania.
2. Uruchomienie widoku nowej aktywności.
3. Uruchomienie serwisu śledzenia użytkownika.
4. Zakończenie serwisu śledzenia użytkownika.
5. Zapisanie nowej aktywności do lokalnej bazy danych.

**Przypadek użycia:** Usunąć aktywność.

**Cel:** Usunięcie aktywności wskazanej przez użytkownika z lokalnej bazy danych.

**Warunki wstępne:** Aktualnie wyświetlony jest widok listy aktywności oraz baza aktywności nie jest pusta.

**Przebieg PU:**

1. Użytkownik klika na ikonę usuwania przy wybranej, konkretnej aktywności.
2. Usunięcie aktywności z bazy danych.
3. Odświeżenie wyświetlanej listy aktywności.

**Przypadek użycia:** Oblicz BMI.

**Cel:** Wyświetlenie BMI użytkownika.

**Przebieg PU:**

1. Użytkownik wybiera pozycję „Calculate BMI” z podręcznego menu.
2. Wyświetlenie widoku kalkulatora.
3. Użytkownik wypełnia wymagane pola.
4. Użytkownik klika przycisk „Calculate”.
5. Obliczenie BMI na podstawie wprowadzonych danych.
6. Wyświetlenie widoku z rezultatem.

**Przypadek użycia:** Oblicz BFP.

**Cel:** Wyświetlenie BFP użytkownika.

**Przebieg PU:**

1. Użytkownik wybiera pozycję „Calculate BFP” z podręcznego menu.
2. Wyświetlenie widoku kalkulatora.
3. Użytkownik wypełnia wymagane pola.
4. Użytkownik klika przycisk „Calculate”.
5. Obliczenie BFP na podstawie wprowadzonych danych.
6. Wyświetlenie widoku z rezultatem.

### 3.3. Wymagania niefunkcjonalne

Aplikacja jest projektowana i tworzona z uwzględnieniem wysokiej użyteczności i niezawodności. Wszystkie działania i operacje przeprowadzane w aplikacji są analizowane pod kątem jak najlepszej wydajności. Przykładowo pobieranie kolekcji z relacji „do wielu” z bazy danych jest przeprowadzane tylko raz, a kolejne zapytania operują na danych trzymanych w pamięci cache. Celem tego zabiegu jest szybkie działanie aplikacji i wysyłanie zapytań do bazy danych, które wymagają więcej czasu, tylko wtedy kiedy jest to niezbędne.

Największy nacisk w kierunku niezawodności jest skupiony na module śledzenia lokalizacji, który nie może być wolny w działaniu oraz musi być wolny od błędów, ponieważ każdy z nich może być krytyczny dla pracy całej aplikacji.

### 3.4. Używane technologie

#### 3.4.1. Android



Rys. 4 - Logo systemu Android

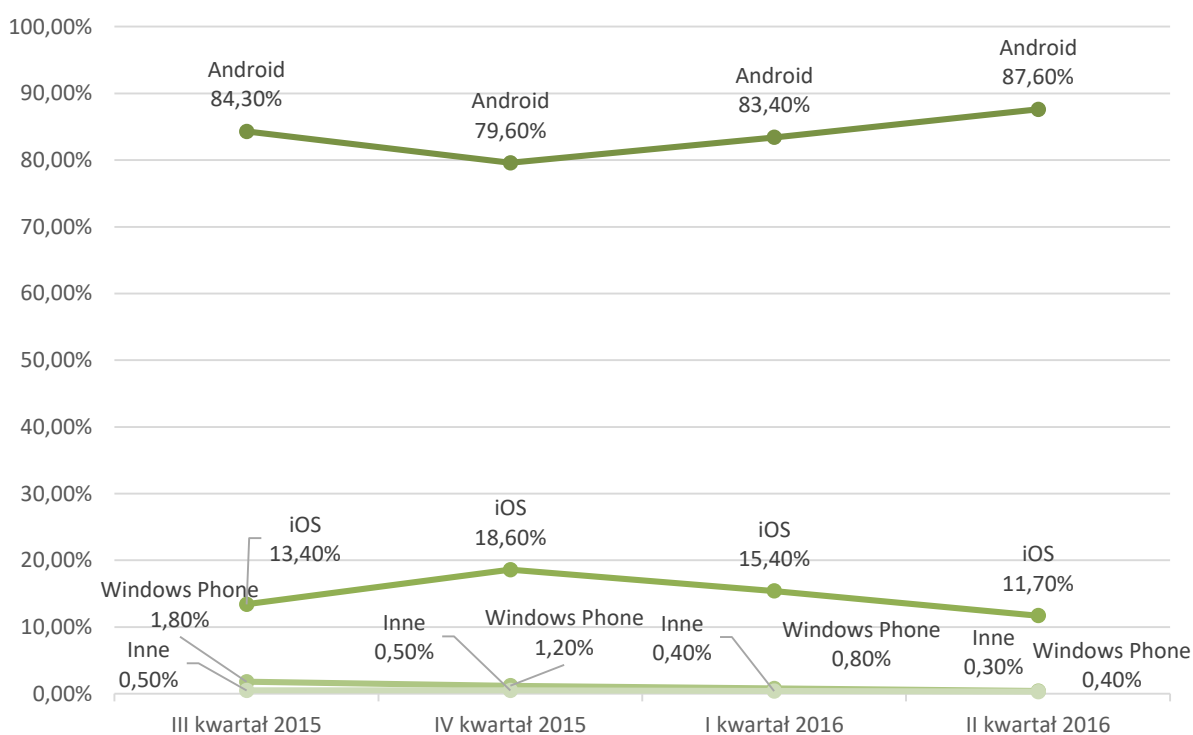
System operacyjny Android został wybrany w niniejszej pracy z kilku powodów:

- Największej bazy użytkowników wśród systemów operacyjnych każdego rodzaju.
- Największego udziału rynkowego wśród mobilnych systemów operacyjnych na świecie.<sup>[1]</sup>
- Największy udział wśród mobilnych systemów operacyjnych w Polsce.<sup>[2]</sup>
- Własnego doświadczenia zawodowego.

Tab. 1 - Udział rynkowy systemów mobilnych<sup>[1]</sup>

Okres	Android	iOS	Windows Phone	Inne
III kwartał 2015	84.3%	13.4%	1.8%	0.5%
IV kwartał 2015	79.6%	18.6%	1.2%	0.5%
I kwartał 2016	83.4%	15.4%	0.8%	0.4%
II kwartał 2016	87.6%	11.7%	0.4%	0.3%

Dominacja Androida na rynku jest niezmienna od kilku lat i nic nie wskazuje, aby trend się odwrócił. Pomimo wysokiej pozycji na ogólnoswiatowym rynku, w niektórych krajach (Wielka Brytania, Stany Zjednoczone) w sprzedaży króluje iOS. Największym powodem wahań w udziałach procentowych są premiery nowych, flagowych telefonów z konkretnymi systemami. Przykładem jest premiera iPhone'a 6s we wrześniu/październiku 2015, którą można zaobserwować na wykresie.



Rys. 5 - Wykres liniowy systemów mobilnych i ich udziałów

Za system Android odpowiedzialna jest aktualnie grupa Open Handset Alliance będąca sojuszem biznesowym 78 firm, skupiająca największe koncerny branży IT, zawiązana z inicjatywy Google i w głównej mierze prowadzona przez tę firmę.

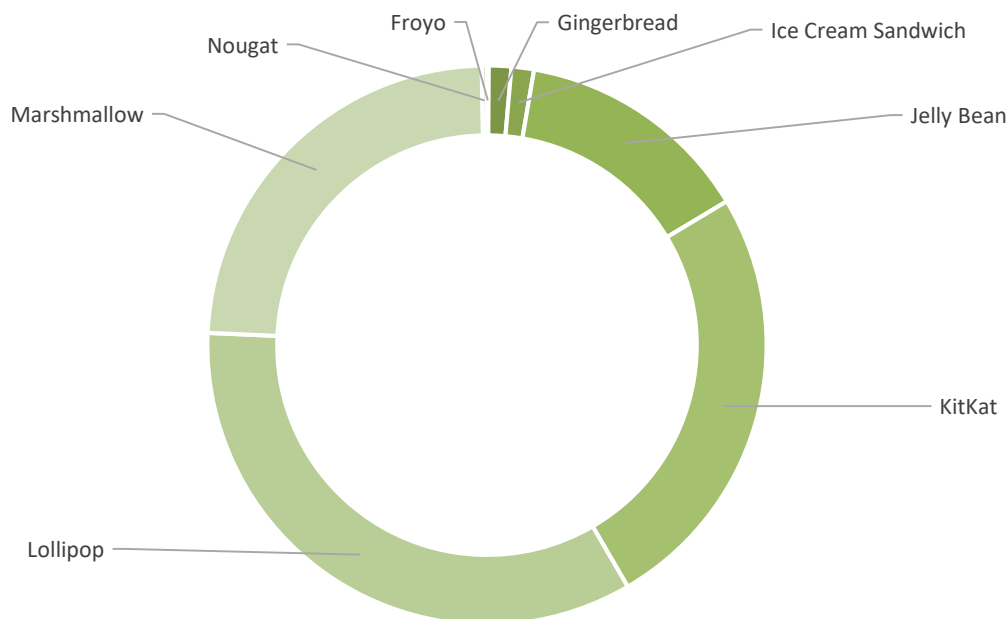
Każda nowa wersja systemu posiada odpowiadające jej API oraz SDK udostępniane deweloperom. Największym krokiem milowym dla systemu był Android 5.0 Lollipop

wprowadzając do środowiska Material Design, styl graficzny stworzony i faworyzowany przez Google. W nowszych wersjach nowości designu oraz nowe komponenty graficzne wprowadzane są w osobnej bibliotece Support Library, dla której priorytetem jest wsteczna kompatybilność ze starszymi wersjami systemu. Deweloper podnosząc wersję implementowanego API w aplikacji musi także wziąć pod uwagę wymagania jakie ze sobą niesie dana wersja. Przykładem jest nowa obsługa pozwoleń aplikacji w Androidzie 6.0 Marshmallow lub zabezpieczenia przy odnośnikach URI prowadzących poza aplikację w Androidzie 7.0 Nougat.

Programista w manifeście tworzonej aplikacji musi zadeklarować minimalne SDK, na którym ma działać aplikacja, balansując między ilością urządzeń z zainstalowaną wersją Androida, a nowymi funkcjonalnościami z brakiem wstecznej kompatybilności, których brak w starych API. Obecnie rekomendowane jest pisanie aplikacji dla Androida 4.4 KitKat (API 19) oraz nowszych wersji, co przekłada się na około 83.6% urządzeń działających na systemie Android<sup>[3]</sup>. W niniejszej pracy skorzystano z minimalnego SDK 19, ponieważ zejście niżej wiązałoby się z utratą niektórych funkcjonalności użytych w aplikacji, np. transparentnego paska systemowego lub obsługi wektorowych ikon.

Tab. 2 - Wersje Androida oraz udziały w dystrybucji<sup>[3]</sup> – dane zebrane w 7 dniowym okresie kończącym się 7 listopada 2016. Wersje z udziałem mniejszym niż 0.1% nie zostały pokazane.

Wersja	Nazwa kodowa	API	Dystrybucja
2.2	Froyo	8	0.1%
2.3.3 – 2.3.7	Gingerbread	10	1.3%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	1.3%
4.1.x	Jelly Bean	16	4.9%
4.2.x		17	6.8%
4.3		18	2.0%
4.4	KitKat	19	25.2%
5.0	Lollipop	21	11.3%
5.1		22	22.8%
6.0	Marshmallow	23	24.0%
7.0	Nougat	24	0.3%



Rys. 6 - Wykres kołowy wersji Androida oraz udziałów w dystrybucji

### 3.4.2. Pozostałe technologie

Android pomimo obsługiwanego natywnego C i C++ przy tworzeniu niskopoziomych frameworków lub silników, nie zapewnia pełnego wsparcia jak dla głównego języka – Javy. Aktualnie najnowsza wersja API zaczęła wspierać Javę 8, lecz z powodu wstecznej kompatybilności dla starszych telefonów i wersji systemu, aplikacja w niniejszej pracy jest pisana w Javie 7 – wciąż najszerzej używanej wśród programistów Androida.

SQLite jest bazą danych SQL z otwartym źródłem, która przetrzymuje dane w pliku tekstowym na urządzeniu. Android zawiera w sobie zaimplementowaną obsługę baz SQLite, które posiadają wszystkie funkcjonalności relacyjnych baz danych. W pisanej aplikacji został użyty framework wspierający obsługę bazy danych, wspomniany w 4.2 oraz 5.4.

# 4. Projekt systemu

## 4.1. Architektura aplikacji

Android wspiera domyślnie wzorzec projektowy oparty na widoku oraz modelu. Każdy widok, który widzimy na ekranie telefonu jest zdefiniowany w osobnym pliku w formacie XML zawierającym wszystkie widoczne komponenty. Plik z layoutem można przypisywać do wielu modeli, nie zawiera on żadnej logiki i możliwe jest zagnieżdżanie jednego widoku XML w drugim.

Rolę modelu w Androidzie pełni aktywność (ang. *activity*), w której zaimplementowana jest cała logika oraz do której przypisujemy plik z widokiem. Dodatkowo w niniejszej pracy, zgodnie z praktykami wydajnego programowania na Androidzie, aktywność posiada zagnieżdżone w niej kontenery z fragmentami. Umożliwiają one większą elastyczność w zarządzaniu widokami, ponieważ aktywność z łatwością może wymieniać fragmenty na inne. Zapewnia to wtedy lepszą wydajność i szybsze działanie aplikacji. Fragment również posiada przypisany plik XML z widokiem.

Proces kompilacji, obsługa bibliotek i zależności oraz budowanie projektu jest zarządzane i obsługiwane przez system Gradle.

## 4.2. Mechanizm łączenia z bazą danych

Baza danych SQLite jest zapisywana lokalnie na urządzeniu w prywatnej przestrzeni dyskowej, która jest zintegrowana z aplikacją i niedostępna dla innych. Android zapewnia szereg różnych API zawartych w klasie SQLiteOpenHelper, obsługujących zarządzanie bazą danych. Podczas używania tej klasy w celu uzyskania referencji do bazy, operacje wymagające dłuższego czasu działania – tworzenie i modyfikowanie bazy danych są wykonywane w momencie ich wywołania, a nie przy uruchomieniu aplikacji.

W niniejszej pracy został użyty framework greenDAO zapewniający mapowanie obiektowo-relacyjne. Pełni rolę pomostu między bazą danych SQLite a obiektami w Javie. Udostępnia szereg metod pozwalających na edycje tabel oraz adnotacje pozwalające np. w łatwy sposób zastosować asocjacyjną encję.



### 4.3. Mechanizm lokalizowania użytkownika

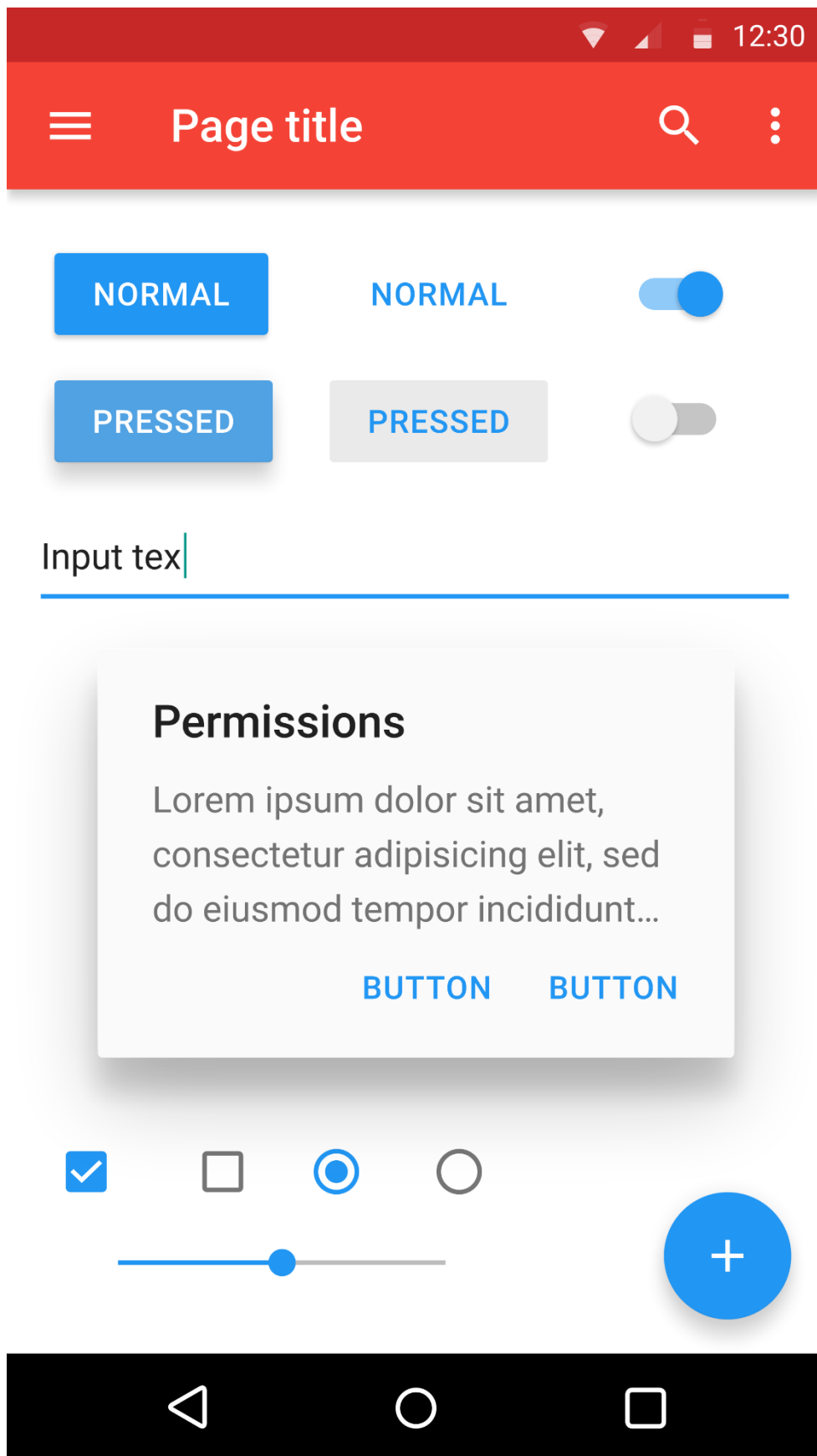
Proces lokalizowania użytkownika jest możliwy dzięki usłudze Google Play Services. Jest ona dostarczana w bibliotece jako klasa `GoogleApiClient` zawierając w sobie szereg różnych API, w tym także `FusedLocationProviderApi`, które jest używane w aplikacji. `FusedLocationProviderApi` zapewnia wysokiej jakości odpowiedzi o lokacji urządzenia na podstawie sygnału GPS, LTE lub Wi-Fi, które z kolei są wychwytywane przez listener i obsługiwane przez zaimplementowaną logikę w aplikacji. Cały opisany proces działa w serwisie (ang. *service*), który jest niepodatny na minimalizację aplikacji lub na wyłączenie ekranu urządzenia.

### 4.4. Projekt widoków interfejsu

Wszystkie widoki w aplikacji projektowane są z użyciem Material Designu. Jest to zbiór zasad i reguł stworzony przez Google, który łączy klasyczne zasady dobrego designu z innowacją i nową technologią. Celem firmy było stworzenie graficznego systemu, który pozwala ujednolicić wszystkie ich produkty na każdej platformie. Aktualnie Material Design jest silnie rekomendowany przez Google jako docelowy design aplikacji na Androida oraz polecany do aplikacji webowych.

Wyróżniającym czynnikiem designu jest fakt, że materiałowe środowisko istnieje w przestrzeni trójwymiarowej, co oznacza że każdy komponent ma także swoją grubość. Wynikiem tego są np. efekty zaciemnienia komponentów leżących niżej od pozostałych lub efekt cienia przy wciskaniu materiałowego przycisku. Wszystkie podstawowe komponenty są dostępne domyślnie, a dodatkowe - dostępne w bibliotece Support Library. Material Design został wprowadzony w Androidzie 5.0 Lollipop, co oznacza, że kompatybilność ze starszymi wersjami nie jest pełna. Na wersji 4.4 KitKat nie są widoczne np. efekty wciskanego przycisku.

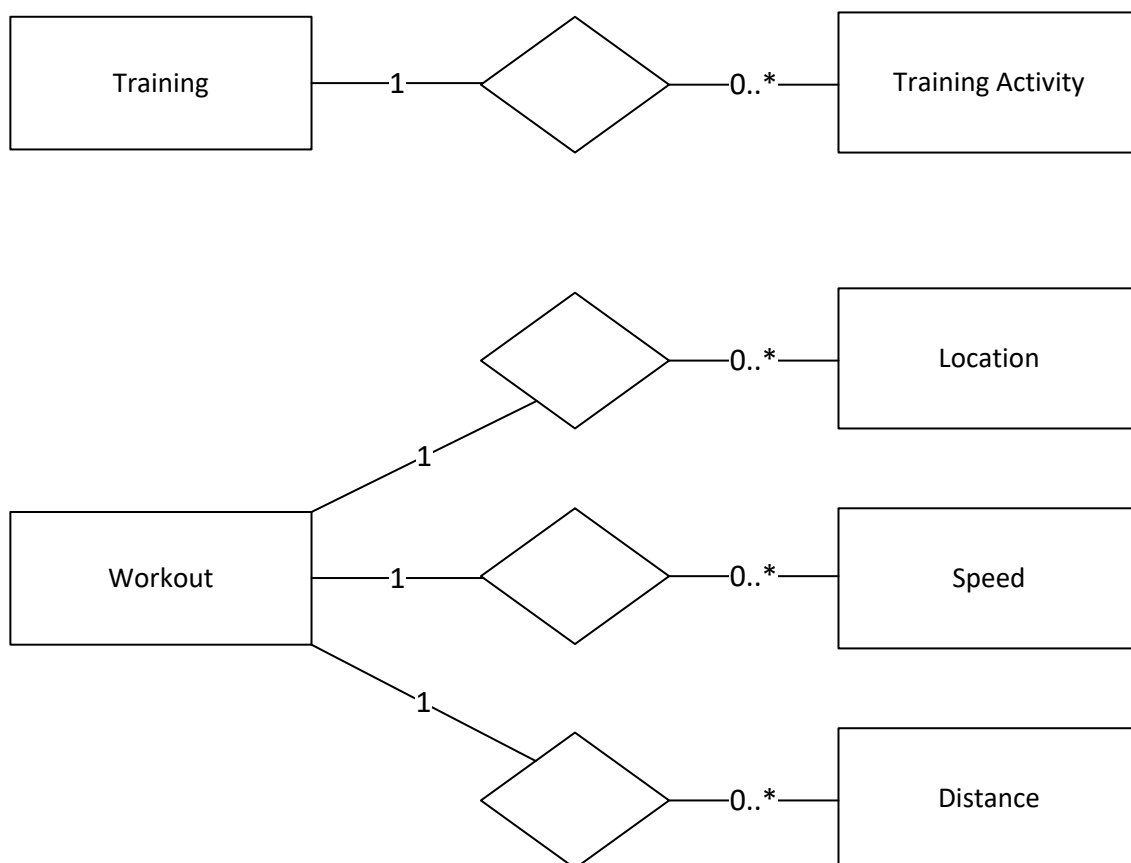
Aplikacje na Androidzie posiadają główny wątek, którym jest wątek interfejsu użytkownika. Z tego powodu bardziej wymagające operacje muszą być wykonywane na osobnym wątku w celu uniknięcia efektu zamrożenia interfejsu.



Rys. 7 - Podstawowe komponenty Material Design

## 4.5. Projekt bazy danych

### 4.5.1. Model konceptualny

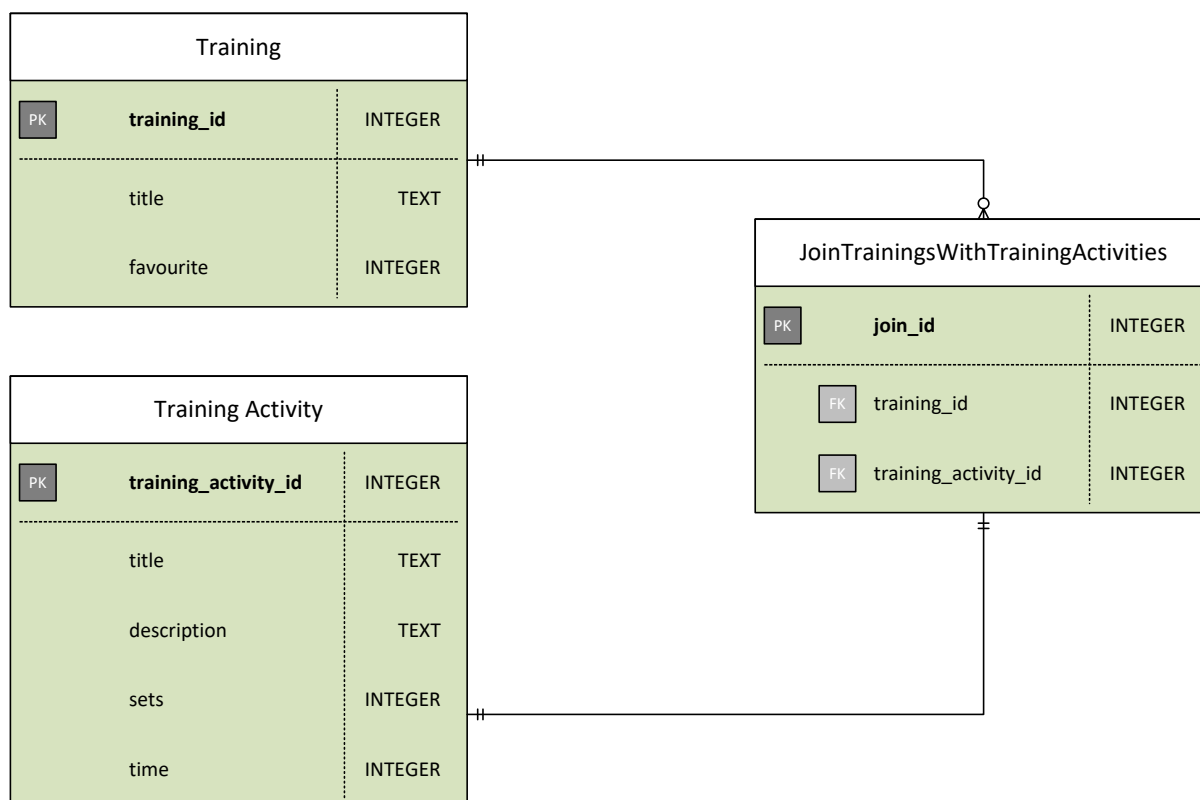


Rys. 8 - Model konceptualny bazy danych – notacja Chena

Dla modułu planów treningowych zaprojektowano model „Training” zawierający kolekcję ćwiczeń „Training Activity”. Relacja między obiema encjami to jeden do wielu.

Encja „Workout” jest ogólnym wynikiem modułu śledzenia użytkownika. Posiada relację jeden do wielu z encjami „Location”, „Speed” oraz „Distance”, które są modyfikowane przy każdej aktualizacji lokacji użytkownika.

## 4.5.2. Model fizyczny



Rys. 9 - Model fizyczny treningu i jego ćwiczeń – notacja kurzej łapki

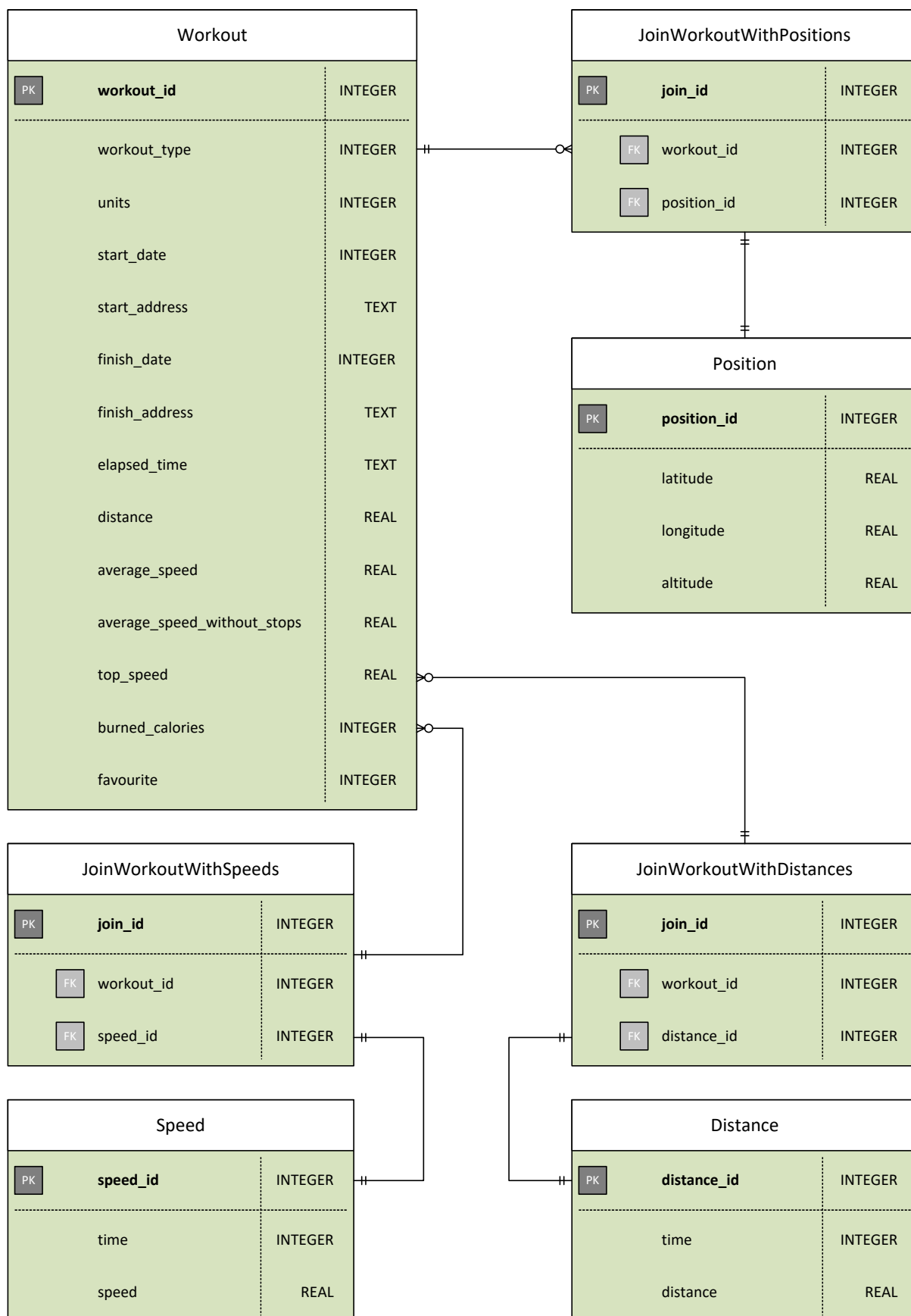
Ponieważ model treningu zawiera w sobie kolekcję ćwiczeń, wymagane było stworzenie dodatkowej tabeli asocjacyjnej „JoinTrainingsWithTrainingActivity”. Przy każdej operacji dodawania i usuwania ćwiczenia lub treningu tabela asocjacji jest również aktualizowana.

Tab. 3 - Opis atrybutów encji Training

Atrybuty „Training”	Opis
training_id	Klucz główny treningu, używany jako klucz obcy w tabeli asocjacyjnej.
title	Główna nazwa treningu.
favourite	Stan polubienia treningu, domyślnie ustawiany na false.

Tab. 4 - Opis atrybutów encji Training Activity

Atrybuty „Training Activity”	Opis
training_activity_id	Klucz główny ćwiczenia, klucz obcy w tabeli asocjacyjnej.
title	Główna nazwa ćwiczenia.
description	Opis ćwiczenia.
sets	Ilość serii lub powtórzeń ćwiczenia.
time	Czas ćwiczenia w milisekundach



Rys. 10 - Model fizyczny aktywności fizycznej – notacja kurzej łapki

Model aktywności fizycznej „Workout” zawiera w sobie kolekcje lokacji, prędkości oraz dystansów. Podobnie jak w poprzednim modelu użyte zostały encje asocjacyjne dla wspomnianych trzech kolekcji.

Tab. 5 - Opis atrybutów encji Workout

Atrybuty „Workout”	Opis
workout_id	Klucz główny aktywności, używany w każdej tabeli asocjacyjnej.
workout_type	Typ aktywności – bieganie lub kolarstwo.
units	Rodzaj jednostek danych – metryczne lub imperialne.
start_date	Data rozpoczęcia aktywności w formacie Unix.
start_address	Adres rozpoczęcia aktywności.
finish_date	Data zakończenia aktywności w formacie Unix.
finish_address	Adres zakończenia aktywności.
elapsed_time	Całkowity czas aktywności.
distance	Całkowity dystans.
average_speed	Średnia prędkość uzyskana podczas aktywności.
average_speed_without_stops	Średnia prędkość niezerowych prędkości.
top_speed	Prędkość maksymalna podczas aktywności.
burned_calories	Przybliżone spalane kilokalorie.
favourite	Stan polubienia aktywności, domyślnie ustawiany na false.

Tab. 6 - Opis atrybutów encji Position

Atrybuty „Position”	Opis
position_id	Klucz główny lokacji, używany jako klucz obcy w tabeli asocjacyjnej.
latitude	Szerokość geograficzna lokacji.
longitude	Długość geograficzna lokacji.
altitude	Wysokość geograficzna lokacji

Tab. 7 - Opis atrybutów encji Speed

Atrybuty „Speed”	Opis
speed_id	Klucz główny prędkości, używany jako klucz obcy w tabeli asocjacyjnej.
time	Czas uzyskania prędkości w formacie Unix.
speed	Uzyskana prędkość.

Tab. 8 - Opis atrybutów encji Distance

Atrybuty „Distance”	Opis
distance_id	Klucz główny dystansu, używany jako klucz obcy w tabeli asocjacyjnej.
time	Czas uzyskania dystansu.
distance	Całkowity dystans w aktualnym czasie.

### 4.5.3. Ograniczenia integralnościowe

Każda z opisywanych encji posiada klucz główny, który nie może posiadać wartości NULL oraz musi być unikalny. Klucze główne są obsługiwane i inkrementowane automatycznie przez zaimplementowany framework greenDAO, który zajmuje się integralnością bazy danych.

# 5. Implementacja systemu



Rys. 11 - Logo aplikacji Pocket Trainer

## 5.1. Implementacja bazy danych

### 5.1.1. Modele encji

Każdy model encji został zaimplementowany jako pojedyncza klasa z wykorzystaniem adnotacji obsługiwanych przez framework greenDAO. Obsługa mapowania ORM dla danej klasy z adnotacjami tworzona jest w momencie kompilacji i budowania aplikacji. Generowane jest wtedy DAO, wszystkie potrzebne metody do zarządzania tabelą (posiadającą odpowiednią adnotację „@Generated”) oraz obsługa zdefiniowanej encji asocjacyjnej.

Niniejsze listingi pozbawione są metod wygenerowanych przez framework, getterów i setterów, a atrybuty odpowiadające polom modeli opisane zostały w 4.5.2.

Listing 1 - Model encji Training – definicja klasy

```
@Entity  
public class Training {
```

Adnotacja klasy „@Entity” informuje framework o celu interpretowania jej jako model encji oraz wszystkich jej pól jako atrybuty tabeli (jeśli nie ma odpowiedniej adnotacji ignorowania).



Listing 2 - Model encji Training - definicje pól

```
@Id(autoincrement = true)
private Long id;

private String title;

private boolean favourite;
```

Pole „id” poprzez adnotację definiowane jest jako unikalny klucz główny encji, który jest automatycznie inkrementowany dla nowych rekordów w tabeli.

Listing 3 - Model encji Training - definicja kolekcji

```
@ToMany
@JoinEntity(
    entity = JoinTrainingsWithTrainingActivities.class,
    sourceProperty = "trainingId",
    targetProperty = "trainingActivityId"
)
private List<TrainingActivity> trainingActivities;
```

Kolekcja ćwiczeń treningu. W adnotacji określony został typ relacji oraz klasa będąca encją asocjacyjną między treningiem a ćwiczeniem.

Listing 4 - Model encji Training - metody

```
public Training(Long id, String title, boolean favourite) {
    this.id = id;
    this.title = title;
    this.favourite = favourite;
}

public Training() {
}

@Override
public String toString() {
    return "TrainingActivity{" +
        "id=" + id +
        ", title=" + title + '\'' +
        ", trainingActivities=" + trainingActivities + '\'' +
        ", favourite=" + favourite + '\'' +
        '}';
}
```

Metoda toString() jest używana szeroko w projekcie dla wygodniejszego debugowania (szybkie sprawdzenie zawartości obiektu).

Listing 5 - Model encji Training Activity

```
@Entity
public class TrainingActivity {

    @Id(autoincrement = true)
    private Long id;

    private String title;

    private String description;

    private int sets;

    private long time;

    public TrainingActivity(Long id, String title, String description, int
sets,
        long time) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.sets = sets;
        this.time = time;
    }
}
```

Listing 6 - Model encji JoinTrainingsWithTrainingActivities

```
@Entity
public class JoinTrainingsWithTrainingActivities {

    @Id(autoincrement = true)
    private Long id;

    private long trainingId;

    private long trainingActivityId;

    public JoinTrainingsWithTrainingActivities(Long id, long trainingId,
        long trainingActivityId) {
        this.id = id;
        this.trainingId = trainingId;
        this.trainingActivityId = trainingActivityId;
    }
}
```

Encja „JoinTrainingsWithTrainingActivities” zajmuje się asocjacją treningów oraz ich ćwiczeń. Każdy rekord w tabeli posiada indeks treningu oraz indeks ćwiczenia, który jest unikalny i powtarza się tylko raz.

Listing 7 - Model encji Workout - pola klasy standardowego typu

```
@Entity
public class Workout {

    @Id(autoincrement = true)
    private Long id;

    private String finishAddress;

    private String elapsedTime;

    private double distance;

    private double averageSpeed;

    private double averageSpeedWithoutStops;

    private double topSpeed;

    private int burnedCalories;

    private boolean favourite;
```

Listing 8 - Model encji Workout - pola klasy niestandardowego typu

```
@Convert(converter = WorkoutTypeTransformer.class, columnType =
Integer.class)
private WorkoutType workoutType;

@Convert(converter = UnitsTransformer.class, columnType = Integer.class)
private Units units;

@Convert(converter = CalendarTransformer.class, columnType = Long.class)
private Calendar startDate;

private String startAddress;

@Convert(converter = CalendarTransformer.class, columnType = Long.class)
private Calendar finishDate;
```

Pola niestandardowego typu muszą mieć zaimplementowany konwerter zajmujący się serializacją i deserializacją. W powyższym przypadku pola enum parsowane są na wartość Integer, a pola typu Calendar na odpowiadającą liczbę Long w formacie czasu Unix.

Listing 9 - Model encji Workout - kolekcje klasy

```
@ToMany
@JoinEntity(
    entity = JoinWorkoutWithPositions.class,
    sourceProperty = "workoutId",
    targetProperty = "positionId"
)
private List<Position> locationsList;

@ToMany
@JoinEntity(
    entity = JoinWorkoutWithSpeeds.class,
    sourceProperty = "workoutId",
    targetProperty = "speedId"
)
private List<Speed> speedsList;

@ToMany
@JoinEntity(
    entity = JoinWorkoutWithDistances.class,
    sourceProperty = "workoutId",
    targetProperty = "distanceId"
)
private List<Distance> distancesList;
```

Listing 10 - Model encji Position

```
@Entity
public class Position {

    @Id(autoincrement = true)
    private Long id;

    private double latitude;

    private double longitude;

    private double altitude;

    public Position(Long id, double lat, double long, double alt) {
        this.id = id;
        this.latitude = lat;
        this.longitude = long;
        this.altitude = alt;
    }
}
```

Listing 11 - Model encji JoinWorkoutWithPositions

```
@Entity
public class JoinWorkoutWithPositions {

    @Id(autoincrement = true)
    private Long id;

    private long workoutId;

    private long positionId;

    public JoinWorkoutWithPositions(Long id, long workoutId, long posId) {
        this.id = id;
        this.workoutId = workoutId;
        this.positionId = posId;
    }
}
```

Listing 12 - Model encji Speed

```
@Entity
public class Speed {

    @Id(autoincrement = true)
    private Long id;

    private long time;

    private double speed;

    public Speed(Long id, long time, double speed) {
        this.id = id;
        this.time = time;
        this.speed = speed;
    }
}
```

Listing 13 - Model encji JoinWorkoutWithSpeeds

```
@Entity
public class JoinWorkoutWithSpeeds {

    @Id(autoincrement = true)
    private Long id;

    private long workoutId;

    private long speedId;

    public JoinWorkoutWithSpeeds(Long id, long workoutId, long speedId) {
        this.id = id;
        this.workoutId = workoutId;
        this.speedId = speedId;
    }
}
```

Listing 14 - Model encji Distance

```
@Entity
public class Distance {

    @Id(autoincrement = true)
    private Long id;

    private long time;

    private double distance;    }

    public Distance(Long id, long time, double distance) {
        this.id = id;
        this.time = time;
        this.distance = distance;
    }
}
```

Listing 15 - Model encji JoinWorkoutWithDistances

```
@Entity
public class JoinWorkoutWithDistances {

    @Id(autoincrement = true)
    private Long id;

    private long workoutId;

    private long distanceId;

    public JoinWorkoutWithDistances(Long id, long workoutId, long
distanceId) {
        this.id = id;
        this.workoutId = workoutId;
        this.distanceId = distanceId;
    }
}
```

### 5.1.2. Mapowanie obiektowo-relacyjne

Mapowanie ORM odbywa się w większości poprzez opisywany wcześniej framework greenDAO. Klasa oznaczona adnotacją encji `@Entity` jest mapowana ze wszystkimi zdefiniowanymi polami. Pola niestandardowego typu wymagają dodatkowo implementacji transformerów, które konwertują obiekt do typu standardowego, interpretowanego przez język SQLite. Kolekcje wymagają osobnych encji asocjacyjnych (opisywanych w poprzednim punkcie) pozwalających na relację „do wielu”.

Listing 16 - Transformer typów enum - WorkoutType

```
public class WorkoutTypeTransformer implements
PropertyConverter<WorkoutType, Integer> {

    @Override
    public WorkoutType convertToEntityProperty(Integer databaseValue) {
        if (databaseValue == null) {
            return null;
        }
        for (WorkoutType workoutType : WorkoutType.values()) {
            if (workoutType.getId() == databaseValue) {
                return workoutType;
            }
        }
        return null;
    }

    @Override
    public Integer convertToDatabaseValue(WorkoutType entityProperty) {
        if (entityProperty == null) {
            return null;
        } else {
            return entityProperty.getId();
        }
    }
}
```

Metoda `convertToEntityProperty(Integer databaseValue)` konwertuje parametr `Integer` otrzymywany z bazy danych na obiekt enum zdefiniowany w encji.

Metoda `convertToDatabaseValue(WorkoutType entityProperty)` zwraca `Integer` na podstawie obiektu z typem enum.

Działanie poniższego transformera `UnitsTransformer` opiera się na takiej zasadzie.

Listing 17 - Transformer typów enum - Units

```
public class UnitsTransformer implements PropertyConverter<Units, Integer>
{
    @Override
    public Units convertToEntityProperty(Integer databaseValue) {
        if (databaseValue == null) {
            return null;
        }
        for (Units unit : Units.values()) {
            if (unit.getId() == databaseValue) {
                return unit;
            }
        }
        return null;
    }

    @Override
    public Integer convertToDatabaseValue(Units entityProperty) {
        if (entityProperty == null) {
            return null;
        } else {
            return entityProperty.getId();
        }
    }
}
```

Listing 18 - Transformer typu niestandardowego - Calendar

```
public class CalendarTransformer implements PropertyConverter<Calendar,
Long> {
    @Override
    public Calendar convertToEntityProperty(Long databaseValue) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(databaseValue);
        return calendar;
    }

    @Override
    public Long convertToDatabaseValue(Calendar entityProperty) {
        return entityProperty.getTimeInMillis();
    }
}
```

Metoda `convertToEntityProperty(Long databaseValue)` zwraca obiekt `Calendar`, zdefiniowany w encji, na podstawie parametru `Long`.

Metoda `convertToDatabaseValue(Calendar entityProperty)` konwertuje wartość encji typu `Calendar` do odpowiadającej wartości typu `Long` w formacie czasu Unix.

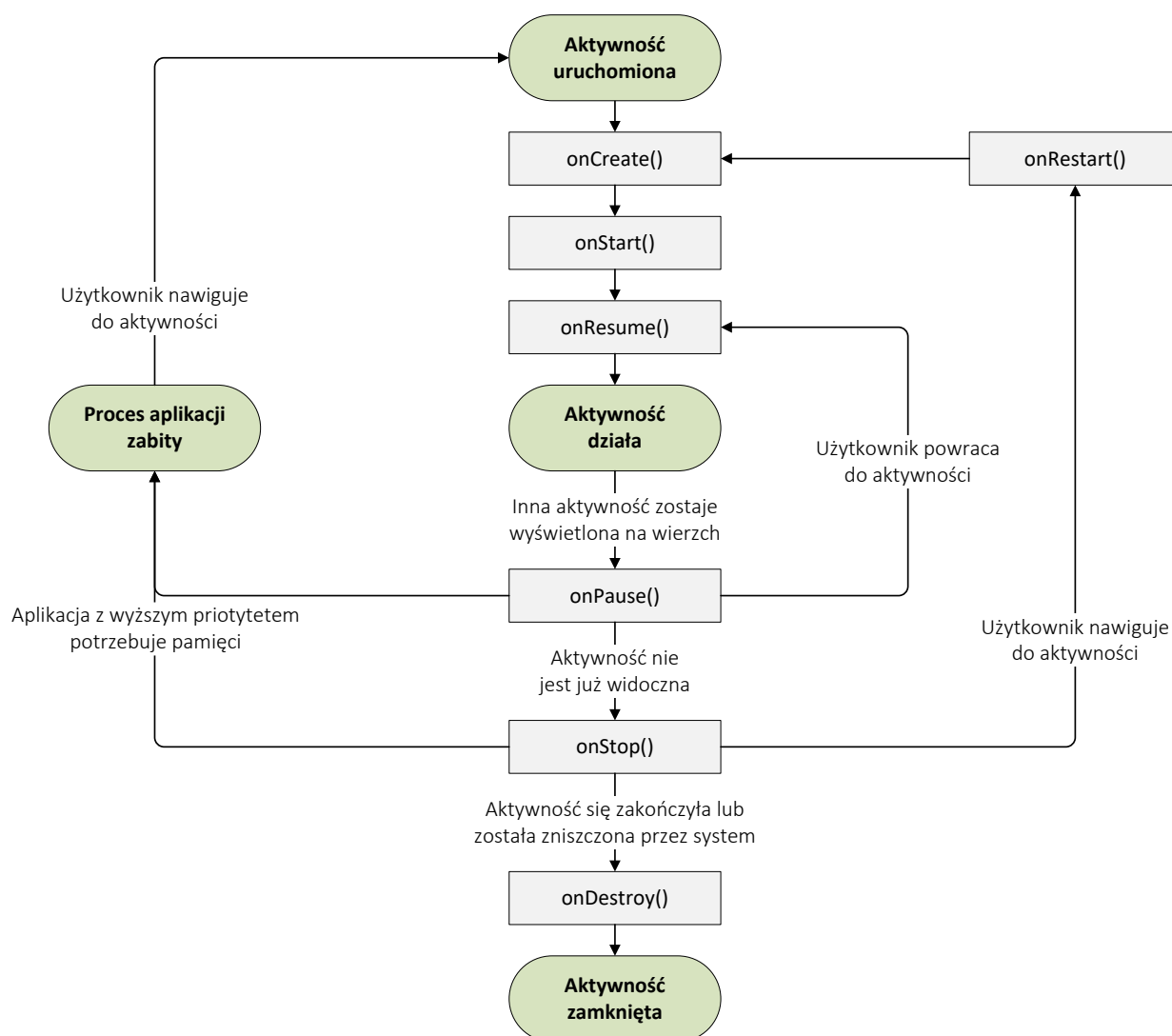


## 5.2. Implementacja modułów aplikacji

### 5.2.1. Definicje komponentów aplikacji

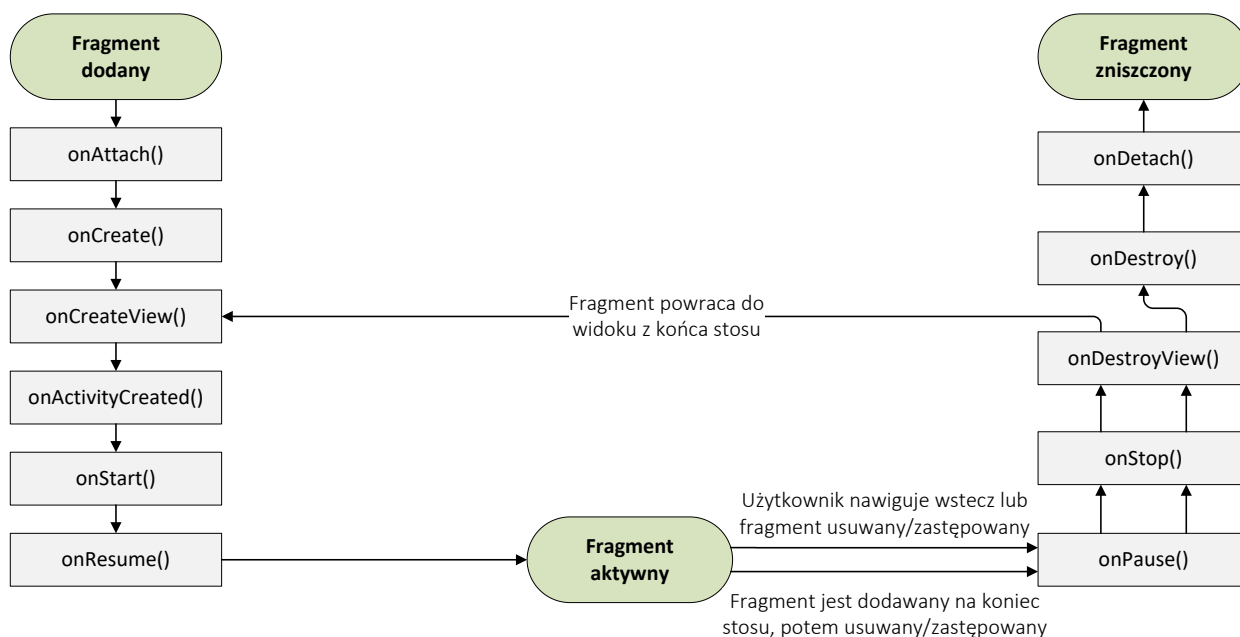
Aktywność jest najważniejszym komponentem aplikacji na systemie Android. Ma za zadanie dostarczenie ekranu widoku, z który pozwala na interakcję z użytkownikiem.

Aplikacje zazwyczaj posiadają więcej niż jedną aktywność, które są luźno związane ze sobą. Każda aktywność może rozpocząć inną, aby umożliwić użytkownikowi inne akcje. Jednocześnie, kiedy jedna aktywność zostanie rozpoczęta, poprzednia przechodzi w stan zatrzymania i system przetrzymuje ją na własnym stosie opierającym się na zasadzie „last in, first out”. Oznacza to, że kiedy użytkownik kończy jedną aktywność oraz wcisnie przycisk powrotu, jest ona brana ze stosu, niszczona i przywracana jest poprzednia aktywność.



Rys. 12 - Cykl życia aktywności

Fragment reprezentuje porcję widoku i logiki. Fragmenty można łączyć w jednej aktywności oraz używać wielokrotnie w różnych aktywnościach. Są one w pewien sposób nakładką na aktywność z własnym cyklem życia, layoutem oraz logiką. Można je dowolnie usuwać i dodawać podczas działania aktywności, od której są całkowicie zależne. Oznacza to, że każdy stan aktywności, taki jak zatrzymanie lub zniszczenie, ma wpływ na stan przywiązanych do niego fragmentów.



Rys. 13 - Cykl życia fragmentu

Serwis jest ostatnim ważnym komponentem aplikacji, który zajmuje się bardzo długimi operacjami, działa w tle i nie dostarcza żadnego interfejsu użytkownika. Może zostać uruchomiony przy starcie aplikacji lub w dowolnej chwili działania przez inny komponent. Od tego momentu serwis kontynuuje pracę w tle nawet jeśli użytkownik przełączy się do innej aplikacji lub wyłączy ekran urządzenia. Dodatkowo, inny komponent może się powiązać z serwisem w celu bezpośredniej komunikacji. W niniejszej pracy został użyty jeden serwis zajmujący się śledzeniem lokalizacji użytkownika. Jest on powiązany z aktywnością oraz fragmentem, któremu wysyła dane do wyświetlenia na interfejsie użytkownika.

## 5.2.2. Definicje komponentów interfejsu

Najważniejsze, niestandardowe komponenty interfejsu graficznego użyte w aplikacji:

- **Toolbar**  
Górny pasek narzędziowy umożliwiający powrót do poprzedniej aktywności lub własne, zdefiniowane akcje, np. skrót do ustawień. W aplikacji używany w każdej aktywności.
- **Navigation Drawer**  
Wysuwany panel od lewej strony ekranu pełniący rolę menu. Umożliwia nawigację pomiędzy głównymi funkcjonalnościami aplikacji.
- **Recycler View**  
Kontener wyświetlający elementy w formie zaawansowanej listy. Wymaga implementacji adaptera, który zajmuje się obsługą wyświetlanego interfejsu oraz danych. Podczas przewijania listy elementy są tworzone i usuwane na bieżąco w celu jak największej wydajności.
- **View Pager**  
Kontener umożliwiający trzymanie kilku fragmentów w formie zakładek. Przesuwanie palca w lewo lub w prawo przełącza fragmenty.
- **Card View**  
Jest to kontener wyświetlający dane, które powinny być skupione w jednym miejscu lub bardziej widoczne od pozostałych. Może zawierać w sobie inne komponenty.
- **Dialog**  
Okno dialogowe informujące użytkownika o konkretnej informacji lub zapytaniu. Może zostać zaimplementowany jako fragment z niestandardowymi komponentami.
- **Bottom Sheet**  
Wysuwany panel od dołu ekranu. Zawiera w sobie fragment z niestandardowym wyglądem. W aplikacji użyty jako interfejs z przyciskami na ekranie mapy.
- **Floating Action Button**  
Pełni rolę przycisku z akcją, która powinna być bardziej wyszczególniona. Reprezentowany przez okrągły przycisk, unoszący się ponad resztę interfejsu użytkownika. W aplikacji zaimplementowany w celu dodawania i usuwania elementów na liście.

### 5.2.3. Moduł głównego menu

W niniejszym punkcie oraz w kolejnych zostaną przedstawione najważniejsze fragmenty kodu wraz z rzutami ekranu reprezentującymi widoki.

Aktywność uruchamiana przy starcie aplikacji pełni rolę głównego kontenera, który wyświetla element wybrany z wysuwanego menu. Wszystkie elementy to fragmenty odpowiadające głównym funkcjonalnościom aplikacji.

Listing 19 - Moduł głównego menu - zmiana fragmentów

```
@Override
public void changeFragment(final Fragment fragment, boolean useDelay) {
    mCurrentFragment = fragment;
    final FragmentManager fragmentManager = getSupportFragmentManager();
    Fragment oldFragment =
fragmentManager.findFragmentById(FRAGMENT_CONTAINER);
    if (oldFragment != null) {
        fragmentManager
            .beginTransaction()
            .setCustomAnimations(R.anim.slide_in_right,
R.anim.slide_out_right, R.anim.slide_in_right, R.anim.slide_out_right)
            .remove(oldFragment)
            .commit();
    }
    Handler handler = new Handler();
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            fragmentManager
                .beginTransaction()
                .setCustomAnimations(R.anim.slide_in_right,
R.anim.slide_out_right, R.anim.slide_in_right, R.anim.slide_out_right)
                .add(FRAGMENT_CONTAINER, fragment)
                .commit();
        }
    }, useDelay ? 500 : 0);
}
```

Metoda zmiany fragmentu korzysta ze zdefiniowanych animacji pojawiania i zanikania oraz z menedżera fragmentów. Stary fragment (jeśli istnieje) zostaje usunięty, a na jego miejsce pojawia się nowy. Zostaje tu użyty nowy wątek trwający 500 milisekund w celu płynnego przejścia.

Listing 20 - Moduł głównego menu - listener elementu menu

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.nav_home) {
        setTitle(R.string.app_name);
    } else {
        setTitle(item.getTitle());
    }

    if (!item.isChecked()) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            mAppBarLayout.setElevation(DEFAULT_APP_BAR_ELEVATION);
        }
        switch (item.getItemId()) {
            case R.id.nav_home:
                changeFragment(HomeFragment.newInstance(), true);
                break;
            case R.id.nav_planner:
                changeFragment(GymTrainingsListFragment.newInstance(), true);
                break;
            case R.id.nav_workout:
                changeFragment(NewWorkoutFragment.newInstance(), true);
                break;
            case R.id.nav_history:
                changeFragment(WorkoutsHistoryFragment.newInstance(), true);
                break;
            case R.id.nav_bmi:
                changeFragment(BmiFragment.newInstance(), true);
                break;
            case R.id.nav_bfp:
                changeFragment(BfpFragment.newInstance(), true);
                break;
            case R.id.nav_quit:
                finish();
                break;
        }
    }

    mDrawerLayout.closeDrawer(GravityCompat.START);
    return true;
}
```

Powyższa metoda zawarta jest w zaimplementowanym interfejsie klasy – `OnNavigationItemSelectedListener`. Nasłuchuje on każde kliknięcie elementu w wysuwanym menu oraz przeprowadza odpowiednią akcję, którą jest zmiana odpowiedniego fragmentu, podświetlenie elementu w menu oraz ustawienie nazwy aktywności, która jest wyświetlona na toolbarze (górnym pasku narzędziowym).

Wartym uwagi jest fragment z instrukcją warunkową:

Listing 21 - Moduł głównego menu - sprawdzenie wersji systemu

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {  
    mAppBarLayout.setElevation(DEFAULT_APP_BAR_ELEVATION);  
}
```

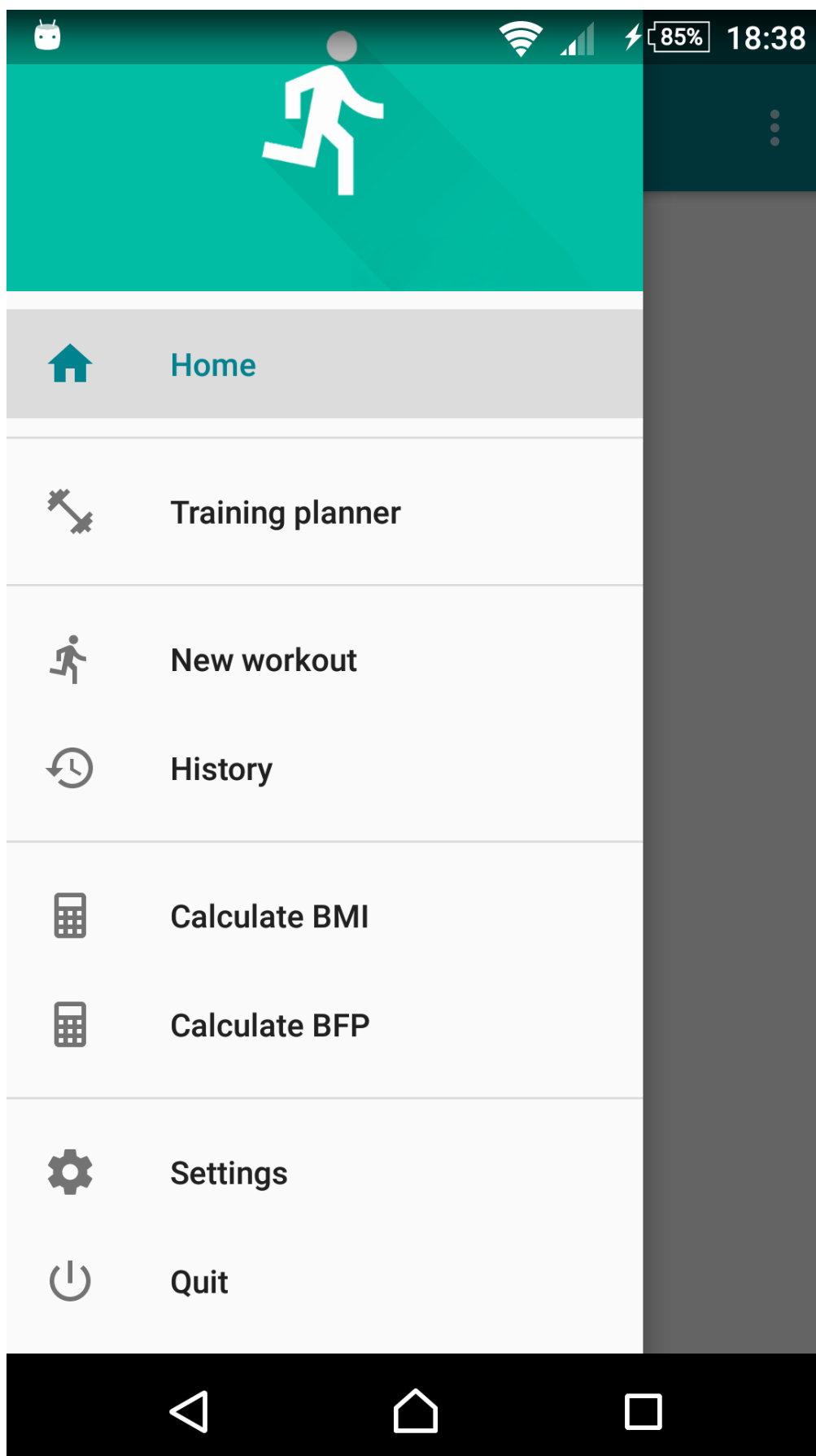
Sprawdza on wersję zainstalowanego systemu, na którym działa aplikacja. Jeśli aktualną wersją jest Android 5.0 Lollipop (API 21, które wprowadziło Material Design do środowiska) lub nowszy – zostaje ustawione podwyższenie (ang. *elevation*) górnego paska narzędziowego zgodnie z zaleceniami Google o tworzeniu interfejsu. Jest to spowodowane faktem, że aplikacja jest w stanie także działać na systemach o wersji niższej niż 5.0 Lollipop.

Listing 22 - Moduł głównego menu - obsługa kliknięcia strzałki powrotu

```
@Override  
public void onBackPressed() {  
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);  
    if (drawer.isDrawerOpen(GravityCompat.START)) {  
        drawer.closeDrawer(GravityCompat.START);  
    } else if (!(mCurrentFragment instanceof HomeFragment)) {  
        mNavigationView.getMenu().findItem(R.id.nav_home).setChecked(true);  
        setTitle(R.string.app_name);  
        changeFragment(HomeFragment.newInstance(), true);  
    } else {  
        super.onBackPressed();  
    }  
}
```

Kliknięcie strzałki powrotu na pasku narzędziowym powoduje jedną z trzech akcji:

- Jeśli otwarty jest navigation drawer – zamknięcie go
- Jeśli aktualny fragment nie jest główny – cofnięcie do fragmentu domowego
- Wyłączenie aplikacji



Rys. 14 - Zrzut ekranu głównego menu

## 5.2.4. Moduł planowania treningów

Fragment wyświetlający treningi zapisane w bazie danych uruchamiany jest poprzez wysuwane menu. Zawiera on w sobie jedynie komponent RecyclerView zajmujący się wyświetlaniem listy treningów oraz Floating Action Button, który obsługuje akcje dodawania i usuwania elementu z listy. Każde kliknięcie na element listy przenosi użytkownika do nowej aktywności z ćwiczeniami danego treningu.

Listing 23 - Moduł planowania treningów - pobieranie listy z bazy danych

```
public void setTrainings() {
    List<Training> trainingsList = mTrainingDao.loadAll();

    for (Training training : trainingsList) {
        mTrainingsList.add(new TrainingItem(training));
    }

    mAdapter.setTrainingsList(mTrainingsList);
    mAdapter.notifyDataSetChanged();
}
```

Metoda pobierania treningów z lokalnej bazy danych wywoływana jest przy starcie fragmentu. Wykorzystując referencję do DAO treningów ładowane są wszystkie rekordy z tabeli i przypisywane do kolekcji. Elementy kolekcji są parsowane na typ „TrainingItem”, który dodatkowo obsługuje zaznaczanie i odznaczanie elementu, po czym przypisywane do kolekcji klasy „mTrainingsList”. Następnie adapter listy otrzymuje kolekcję i jest powiadamiany o zmianie zbioru danych.

Listing 24 - Moduł planowania treningów - ustawianie elementu listy

```
public void setTraining(@NonNull Training training) {
    mDescriptionTextView.setText(training.getTitle());
    int activities = training.getTrainingActivities().size();
    if (activities > 1) {

        mActivitiesTextView.setText(String.format(TrainerApplication.getContext().getString(R.string.training_activities_num), activities));
    } else if (activities == 1) {
        mActivitiesTextView.setText(R.string.training_activities_one);
    } else {
        mActivitiesTextView.setText(R.string.training_activities_zero);
    }
}
```

Powyższa metoda ustawia widok pojedynczego elementu. Wyświetlany jest tytuł treningu oraz rozmiar kolekcji ćwiczeń w obiekcie „Training”.



Listing 25 - Moduł planowania treningów - dodawanie treningu do bazy danych

```
public void addNewTraining(Training training) {
    mTrainingDao.insert(training);
    mTrainingsList.add(new TrainingItem(training));
    mAdapter.notifyItemInserted(mTrainingsList.size() - 1);
}
```

Dodawanie nowego treningu wywoływane jest po wciśnięciu przycisku Floating Action Button oraz po wypełnieniu okna dialogowego. Do referencji DAO treningów wkładany jest nowy obiekt, który został wcześniej stworzony i przekazany z oknie dialogowym. Obiekt treningu jest także dodawany do kolekcji klasy. Adapter listy jest powiadamiany o zmianie zbioru danych, po czym następuje animacja dodania nowego elementu do wyświetlonej listy.

Listing 26 - Moduł planowania treningów - usuwanie treningu z bazy danych

```
public void deleteTraining(final TrainingItem item) {
    mTrainingDao.deleteByKey(item.getObject().getId());
    QueryBuilder queryBuilder =
    TrainerApplication.getDaoSession().getJoinTrainingsWithTrainingActivitiesDa
    o().queryBuilder();

    queryBuilder.where(JoinTrainingsWithTrainingActivitiesDao.Properties.Traini
ngId.eq(item.getObject().getId()));
    List<JoinTrainingsWithTrainingActivities>
    joinTrainingsWithTrainingActivitiesList = queryBuilder.list();
    for (JoinTrainingsWithTrainingActivities itemJoin :
    joinTrainingsWithTrainingActivitiesList) {

        TrainerApplication.getDaoSession().getTrainingActivityDao().deleteByKey(ite
        mJoin.getTrainingActivityId());

        TrainerApplication.getDaoSession().getJoinTrainingsWithTrainingActivitiesDa
        o().deleteByKey(itemJoin.getId());
    }

    int index = mTrainingsList.indexOf(item);
    if (index >= 0) {
        mTrainingsList.remove(index);
        mAdapter.notifyItemRemoved(index);
    }
    showAddButton();
}
```

Metoda usuwania treningu z bazy danych wymaga więcej operacji niż dodawanie z racji występowania relacji „do wielu” z inną tabelą. Każde ćwiczenie przypisane do usuwanego treningu wymaga usunięcia z bazy danych ćwiczeń oraz z tabeli asocjacyjnej. Po usunięciu treningu metodą deleteByKey(), w pętli for usuwane są także ćwiczenia. Każde ćwiczenie

posiadające w encji asocjacyjnej usuwany trening jest usuwane poprzez referencję do „TrainingActivityDao” oraz „JoinTrainingsWithTrainingActivitiesDao”.

Listing 27 - Moduł planowania treningów - okno dialogowe dodawania

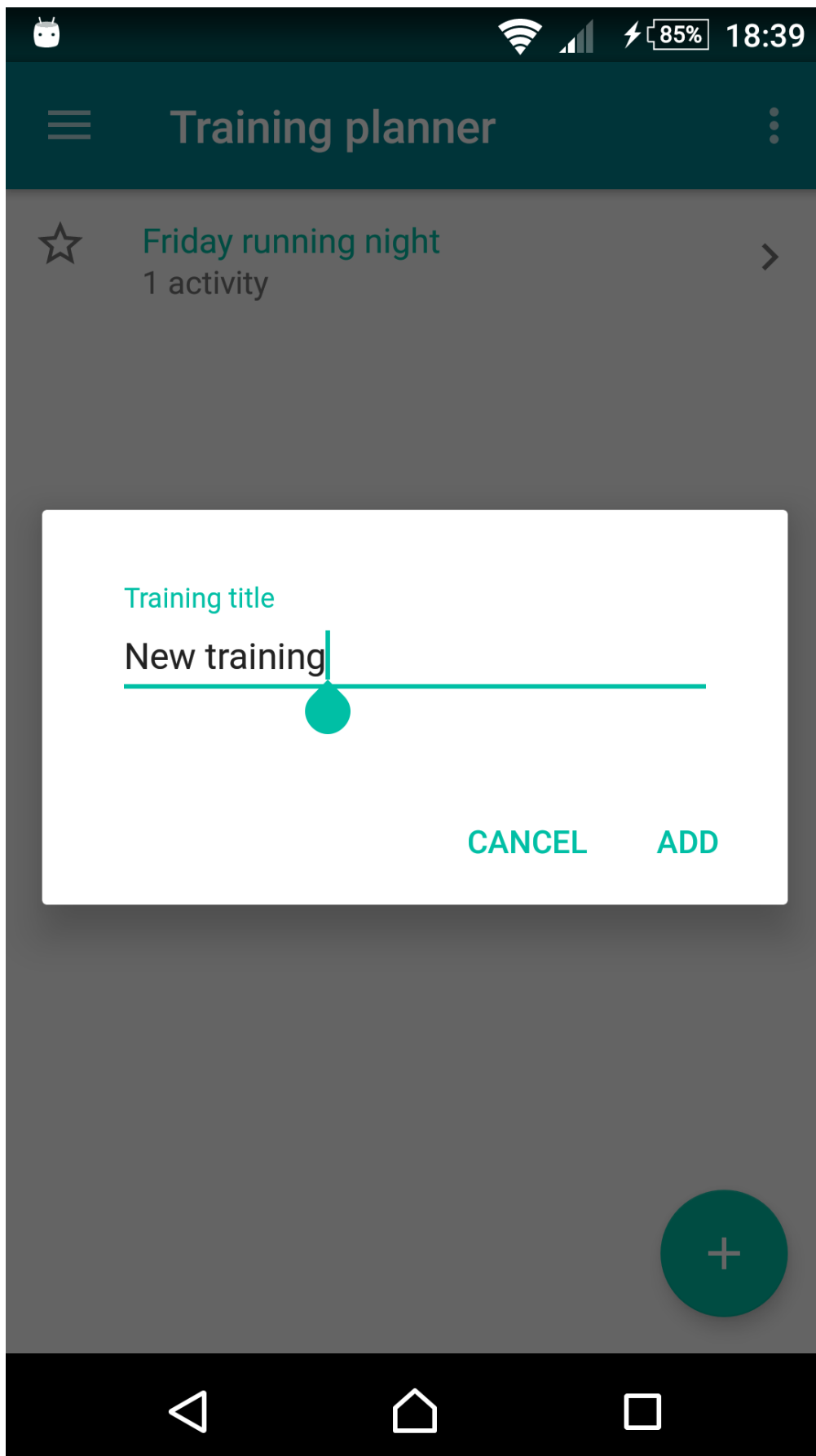
```
public void showNewTrainingDialog() {
    final AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity(), R.style.TrainerTheme_Dialog);
    LayoutInflater inflater = LayoutInflater.from(getActivity());
    View dialogView = inflater.inflate(R.layout.dialog_new_training, null);
    final EditText trainingTitleEditText = (EditText)
dialogView.findViewById(R.id.training_edit_title);
    builder.setView(dialogView)
        .setPositiveButton(R.string.add, new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                Training training = new Training(null,
trainingTitleEditText.getText().toString(), false);
                addNewTraining(training);
            }
        })
        .setNegativeButton(R.string.cancel, null);
    AlertDialog dialog = builder.create();
    dialog.setCancelable(false);
    dialog.show();
}
```

Po wciśnięciu na Floating Action Button wyświetlane jest okno dialogowe z polem do wypełnienia zawierające nazwę treningu. Kliknięcie na ikonę „Add” skutkuje przekazaniem obiektu „Training” do metody dodającej rekord do bazy danych.

Listing 28 - Moduł planowania treningów - okno dialogowe usuwania

```
public void showConfirmDeleteDialog(final TrainingItem item) {
    mDialog = new AlertDialog.Builder(getContext(),
R.style.TrainerTheme_Dialog)
        .setTitle(R.string.delete_training_title)
        .setMessage(R.string.delete_training_msg)
        .setCancelable(true)
        .setPositiveButton(R.string.delete, new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                deleteTraining(item);
            }
        })
        .setNegativeButton(R.string.cancel, null)
        .create();
    mDialog.show();
}
```

Okno dialogowe usuwania wywołuje opisywaną wcześniej metodę deleteTraining().



Rys. 15 - Zrzut ekranu dodawania nowego treningu

Listing 29 - Moduł planowania treningów - przejście do nowej aktywności

```
public void openTrainingActivity(Training training) {
    Intent intent = new Intent(getActivity(), TrainingActivity.class);
    intent.putExtra(TrainingActivity.EXTRA_TRAINING, training.getId());
    intent.putExtra(TrainingActivity.EXTRA_TITLE, training.getTitle());
    startActivity(intent);
}
```

Kliknięcie w danych element na liście treningów przenosi użytkownika do nowej aktywności oraz fragmentu. Przekazywana jest nazwa treningu, która jest ustawiana na toolbarze w aktywności oraz indeks „id”, dzięki któremu trening jest odnajdywany we fragmencie z bazy danych.

Listing 30 - Moduł planowania treningów - pobieranie ćwiczeń z bazy danych

```
private void getTrainingFromDao(long id) {
    TrainingDao trainingDao =
    TrainerApplication.getDaoSession().getTrainingDao();
    mJoinTrainingsWithTrainingActivitiesDao =
    TrainerApplication.getDaoSession().getJoinTrainingsWithTrainingActivitiesDao();
    mTrainingActivityDao =
    TrainerApplication.getDaoSession().getTrainingActivityDao();

    QueryBuilder queryBuilder =
    trainingDao.queryBuilder().where(TrainingDao.Properties.Id.eq(id));

    Logger.error(mJoinTrainingsWithTrainingActivitiesDao.loadAll().size());
    mTraining = (Training) queryBuilder.unique();
}
```

Po otrzymaniu „id” treningu wywoływana jest metoda, w której pobierany jest trening, jego asocjacje do ćwiczeń oraz ćwiczenia, które są wyświetlane na liście.

Listing 31 - Moduł planowania treningów - dodawanie nowego ćwiczenia

```
public void addNewTrainingActivity(TrainingActivity trainingActivity) {
    mTrainingActivityDao.insert(trainingActivity);
    mJoinTrainingsWithTrainingActivitiesDao.insert(new
    JoinTrainingsWithTrainingActivities(null, mTraining.getId(),
    trainingActivity.getId()));
    mActivitiesList.add(new TrainingActivityItem(trainingActivity));
    mAdapter.notifyItemInserted(mActivitiesList.size() - 1);
}
```

Dodawanie nowego ćwiczenia do bazy danych wymaga także stworzenia nowego rekordu w tabeli asocjacji z obecnie wybranym treningiem.

Listing 32 - Moduł planowania treningów - usuwanie ćwiczenia

```
public void deleteTraining(final TrainingActivityItem item) {
public void deleteTraining(final TrainingActivityItem item) {
    QueryBuilder queryBuilder =
mJoinTrainingsWithTrainingActivitiesDao.queryBuilder();

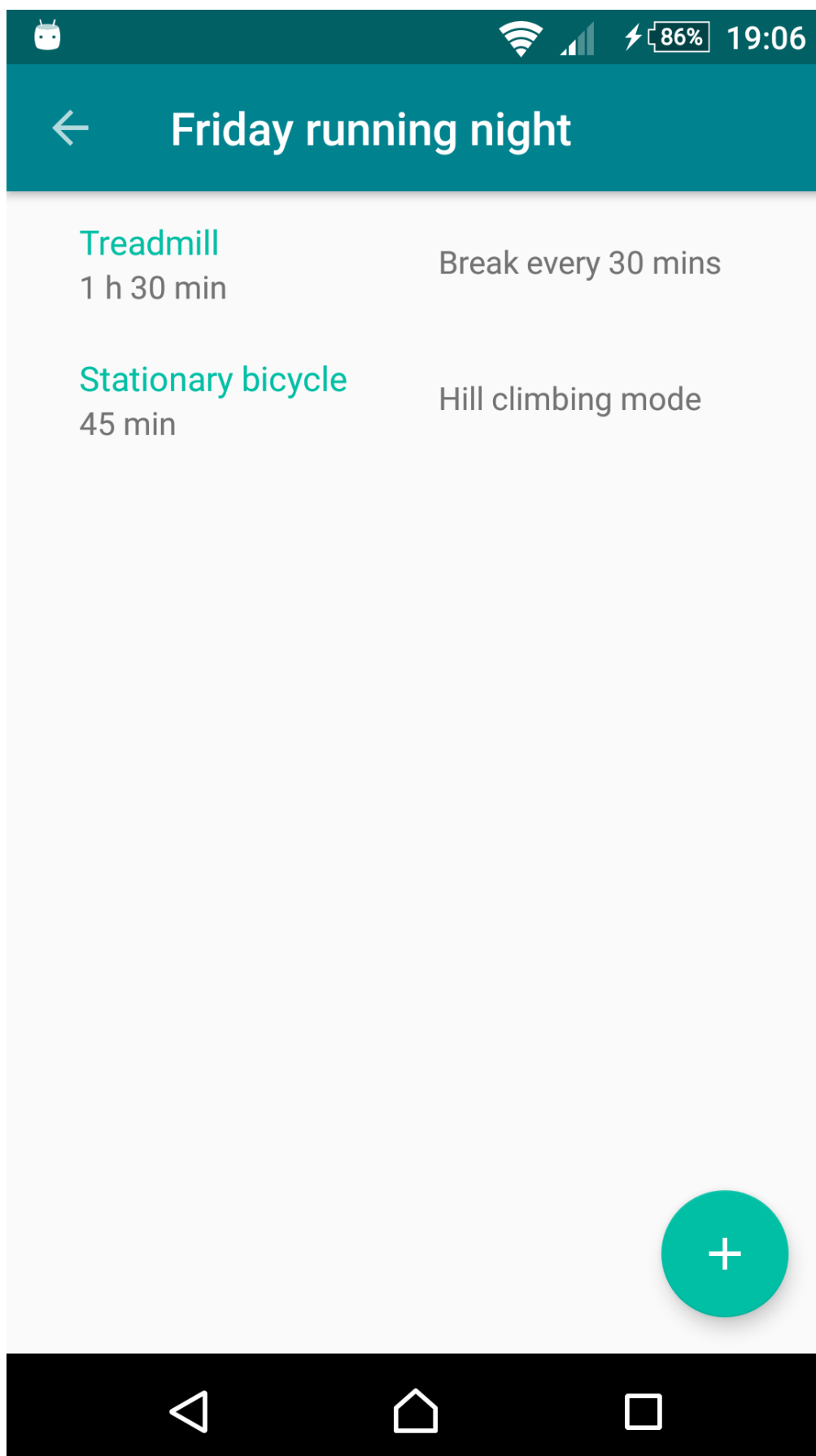
queryBuilder.and(JoinTrainingsWithTrainingActivitiesDao.Properties.Training
Id.eq(mTraining.getId()),

JoinTrainingsWithTrainingActivitiesDao.Properties.TrainingActivityId.eq(ite
m.getObject().getId()));
    JoinTrainingsWithTrainingActivities joinTrainingsWithTrainingActivities
= (JoinTrainingsWithTrainingActivities) queryBuilder.list().get(0);
    int index = mActivitiesList.indexOf(item);

    if (joinTrainingsWithTrainingActivities != null && index >= 0) {
        mTrainingActivityDao.deleteByKey(item.getObject().getId());

mJoinTrainingsWithTrainingActivitiesDao.deleteByKey(joinTrainingsWithTraini
ngActivities.getId());
        mActivitiesList.remove(index);
        mAdapter.notifyItemRemoved(index);
    }
    showAddButton();
}
```

Usunięcie ćwiczenia odbywa się poprzez usunięcie jego rekordu w tabeli ćwiczeń oraz jego asocjacji do aktualnego treningu.



Rys. 16 - Zrzut ekranu listy ćwiczeń danego treningu

## 5.2.5. Moduł śledzenia użytkownika

Kolejną pozycją w wysuwanym menu jest fragment mający za zadanie przygotowanie do uruchomienia nowej aktywności fizycznej. Użytkownik wybiera jeden z dwóch typów aktywności (kolarstwo, bieganie), jednostki metryczne lub imperialne oraz opcjonalnie tryb mniejszej dokładności pozwalający na mniejsze zużycie baterii, ale dostarczający mniej dokładne dane z uwagi na większy interwał czasu między aktualizacjami lokalizacji.

Listing 33 - Moduł śledzenia użytkownika - listener zmiany jednostek

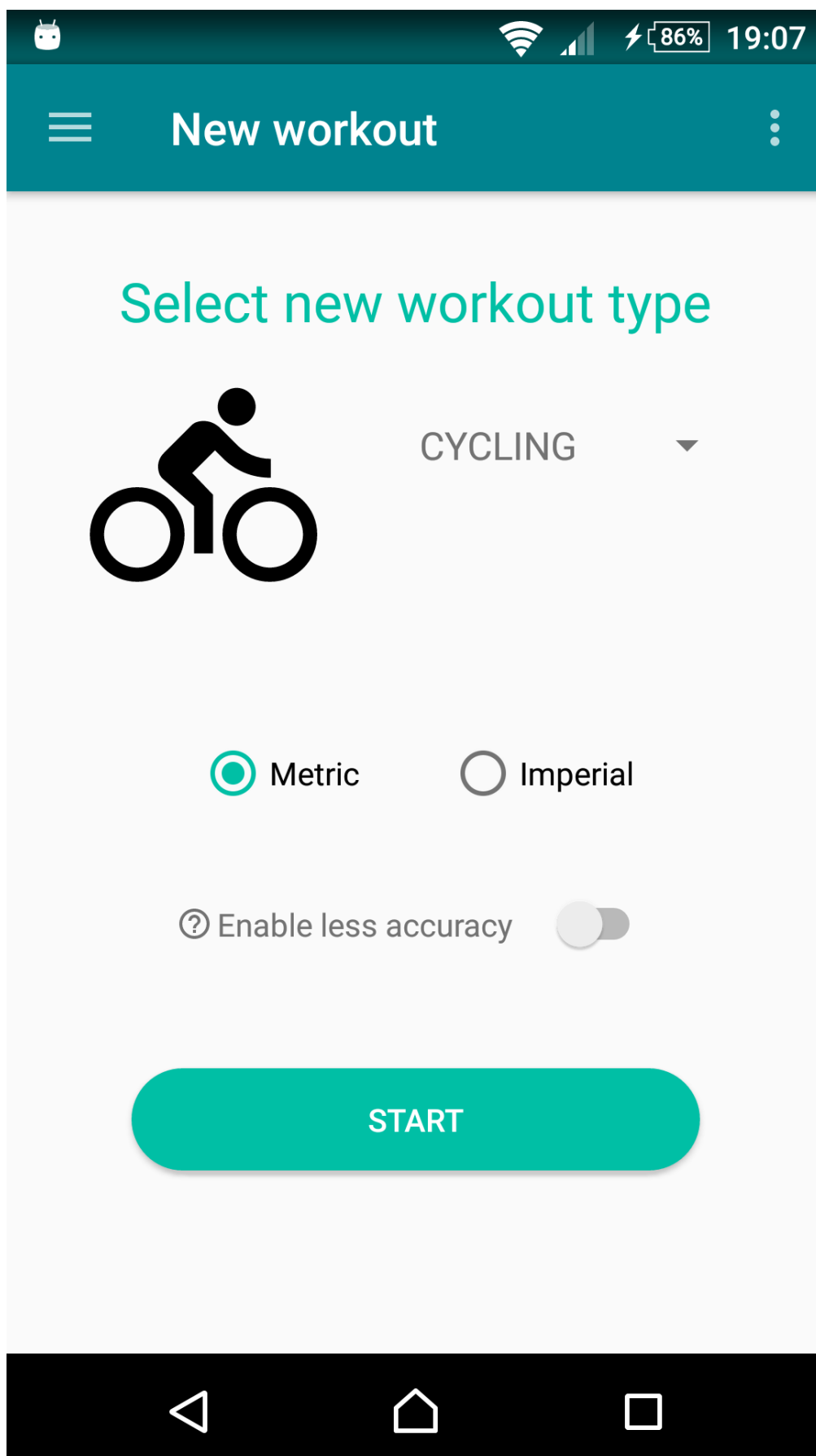
```
unitsRadioGroup.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup radioGroup, int i) {
        if(i == R.id.units_metric) {
            mSettingsPreferences.setMetricUnitsEnabled(true);
        } else if(i == R.id.units_imperial) {
            mSettingsPreferences.setMetricUnitsEnabled(false);
        }
    }
});
```

Każda zmiana jednostki metrycznej jest zachowywana w pamięci cache jako stan true/false.

Listing 34 - Moduł śledzenia użytkownika - sprawdzenie uprawnień GPS

```
private View.OnClickListener onStartListener = new View.OnClickListener()
{
    @Override
    public void onClick(View v) {
        if (checkGpsPermission()) {
            startMapActivity();
        } else {
            requestPermissions(new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
REQUEST_PERMISSION_LOCATION);
        }
    }
};
```

Listener kliknięcia w przycisk „Start” sprawdza czy aplikacja posiada wymagane uprawnienia do używania systemu lokalizacji w urządzeniu. Jeśli nie – wyświetlane jest okno dialogowe pozwalające nadać uprawnienia. Jeśli wszystko jest w porządku uruchamiana jest nowa aktywność.



Rys. 17 - Zrzut ekranu rozpoczynania nowej aktywności



Nowa aktywność zawiera fragment odpowiedzialny za wyświetlanie mapy oraz widoku przebiegu aktywności fizycznej. Przy dolnej krawędzi ekranu znajduje się wysuwany Bottom Sheet ze statystykami oraz przyciskami pauzowania, wznowiania i kończenia aktywności fizycznej.

Listing 35 - Moduł śledzenia użytkownika - tworzenie aktywności

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps_workout);
    mLoadingView = findViewById(R.id.loading_foreground);
    initWorkout();
    initToolbar();
    animateLoading(true);

    if (savedInstanceState == null) {
        changeFragment(WorkoutTracingFragment.newInstance(), false);
    }
    Intent intent = new Intent(this, LocationService.class);
    bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
}
```

Po zainicjalizowaniu obiektu Workout oraz paska narzędziowego wyświetlany jest widok ładowania, który trwa do czasu znalezienia sygnału GPS. Uruchamiany jest także serwis odpowiedzialny za komunikację z API lokalizacji, po czym jest łączony z aktywnością. Oznacza to, że serwis działa tak długo jak istnieje aktywność. Umożliwia to lepszą komunikację między serwisem, aktywnością i fragmentem.

Listing 36 - Moduł śledzenia użytkownika - tworzenie serwisu

```
@Override
public void onCreate() {
    super.onCreate();

    if (mGoogleApiClient == null) {
        mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
    }

    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(UPDATE_INTERVAL_IN_MILLISECONDS);
    mLocationRequest.setFastestInterval(FATEST_UPDATE_INTERVAL_IN_MILLISECONDS);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

Podczas tworzeniu serwisu inicjalizowany jest klient `GoogleApiClient` i żądanie lokalizacji, do którego dodawany jest interwał aktualizowania oraz listenery reagujące na nową lokalizację.

Listing 37 - Moduł śledzenia użytkownika - start API lokalizowania

```
public void startLocationUpdates() {
    if (mLocationServiceCallback.checkGpsPermission()) {
        mRequestingLocationUpdates = true;

        LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, m
LocationRequest, LocationService.this);
    } else {
        Toast.makeText(getApplicationContext(),
R.string.no_location_permission, Toast.LENGTH_SHORT).show();
    }
}
```

Chwilę po stworzeniu serwisu wysyłane jest żądanie do „`FusedLocationApi`” o rozpoczęciu aktualizowania pozycji. Odbywa się to jeśli spełniony jest warunek posiadania przez aplikację uprawnień do używania systemu lokalizowania.

Listing 38 - Moduł śledzenia użytkownika - listener nowych lokalizacji

```
@Override
public void onLocationChanged(Location location) {
    mLocationsList.add(new Position(location));

    if (mCurrentLocation != null) {
        mLastLocation = mCurrentLocation;
        mCurrentLocation = location;
        mCurrentDistance = mCurrentLocation.distanceTo(mLastLocation);
        switch (mLocationServiceCallback.getWorkoutType()) {
            case CYCLING:
                sendKilometers();
                break;
            case RUNNING:
                sendMeters();
                break;
        }
    } else {
        mCurrentLocation = location;
        mLocationServiceCallback.animateLoading(false);
        mLocationServiceCallback.showBottomSheet();
        mLocationServiceCallback.passLocation(new
LatLng(location.getLatitude(), location.getLongitude()));
    }
}
```

Każda nowa otrzymana lokalizacji z API uruchamia powyższą metodę poprzez listener. Obliczana jest odległość jaką przebył użytkownik od poprzedniej lokalizacji, po czym

wywoływana jest odpowiednia metoda (w zależności od wybranego typu aktywności fizycznej – m/s dla biegania oraz km/h dla kolarstwa) odpowiadająca za komunikację z aktywnością. Jeśli jest to pierwsza otrzymana lokalizacja wyłączany jest ekran ładowania.

Listing 39 - Moduł śledzenia użytkownika - obliczanie danych w m/s

```
private void sendMeters() {
    long currentTime = System.currentTimeMillis();
    if (mCurrentDistance > MIN_DISTANCE && mCurrentDistance < MAX_DISTANCE)
    {
        mBurnedCalories += (((mCurrentLocation.getTime() -
mLastLocation.getTime()) / 1000) * 0.23);
        mTotalDistance += mCurrentDistance;
        mCurrentSpeed = mCurrentDistance / ((mCurrentLocation.getTime() -
mLastLocation.getTime()) / 1000);
        if (mCurrentSpeed > mTopSpeed) {
            mTopSpeed = mCurrentSpeed;
        }
        mSpeedsList.add(new Speed(currentTime, mCurrentSpeed));
        mDistancesList.add(new Distance(currentTime, mTotalDistance));
        mLocationServiceCallback.passDataMeters(new
LatLng(mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude()),
mTotalDistance, mCurrentSpeed);
    } else {
        mSpeedsList.add(new Speed(currentTime, 0.0));
        mDistancesList.add(new Distance(currentTime, mTotalDistance));
        mLocationServiceCallback.passDataMeters(new
LatLng(mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude()),
mTotalDistance, 0.0);
    }
}
```

Listing 40 - Moduł śledzenia użytkownika - obliczanie danych w km/h

```
private void sendKilometers() {
    long currentTime = System.currentTimeMillis();
    if (mCurrentDistance > MIN_DISTANCE && mCurrentDistance < MAX_DISTANCE)
    {
        mBurnedCalories += (((mCurrentLocation.getTime() -
mLastLocation.getTime()) / 1000) * 0.23);
        mCurrentDistance /= 1000d;
        mTotalDistance += mCurrentDistance;
        Logger.error("distance: " + mCurrentDistance);
        mCurrentSpeed = mCurrentDistance / ((mCurrentLocation.getTime() -
mLastLocation.getTime()) / 3600000d);
        Logger.error("time: " + (mCurrentLocation.getTime() -
mLastLocation.getTime()) / 3600000d);
        Logger.error("speed: " + mCurrentSpeed);
        if (mCurrentSpeed > mTopSpeed) {
            mTopSpeed = mCurrentSpeed;
        }
        mSpeedsList.add(new Speed(currentTime, mCurrentSpeed));
        mDistancesList.add(new Distance(currentTime, mTotalDistance));
        mLocationServiceCallback.passDataKilometers(new
LatLng(mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude()),
mTotalDistance, mCurrentSpeed);
    }
```

```

    } else {
        mSpeedsList.add(new Speed(currentTime, 0.0));
        mLocationServiceCallback.passDataKilometers(new
LatLng(mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude()),
mTotalDistance, 0.0);
    }
}

```

Powyższe metody liczą wszystkie potrzebne statystyki w odpowiednich jednostkach.

$$BurnedCalories = \frac{(CurrentLocation.getTime() - LastLocation.getTime())}{1000} * 0.23$$

$$TotalDistance = TotalDistance + CurrentDistance$$

$$CurrentSpeed = \frac{CurrentDistace * 1000}{CurrentLocation.getTime() - LastLocation.getTime()}$$

Metody getTime() zwracają czas w milisekundach. Wszystkie obliczone dane są przekazywane do aktywności i fragmentu w celu wyświetlenia ich na interfejsie użytkownika.

Listing 41 - Moduł śledzenia użytkownika - kończenie aktywności fizycznej

```

private void prepareWorkoutResult() {

    new AsyncTask<Void, Void, Void>() {

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            mFragmentManager.showBottomSheet(false);
            animateLoading(true);
        }

        @Override
        protected void onPostExecute(Void aVoid) {
            super.onPostExecute(aVoid);
            setTitle(R.string.workout_summary);

            changeFragment(WorkoutDetailsFragment.newInstance(mWorkout.getId()), true);
            animateLoading(false);
        }

        @Override
        protected Void doInBackground(Void... params) {
            stopLocationUpdates();
            ArrayList<Position> positionArrayList =
mService.getLocationsList();
            mWorkout.setFinishDate(System.currentTimeMillis());
            mWorkout.setElapsedTime(mFragmentManager.getElapsedTime());
            mWorkout.setTopSpeed(mService.getTopSpeed());
            mWorkout.setAverageSpeed(mService.getAverageSpeed());

            mWorkout.setAverageSpeedWithoutStops(mService.getAverageSpeedWithoutStops())

```

```

);
        mWorkout.setDistance(mService.getTotalDistance());

mWorkout.setStartAddress(getLocationAddress(positionArrayList.get(0).getLatitude(), positionArrayList.get(0).getLongitude()));

mWorkout.setFinishAddress(getLocationAddress(positionArrayList.get(positionArrayList.size() - 1).getLatitude(), positionArrayList.get(positionArrayList.size() - 1).getLongitude()));
        mWorkout.setBurnedCalories(mService.getBurnedCalories());
        saveWorkout();
        return null;
    }
    }.execute();
}

```

Po wciśnięciu przez użytkownika przycisku „Finish” wywoływana jest powyższa metoda. Uruchamia ona osobny wątek, ponieważ wykonywane w niej operacje zamrażają interfejs użytkownika, który działa na głównym wątku.

Metoda `onPreExecute()` definiuje co ma być wywołane przez stworzeniem wątku. Chowany jest tutaj Bottom Sheet oraz wyświetlany jest widok ładowania.

W `onPostExecute()`, czyli po zakończeniu pracy wątku, wyłączany jest ekran ładowania oraz zmieniany jest fragment na podsumowanie aktywności fizycznej.

Metoda `doInBackground()` zawiera w sobie wszystko co powinno się wykonywać na osobnym wątku. Ustawiane są pola obiektu `Workout` na podstawie danych pobieranych z serwisu oraz wywoływana jest metoda zapisująca do lokalnej bazy danych.

Listing 42 - Moduł śledzenia użytkownika - zapisywanie do lokalnej bazy danych

```

private void saveWorkout() {
    workoutDao = TrainerApplication.getDaoSession().getWorkoutDao();
    positionDao = TrainerApplication.getDaoSession().getPositionDao();
    SpeedDao speedDao = TrainerApplication.getDaoSession().getSpeedDao();
    distanceDao = TrainerApplication.getDaoSession().getDistanceDao();

    JoinWorkoutWithPositionsDao joinWorkoutWithPositionsDao =
TrainerApplication.getDaoSession().getJoinWorkoutWithPositionsDao();
    JoinWorkoutWithSpeedsDao joinWorkoutWithSpeedsDao =
TrainerApplication.getDaoSession().getJoinWorkoutWithSpeedsDao();
    JoinWorkoutWithDistancesDao joinWorkoutWithDistancesDao =
TrainerApplication.getDaoSession().getJoinWorkoutWithDistancesDao();

    long workoutId = workoutDao.insert(mWorkout);

    for (Position position : mService.getLocationsList()) {
        long id = positionDao.insert(position);
        joinWorkoutWithPositionsDao.insert(new
JoinWorkoutWithPositions(null, workoutId, id));
    }
}

```

```

    for (Distance distance : mService.getDistancesList()) {
        long id = distanceDao.insert(distance);
        joinWorkoutWithDistancesDao.insert(new
JoinWorkoutWithDistances(null, workoutId, id));
    }

    for (Speed speed : mService.getSpeedsList()) {
        long id = speedDao.insert(speed);
        joinWorkoutWithSpeedsDao.insert(new JoinWorkoutWithSpeeds(null,
workoutId, id));
    }
}

```

Zapisywanie aktywności do bazy danych odbywa się poprzez dodanie obiektu Workout oraz jego trzech kolekcji poprzez tabele asocjacyjne.

Listing 43 - Moduł śledzenia użytkownika - pobieranie adresu lokalizacji

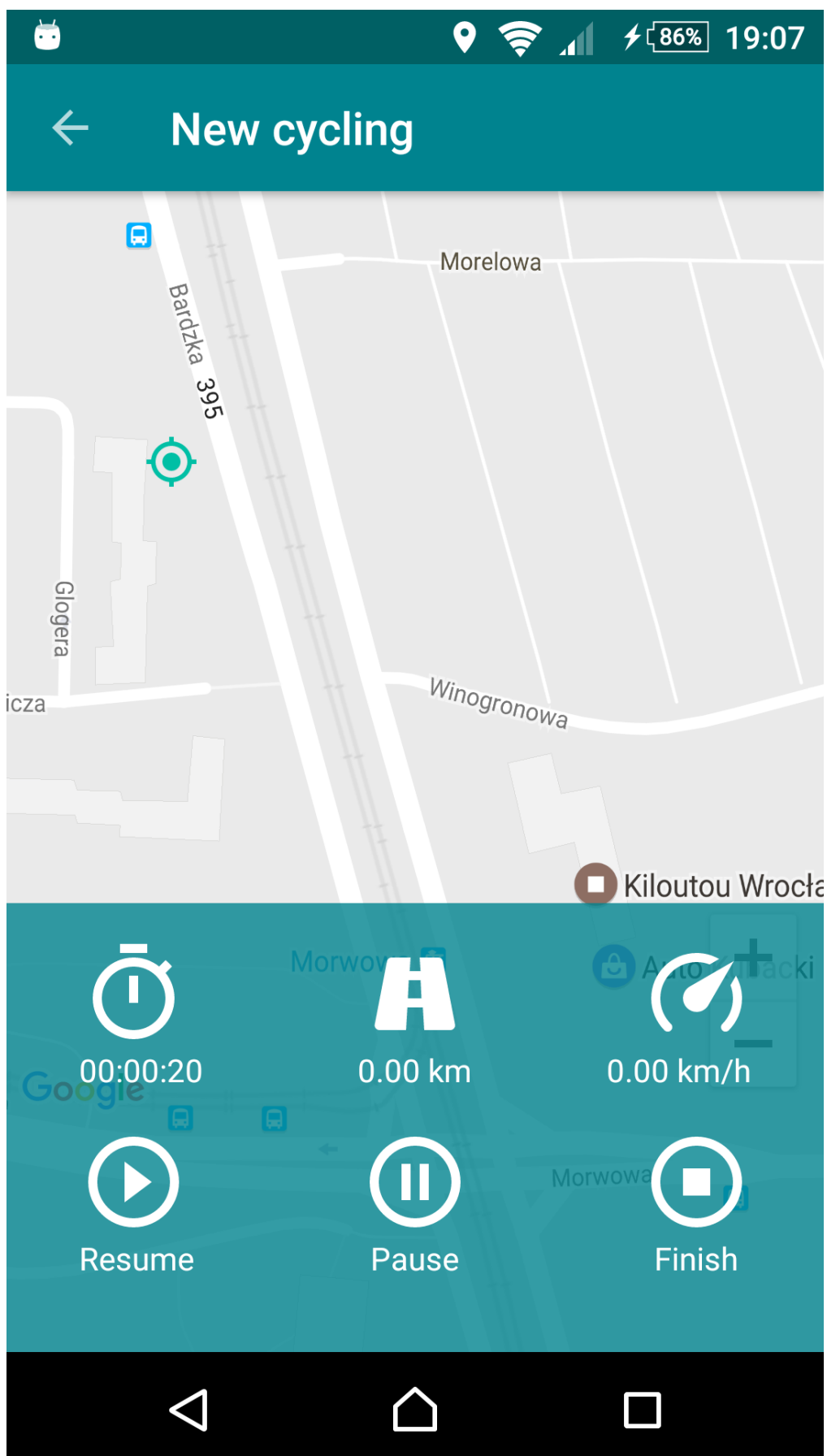
```

private String getLocationAddress(double latitude, double longitude) {
    Geocoder geocoder;
    List<Address> addresses;
    String address = "";
    geocoder = new Geocoder(this, Locale.getDefault());

    try {
        addresses = geocoder.getFromLocation(latitude, longitude, 1);
        if (addresses.get(0).getAddressLine(0) != null &&
!addresses.get(0).getAddressLine(0).isEmpty()) {
            address += addresses.get(0).getAddressLine(0) + " ";
        }
        if (addresses.get(0).getPostalCode() != null &&
!addresses.get(0).getPostalCode().isEmpty()) {
            address += addresses.get(0).getPostalCode() + " ";
        }
        if (addresses.get(0).getLocality() != null &&
!addresses.get(0).getLocality().isEmpty()) {
            address += addresses.get(0).getLocality() + " ";
        }
        if (addresses.get(0).getCountryName() != null &&
!addresses.get(0).getCountryName().isEmpty()) {
            address += addresses.get(0).getCountryName() + " ";
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return address;
}

```

Powyższa metoda jest używana przy ustalaniu adresu początkowej i końcowej lokalizacji. Używane jest tutaj API oparte o Google Maps, które dla danej szerokości i wysokości geograficznej zwraca dokładny adres.



Rys. 18 - Zrzut ekranu widoku mapy

Fragment ze szczegółami aktywności fizycznej może zostać wyświetlony po zakończeniu aktywnej aktywności lub z poziomu listy zakończonych.

Listing 44 - Moduł śledzenia użytkownika - pobieranie aktywności z bazy danych

```
private void getWorkoutFromDao(long id) {
    WorkoutDao workoutDao =
    TrainerApplication.getDaoSession().getWorkoutDao();
    QueryBuilder queryBuilder =
    workoutDao.queryBuilder().where(WorkoutDao.Properties.Id.eq(id));

    mWorkout = (Workout) queryBuilder.unique();
}
```

Razem z uruchomieniem fragmentu przekazywany jest klucz główny obiektu Workout, który jest używany do pobrania go z lokalnej bazy danych.

Listing 45 - Moduł śledzenia użytkownika - wyświetlenie szczegółów aktywności

```
if (mWorkout != null) {

    startTimeTextView.setText(DateFormat.getTimeInstance(DateFormat.SHORT).format(
        mWorkout.getStartDate().getTime()) +
        ", " +
        DateFormat.getDateInstance(DateFormat.LONG).format(mWorkout.getStartDate().
            getTime()));
    startLocationTextView.setText(mWorkout.getStartAddress());

    finishTimeTextView.setText(DateFormat.getTimeInstance(DateFormat.SHORT).format(
        mWorkout.getFinishDate().getTime()) +
        ", " +
        DateFormat.getDateInstance(DateFormat.LONG).format(mWorkout.getFinishDate().
            getTime()));
    finishLocationTextView.setText(mWorkout.getFinishAddress());
    elapsedTimeTextView.setText(mWorkout.getElapsedTime());

    burnedCaloriesTextView.setText(String.format(getResources().getString(R.string.
        burned_units_kcal), mWorkout.getBurnedCalories()));

    switch (mWorkout.getWorkoutType()) {
        case CYCLING:

    backgroundImageView.setImageResource(R.drawable.ic_directions_bike_gray_100
        dp);

    totalDistanceTextView.setText(String.format(getResources().getString(R.string.
        distance_units_km), mWorkout.getDistance()));

    topSpeedTextView.setText(String.format(getResources().getString(R.string.sp
        eed_units_kph), mWorkout.getTopSpeed()));

    averageSpeedTextView.setText(String.format(getResources().getString(R.string.
        speed_units_kph), mWorkout.getAverageSpeed()));

    averageSpeedWithoutTopsTextView.setText(String.format(getResources().getStr
        ing(R.string.speed_units_kph), mWorkout.getAverageSpeedWithoutStops()));
```



```

        break;
    case RUNNING:

        backgroundImageView.setImageResource(R.drawable.ic_directions_run_gray_100dp);

        totalDistanceTextView.setText(String.format(getResources().getString(R.string.distance_units_m), mWorkout.getDistance()));

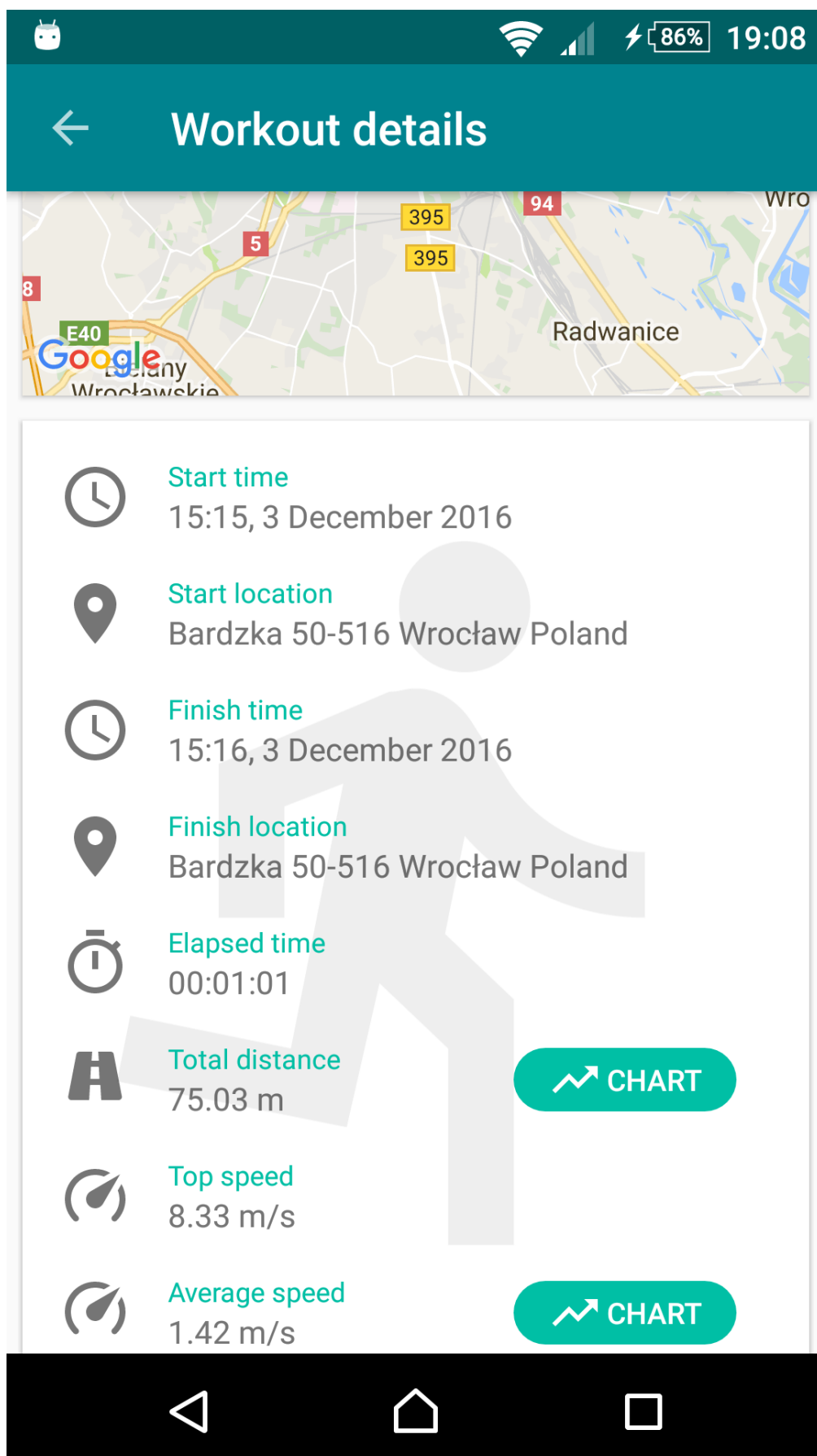
        topSpeedTextView.setText(String.format(getResources().getString(R.string.speed_units_mps), mWorkout.getTopSpeed()));

        averageSpeedTextView.setText(String.format(getResources().getString(R.string.speed_units_mps), mWorkout.getAverageSpeed()));

        averageSpeedWithoutTopsTextView.setText(String.format(getResources().getString(R.string.speed_units_mps), mWorkout.getAverageSpeedWithoutStops()));
        break;
    }
}

```

Pobrana aktywność fizyczna z bazy danych oraz jej dane wyświetlane są na liście zawartej w komponencie Card View.



Rys. 19 - Zrzut ekranu szczegółów treningu

Navigation Drawer prowadzi także do fragmentu wyświetlającego listę archiwalnych aktywności fizycznych. Każdy element posiada wyświetlony czas oraz przebyty dystans. Listener nasłuchujący kliknięcie na dany element przenosi użytkownika do poprzednio opisywanego widoku ze szczegółami aktywności fizycznej. Dostępna jest również ikona gwiazdki umożliwiająca dodanie danej aktywności jako ulubionej pozycji.

Floating Action Button znajdujący się u dołu ekranu pozwala na dodawanie oraz usuwanie elementu listy oraz jego odpowiednika w bazie danych. Akcja dodawania przenosi użytkownika do fragmentu rozpoczynania nowej aktywności fizycznej.

Listing 46 - Moduł śledzenia użytkownika - usuwanie aktywności fizycznej

```
public void deleteWorkout(final WorkoutItem item) {  
    // deleting workout  
    mWorkoutDao.deleteByKey(item.getObject().getId());  
    // deleting locations list  
    QueryBuilder queryBuilder =  
TrainerApplication.getDaoSession().getJoinWorkoutWithPositionsDao().queryBu  
ilder();  
  
queryBuilder.where(JoinWorkoutWithPositionsDao.Properties.WorkoutId.eq(item  
.getObject().getId()));  
    List<JoinWorkoutWithPositions> joinWorkoutWithPositionsList =  
queryBuilder.list();  
    for (JoinWorkoutWithPositions itemJoin :  
joinWorkoutWithPositionsList) {  
  
TrainerApplication.getDaoSession().getPositionDao().deleteByKey(itemJoin.ge  
tPositionId());  
  
TrainerApplication.getDaoSession().getJoinWorkoutWithPositionsDao().deleteB  
yKey(itemJoin.getId());  
    }  
    // deleting speeds list  
    queryBuilder =  
TrainerApplication.getDaoSession().getJoinWorkoutWithSpeedsDao().queryBuild  
er();  
  
queryBuilder.where(JoinWorkoutWithSpeedsDao.Properties.WorkoutId.eq(item.ge  
tObject().getId()));  
    List<JoinWorkoutWithSpeeds> joinWorkoutWithSpeedsList =  
queryBuilder.list();  
    for (JoinWorkoutWithSpeeds itemJoin : joinWorkoutWithSpeedsList) {  
  
TrainerApplication.getDaoSession().getSpeedDao().deleteByKey(itemJoin.getSp  
eedId());  
  
TrainerApplication.getDaoSession().getJoinWorkoutWithSpeedsDao().deleteByKe  
y(itemJoin.getId());  
    }  
    // deleting distances list  
    queryBuilder =  
TrainerApplication.getDaoSession().getJoinWorkoutWithDistancesDao().queryBu  
ilder();
```

```

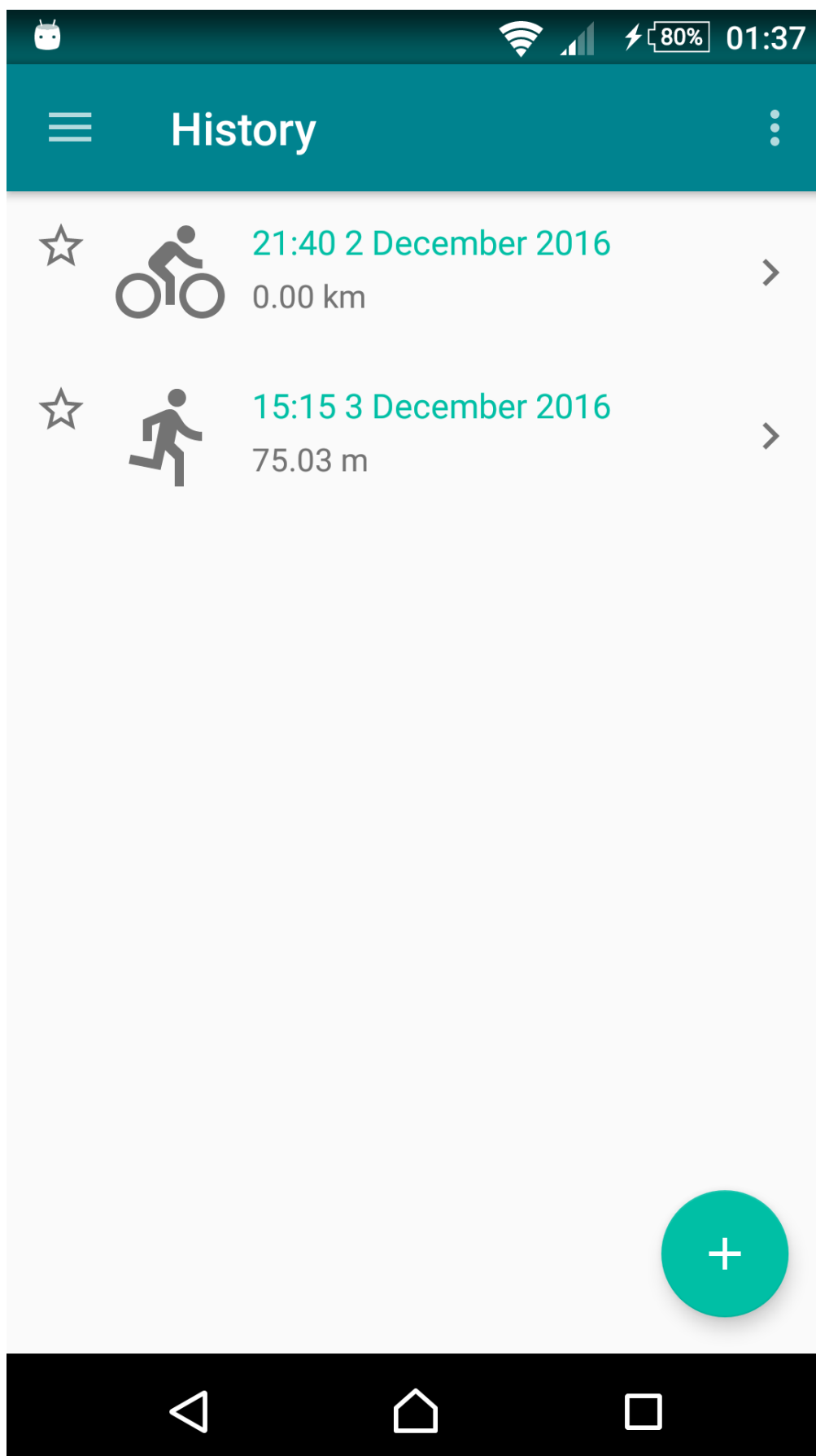
queryBuilder.where(JoinWorkoutWithDistancesDao.Properties.WorkoutId.eq(item
.getObject().getId()));
    List<JoinWorkoutWithDistances> joinWorkoutWithDistancesList =
queryBuilder.list();
    for (JoinWorkoutWithDistances itemJoin :
joinWorkoutWithDistancesList) {

TrainerApplication.getDaoSession().getDistanceDao().deleteByKey(itemJoin.ge
tDistanceId());

TrainerApplication.getDaoSession().getJoinWorkoutWithDistancesDao().deleteB
yKey(itemJoin.getId());
    }

    int index = mWorkoutsList.indexOf(item);
    if (index >= 0) {
        mWorkoutsList.remove(index);
        mAdapter.notifyItemRemoved(index);
    }
    showAddButton();
}

```



Rys. 20 - Zrzut ekranu historii aktywności

## 5.2.6. Moduł pomocniczych kalkulatorów

Pomocnicze kalkulatory dostępne w aplikacji pozwalają użytkownikowi na obliczenie jego własnego indeksu BMI i BFP korzystając ze wzorów:

$$BMI = \frac{waga\ (kg)}{wysokość^2\ (cm)}$$

$$BMI = \frac{waga\ (lb) * 703}{(wysokość\ (in) * 12)^2}$$

$$BFP = 1.2 * BMI + 0.23 * wiek\ (lata) - 5.4 - 10.8$$

Listing 47 - Moduł pomocniczych kalkulatorów - adapter View Pager

```
public static class BmiAdapter extends FragmentPagerAdapter {

    private BmiResultFragment mRangesFragment;

    public BmiAdapter(FragmentManager fragmentManager) {
        super(fragmentManager);
        mRangesFragment = mRangesFragment.newInstance();
    }

    @Override
    public int getCount() {
        return NUM_ITEMS;
    }

    @Override
    public Fragment getItem(int position) {
        switch (position) {
            case 0:
                return BmiCalculatorFragment.newInstance();
            case 1:
                return mRangesFragment;
            default:
                return null;
        }
    }

    @Override
    public CharSequence getPageTitle(int position) {
        switch (position) {
            case 0:
                return "Calculate";
            case 1:
                return "Result";
        }
        return null;
    }
}
```

Widok składa się z fragmentu zawierającego View Pager z dwoma zagnieżdżonymi fragmentami – do wpisywania danych oraz do wyświetlania wyniku. Po wypełnieniu przez użytkownika wszystkich wymaganych pól obliczany jest rezultat. Zagnieżdżony fragment poprzez listener wywołuje we fragmencie-rodzicu metodę przełączenia na drugi zagnieżdżony fragment z wyświetlonym rezultatem.

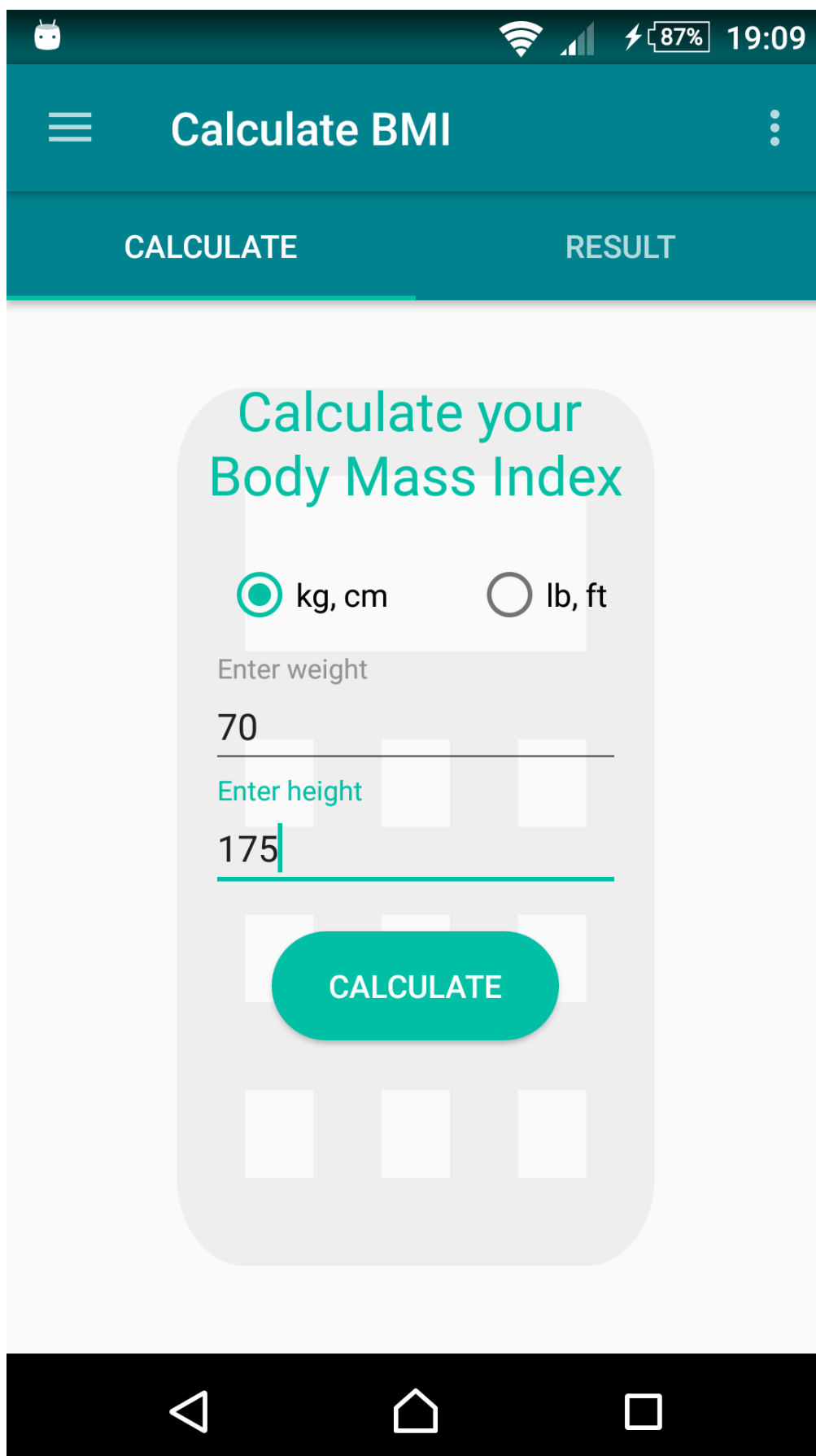
Listing 48 - Moduł pomocniczych kalkulatorów - obliczanie BMI

```
public void calculateBmi(boolean metric, boolean male, String weight,
String height) {
    double result;
    if (metric) {
        result = metricBmi(Double.parseDouble(weight),
(Double.parseDouble(height) / 100));
    } else {
        result = imperialBmi(Double.parseDouble(weight),
(Double.parseDouble(height)) * 12);
    }
    setResult(result);
}

private void setResult(double result) {
    mParentFragment.switchFragment(1, round(result, 2));
}
```

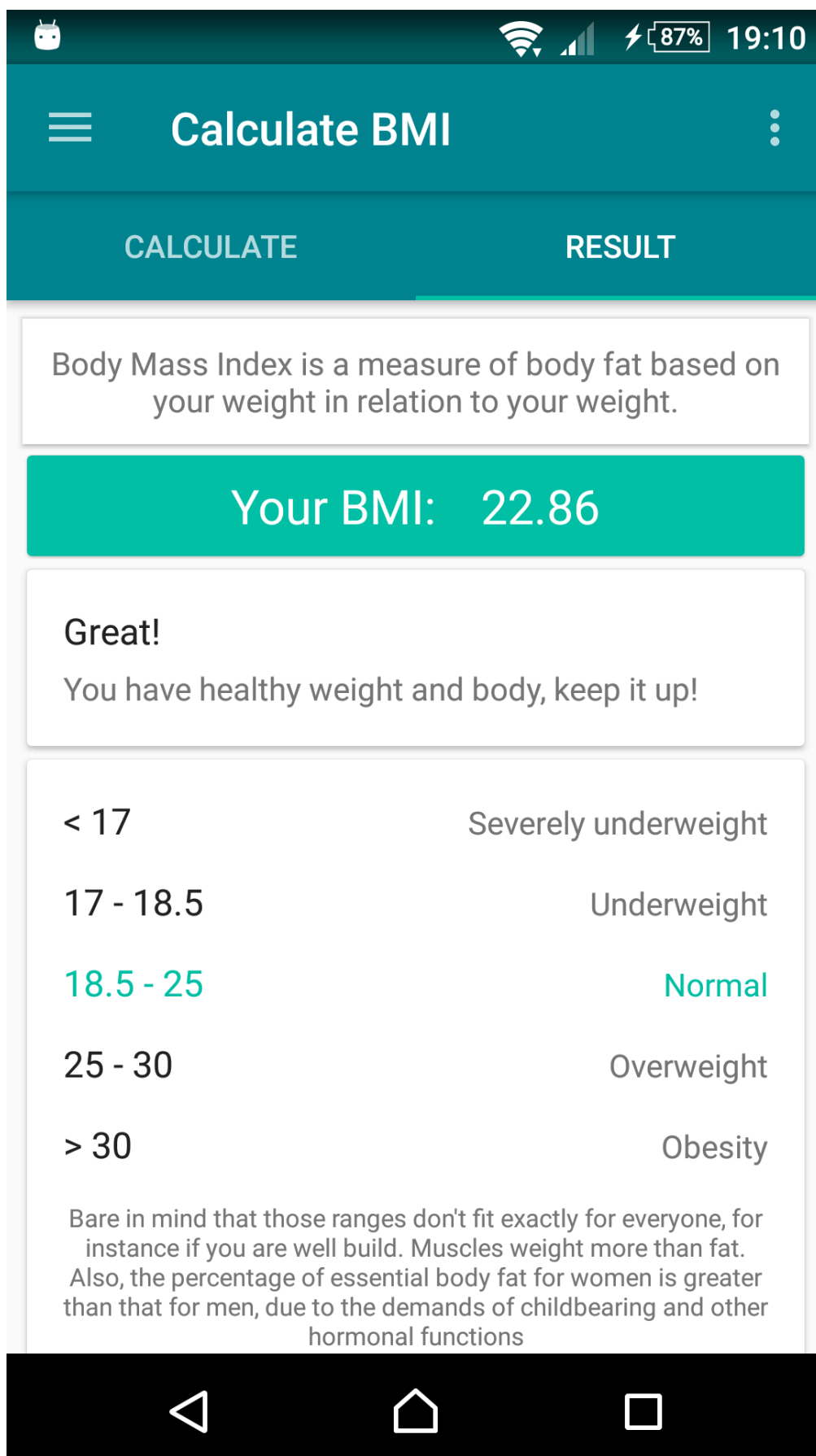
Listing 49 - Moduł pomocniczych kalkulatorów - przełączanie zagnieżdżonych fragmentów

```
public void switchFragment(int page, double result) {
    BmiResultFragment fragment = (BmiResultFragment)
mAdapterViewPager.getItem(1);
    fragment.setResult(result);
    mViewPager.setCurrentItem(page, true);
}
```







Rys. 21 - Zrzut ekranu kalkulatora BMI






Rys. 22 - Zrzut ekranu rezultatu BMI






87%

19:09



Calculate BFP



CALCULATE

RESULT

Calculate your  
Body Fat Percentage

☒ kg, cm

☐ lb, ft

☒ Male

☐ Female

Enter weight

70




Enter height

175

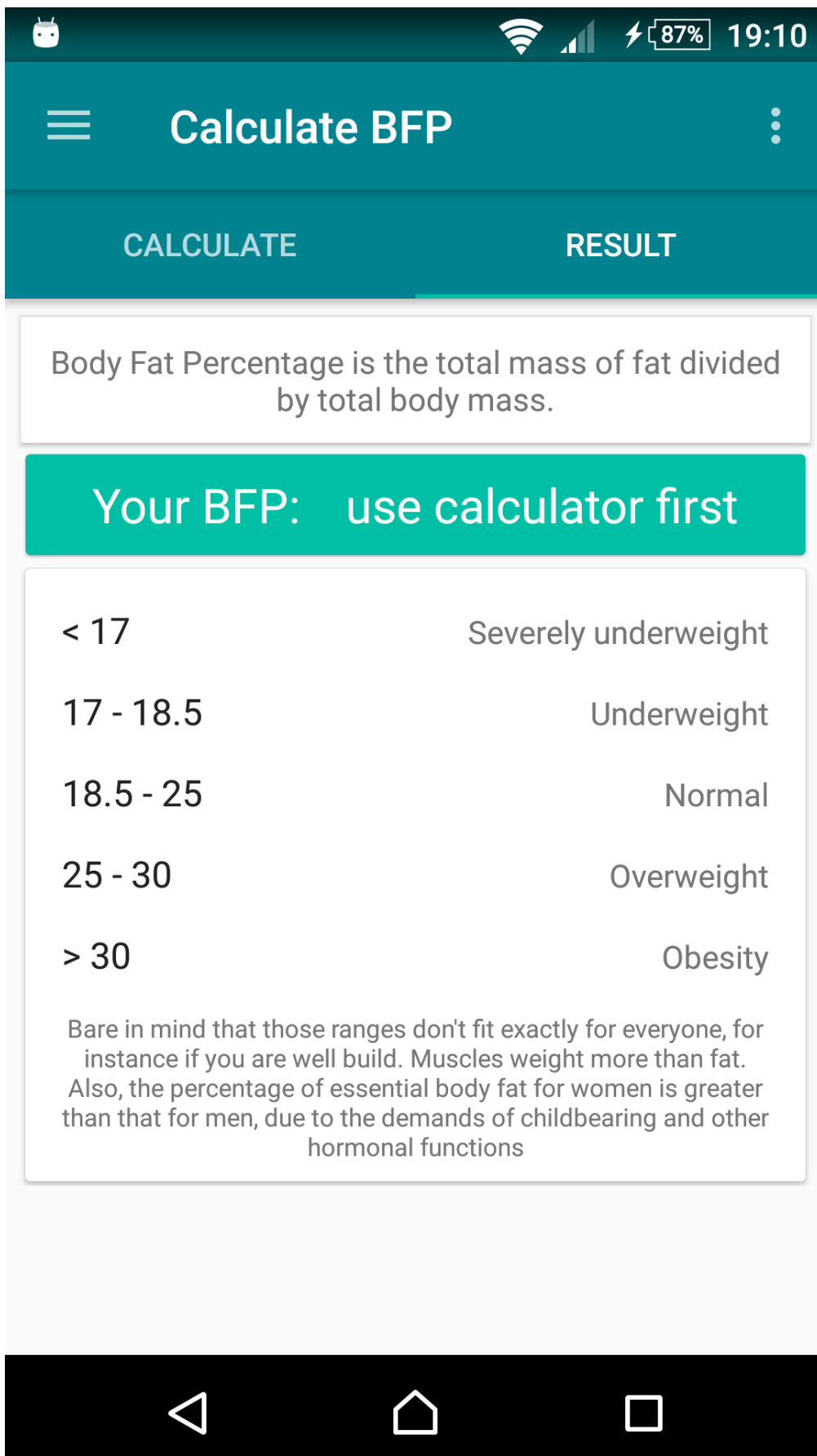
Enter age

22

CALCULATE



Rys. 23 - Zrzut ekranu kalkulatora BFP



## 5.3. Instalacja i konfigurowanie systemu

Do uruchomienia aplikacji wymagane jest urządzenie z systemem operacyjnym Android w wersji 4.4 KitKat lub nowszej. W przypadku posiadania gotowej, zbudowanej aplikacji w formacie archiwum .apk, wystarczy skopiować ją na urządzenie i zainstalować korzystając z domyślnego menedżera pakietów.

Jeśli nie posiadamy pliku .apk niezbędne będzie własnoręczne zbudowanie aplikacji. Aby to wykonać potrzebny jest komputer z zainstalowanymi komponentami:

- Java Development Kit, wersja 1.7 lub nowsza (z ustawioną zmienną środowiskową JAVA\_HOME do katalogu /bin z zainstalowanym JDK).
- Git
- Android Studio (opcjonalne).
- Android Debug Bridge (jeśli nie zainstalowaliśmy Android Studio).

Do instalacji aplikacji poprzez ADB wymagane jest przygotowanie urządzenia, które podłączamy do komputera kablem USB. Następnie wchodzimy w ustawienia oraz w „Opcje programistyczne”, po czym zaznaczamy „Debugowanie USB” jako włączone. Jeśli kategoria „Opcje programistyczne” nie jest widoczna należy najpierw wejść w kategorię „Informacje o telefonie” i przycisnąć siedmiokrotnie „Numer kompilacji”.

W celu zbudowania aplikacji potrzebny jest projekt, w którym aplikacja została zaimplementowana. Pobranie wykonujemy poprzez sklonowanie repozytorium gita. Aby to zrobić wystarczy z poziomu wiersza poleceń podać komendę „git clone [https://github.com/kolendo/pocket\\_trainer.git](https://github.com/kolendo/pocket_trainer.git)”.

Po sklonowaniu projektu należy zbudować i zainstalować aplikację używając jednego z dwóch sposobów:

- Przy użyciu Android Studio:  
Importujemy pobrany projekt wybierając opcję File -> New -> Import Project, a następnie wybieramy plik „settings.gradle” znajdujący się w katalogu projektu.

Po automatycznej synchronizacji Gradle'a, należy wybrać opcję „Run” (ikona zielonego trójkąta), a następnie w nowym oknie wybrać podłączone urządzenie.

- Bez użycia Android Studio:

Otwieramy wiersz poleceń, po czym przechodzimy do katalogu z pobranym projektem. Następnie należy wydać polecenie „gradlew.bat assembleDebug”, jeśli korzystamy z Windowsa lub „./gradlew assembleDebug”, jeśli używamy Unixa. Po zakończeniu skryptu przechodzimy do katalogu /app/build/outputs/apk znajdującego się w projekcie i wpisujemy komendę „adb install -r PocketTrainer-v0.0.1dev-debug.apk” (lub podobnym plikiem .apk z aktualną wersją).

## 5.4. Wykorzystane narzędzia i biblioteki

Aplikacja tworzona jest w Android Studio – bezpłatnym środowisku dedykowanym dla tej platformy. Opiera się ono na oprogramowaniu IntelliJ, którego licencja została zakupiona przez Google. Zewnętrznymi API, używanymi w projekcie są Google Play Services umożliwiające lokalizowanie GPS oraz framework greenDAO posiadający rozbudowane API z mapowaniem obiektowo-relacyjnym. Wszystkie zależności projektu i biblioteki pochodzą od Google, oprócz jednej odpowiedzialnej za animację ładowania – Android-SpinKit, która jest oparta na standardowej licencji Apache.

W projekcie wykorzystany jest także system kontroli wersji Git z publicznym repozytorium na GitHubie.

## 5.5. Testy aplikacji

Testowanie aplikacji zostało wykonane poprzez testy funkcjonalne (testy czarnej skrzynki). Kilka osób dostało wgrane oprogramowanie na swoje telefony, po czym rozpoczęli używanie z perspektywy przeciętnego użytkownika, bez znajomości budowy działania aplikacji. Testy tego typu posiadają większą szansę na wykrycie błędów, lecz z powodu braku programistycznej wiedzy, testerzy nie są w stanie przekazać programiście co było wynikiem błędu. Projekt w niniejszej pracy posiadał kilka nieznaczących, nie mających wpływu na działanie całego systemu błędów. Przykładem jest niepoprawna walidacja wpisywanych

danych do kalkulatora lub brak efektu w interfejsie graficznym po przejściu w inną pozycję wysuwanego menu.

## 6. Podsumowanie

W ramach niniejszej pracy udało się zrealizować wszystkie założone cele. Podczas projektu zostały wykonane następujące zadania:

- Moduł planowania własnych treningów.
- Moduł śledzenia ruchu i aktywności użytkownika.
- Dodatki: kalkulator liczący indeks BMI oraz ilość tłuszczu w ciele BFP.

Przeprowadzona początkowa analiza wymagań pozwoliła zaprojektować w pełni działający i sprawny system, z profesjonalnym designem oraz bogatymi funkcjonalnościami.

Najbardziej problematycznym i wymagającym czasu fragmentem pracy był moduł śledzenia aktywności fizycznej. Wymagał on nie tylko umiejętności programowania w Androidzie, ale także zaprojektowania własnego minisystemu odpowiadającego za liczenie statystyk oraz ruchu użytkownika na podstawie długości i szerokości geograficznych, otrzymywanych co pewien interwał czasu. Rozwiązanie zapewni mi wiedzę o projektowaniu podobnych systemów w przyszłości.

Aplikacja pomimo posiadania zaimplementowanych założeń projektowych i funkcjonalności wystarczających do codziennego użytku podczas uprawiania amatorskiego sportu, może zostać w przyszłości zmieniona pod kątem nowych możliwości. Jednym z pomysłów jest zaprojektowanie REST API umożliwiającego implementację kont użytkowników na serwerze. Pozwoli to każdemu użytkownikowi na posiadanie własnego konta, dzięki czemu zyskuje możliwość synchronizacji własnych danych i ustawień na każdym urządzeniu, na którym się zaloguje. Innym aspektem posiadania kont użytkowników oraz aplikacji opartej na lokalizacji GPS z widokiem mapy jest funkcjonalność społecznościowa. Użytkownicy, którzy wyrażą chęć udostępniania własnej lokalizacji w ustawieniach, będą mogli zobaczyć siebie oraz innych na ekranie mapy. Możliwa jest też implementacja systemu znajomych lub umawianie się na wspólne treningi z nieznanymi

użytkownikami. W ten sposób jesteśmy w stanie połączyć uprawianie sportu oraz nawiązywanie nowych znajomości z nowymi technologiami.

# Bibliografia

- [1] IDC Research, Inc. Smartphone OS Market Share, Sierpień 2016.  
<https://www.idc.com/prodserv/smartphone-os-market-share.jsp> [dostęp dnia 04 grudnia 2016].
- [2] Gregory Twohig. Mobile Software Statistics 2015, Czerwiec 2015.  
<https://mobiforge.com/research-analysis/mobile-software-statistics-2015> [dostęp dnia 04 grudnia 2016].
- [3] Android Developers. Platform Versions, Listopad 2016.  
<https://developer.android.com/about/dashboards/index.html> [dostęp dnia 05 grudnia 2016].
- [4] Iwona Pożniak-Koszałka. Relacyjne bazy danych w środowisku Sybase, 2004.  
[http://www.dbc.wroc.pl/Content/1182/pozniak\\_relacyjne\\_bazy.pdf](http://www.dbc.wroc.pl/Content/1182/pozniak_relacyjne_bazy.pdf) Dolnośląska Biblioteka Cyfrowa [dostęp dnia 05 grudnia 2016].



# Dodatek A:

Płyta CD z projektem oraz zbudowaną aplikacją w formacie .apk.