WOJSKOWA AKADEMIA TECHNICZNA
im. Jarosława Dąbrowskiego
Wydział ELEKTRONIKI

PRACA DYPLOMOWA

Aplikacja mobilna trenera personalnego

(temat pracy dyplomowej)

Inż. Łukasz KOPCZYŃSKI, s. Michała

(stopień wojskowy, tytuł zawodowy, imiona i nazwisko, imię ojca dyplomanta)

ELEKTRONIKA I TELEKOMUNIKACJA
(kierunek studiów)

Systemy teleinformatyczne
(specjalność)

STACJONARNE, STUDIA DRUGIEGO STOPNIA - MAGISTERSKIE
(forma i rodzaj studiów))*

dr inż. Andrzej Poniecki
(stopień wojskowy, tytuł i stopień naukowy, imię i nazwisko promotora pracy dyplomowej)

WARSZAWA 2023

Spis treści

1. Wprowadzenie

Aplikacje mobilne z każdym rokiem zdobywają coraz większą popularność i nie jednokrotnie użytkownicy sięgają po nie częściej niż ich przeglądarkowe odpowiedniki ze względu na poręczność urządzeń typu smartphone oraz stabilniejszą i bogatszą funkcjonalność. Dzisiejsze telefony mają nieporównywalnie większe możliwości, niż komputery stacjonarne produkowane 10 lat temu przez co ich rola w światowym rynku urządzeń IT, do których zaliczamy między innymi : komputery, laptopy, tablety oraz czytniki e-book. Możliwości prezentowane przez dzisiejsze smartphony często wykraczają poza ludzkie potrzeby z przed 15 lat.

Jedną z takich potrzeb jest bycie w dobrej kondycji fizycznej i psychicznej co pozwala na utrzymanie zdrowego trybu życia, który z roku na rok zyskuje na popularności dzięki czemu coraz więcej ludzi decyduje się na mniejszą bądź większą aktywność. Bieganie, jazda na rowerze czy zwyczajny spacer są dobrym punktem startowym jednak osoby myślące długoterminowo, które chcą włączyć sport w swoje życie skłaniają się ku bardziej wymagającym sportom, do których bez wątpienia zalicza się wysiłek siłowy.

Osobom początkującym bardzo ciężko jest zacząć uprawianie sportu siłowego ze względu na wysoki próg wejścia, jeżeli chodzi o wiedzę potrzebną do zdobycia już na samym początku. Wiedza, którą możemy zdobyć w Internecie jest często niesprawdzona, a konsultacja ze specjalistą wymaga od nas znacznych środków fizycznych.

Aplikacja mobilna trenera personalnego powstała z myślą o osobach, które zaczynając swoją przygodę na siłowni chcą zdobyć kompleksową wiedzę, dzięki czemu start będzie prostszy, bezpieczniejszy, a efekty szybciej zauważalne.

2. Wybór środowiska projektowego

2.1. Urządzenie typu smartphone

Smartphone to przenośne urządzenie multimedialne, które łączy w sobie dwie funkcje – telefonu komórkowego oraz przenośnego komputera. Pierwszy egzemplarz smartphone to dzieło firmy IBM o nazwie Simon, który zadebiutował w 1992 roku. Model ten posiadał oprócz funkcji telefonu również kalendarz czy kalkulator. Od tamtej pory rynek smartphone rozwinął się do takiego stopnia, że telefony potrafią niekiedy przewyższyć swoimi komponentami komputery stacjonarne. Ich udział w rynku rośnie z roku na rok i wiele wskazuje na to, że będą one stanowiły poważną konkurencję dla pozostałych komputerów ze względu na możliwości, dzięki zaimplementowanym komponentom oraz swojej poręczności.

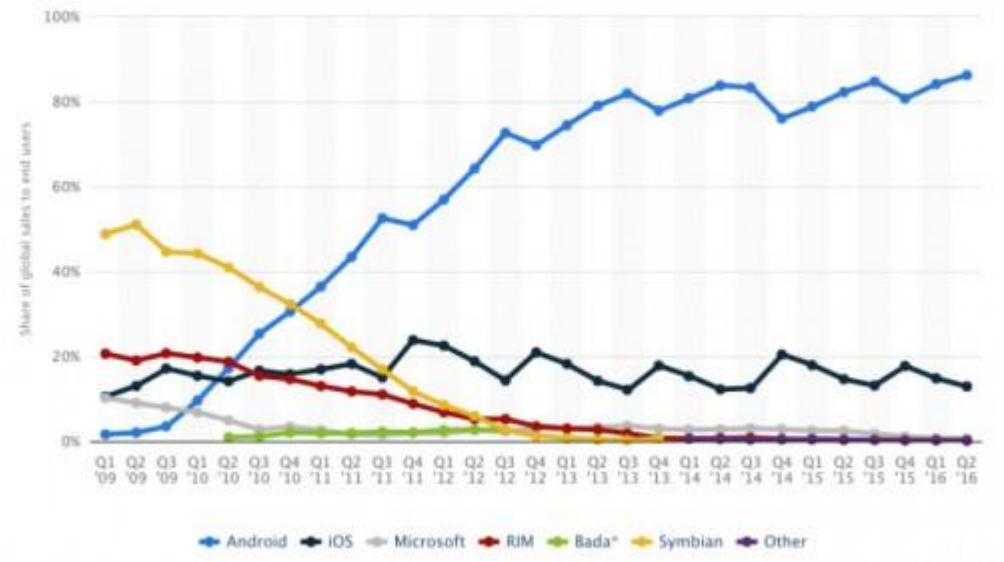
Rys 2.1 Sprzedaż smartphonów w latach 2019-2022. [1]

Analizując wykres przedstawiony na rysunku 2.1 można wywnioskować, że pozycja rynku smartphone jest nieustabilizowana. Należy mieć jednak na uwadze, że był to czas pandemii dlatego spadek między trzecim kwartałem 2019 roku oraz pierwszym kwartałem 2020 roku jest tak znaczący. Okres przejściowy w pandemii to wzrost sprzedaży o blisko 30 procent po czym następuje ponowny spadek związany z

niepewnością rynkową. Mimo, że trzeci kwartał 2022 roku zakończony jest z bilansem ujemnej sprzedaży można spodziewać się powrotu do wartości dodatnich wzrostu sprzedaży w związku z rosnącym trendem przedstawionej powyżej funkcji.

2.2. Wybór systemu operacyjnego

Rynek systemów operacyjnych został niemalże w pełni przejęty przed dwa najpopularniejsze rozwiązania – system Android oraz IOS. System operacyjny Android to system z jądrem Linux rozwijany nieprzerwanie od 2005 roku, kiedy to firma Google zakupiła Android Inc. Od tamtej pory amerykański gigant technologiczny nieprzerwanie rozwija ten system przejmując znaczny udział w rynku. System operacyjny IOS w wersji oryginalnej jest przeznaczony wyłącznie dla urządzeń mobilnych firmy Apple Inc. Stale akutalizowany system operacyjny, który swoją premierę miał ponad 15 lat temu bazuje na systemie operacyjnym MAC OS X 10.5 i tym samym na Darwinie, czyli na uniksowej podstawie systemu operacyjnego macOS firmy Apple.



Rys 2.2 Udział systemów operacyjnych przeznaczonych na platformy mobilne w latach 2009-2016 [2].

Analizując dane z rysunku drugiego przedstawiające znaczenie różnych systemów operacyjnych na rynku w latach 2009-2016 zauważyć można, że już w 2013 roku rynek systemów dla urządzeń mobilnych zdominowały dwa systemy operacyjne, o których zostało wspomniane w poprzednim akapicie. Przeważający udział systemu operacyjnego Android nad systemem iOS wynika głównie z faktu, że system android dostępny jest na smartfonach różnych producentów smartphonów, którzy często

konkurują ze sobą na rynku o klienta. Oferują różne rozwiązania w ramach komponentów oraz wyglądu natomiast system operacyjny jest identyczny. Przykładem takiego rozwiązania jest firma Samsung oraz Xiaomi. System operacyjny iOS jest dostępny tylko i wyłącznie dla urządzeń mobilnych produkowanych przez firmę Apple. Oznacza to, że żadna inna firma nie ma możliwości kupna ich systemu operacyjnego do zaimplementowania w swoim urządzeniu mobilnym. Dzięki takiemu rozwiązaniu często użytkownicy systemu iOS uważani są za zamkniętą grupę użytkowników w związku z faktem, że sam system jest ściśle zamknięty i firma Apple nie udostępnia kodu źródłowego oprogramowania.

Dyskusja nad tym, który system operacyjny jest „lepszy" możliwie nigdy nie zostanie doprowadzona do konsensusu. Oba systemy prezentują unikalne i innowacyjne rozwiązania w zakresach bezpieczeństwa, komfortu użytkowania oraz jakości swoich produktów. Faktem jest jednak, że przeważający udział w rynku ma zdecydowanie system operacyjny Android. Aplikacje tworzone dla tego systemu operacyjnego trafiają -w zależności od aktualnej sytuacji na rynku systemów dla aplikacji mobilnych – do co najmniej 75% użytkowników zgodnie z wykresem przedstawionym na rys.2. analizując lata 2013-2016. Otwiera to zatem zdecydowanie większe możliwości dotarcia do większej grupy użytkowników, niż system iOS co można uznać za przewagę tego systemu.

2.3. System operacyjny Android

System Android jest mobilnym systemem operacyjnym, który oparty jest na zmodyfikowanej wersji jądra Linuxa i innego oprogramowania typu „open source" oznaczającego możliwość dostępu i modyfikowania kodu źródłowego. System ten przeznaczony jest głównie dla urządzeń mobilnych z ekranem dotykowym, takich jak tablety czy smartphony. Android rozwijany jest przez konsorcjum programistów zwane jako Open Handset Alliance.

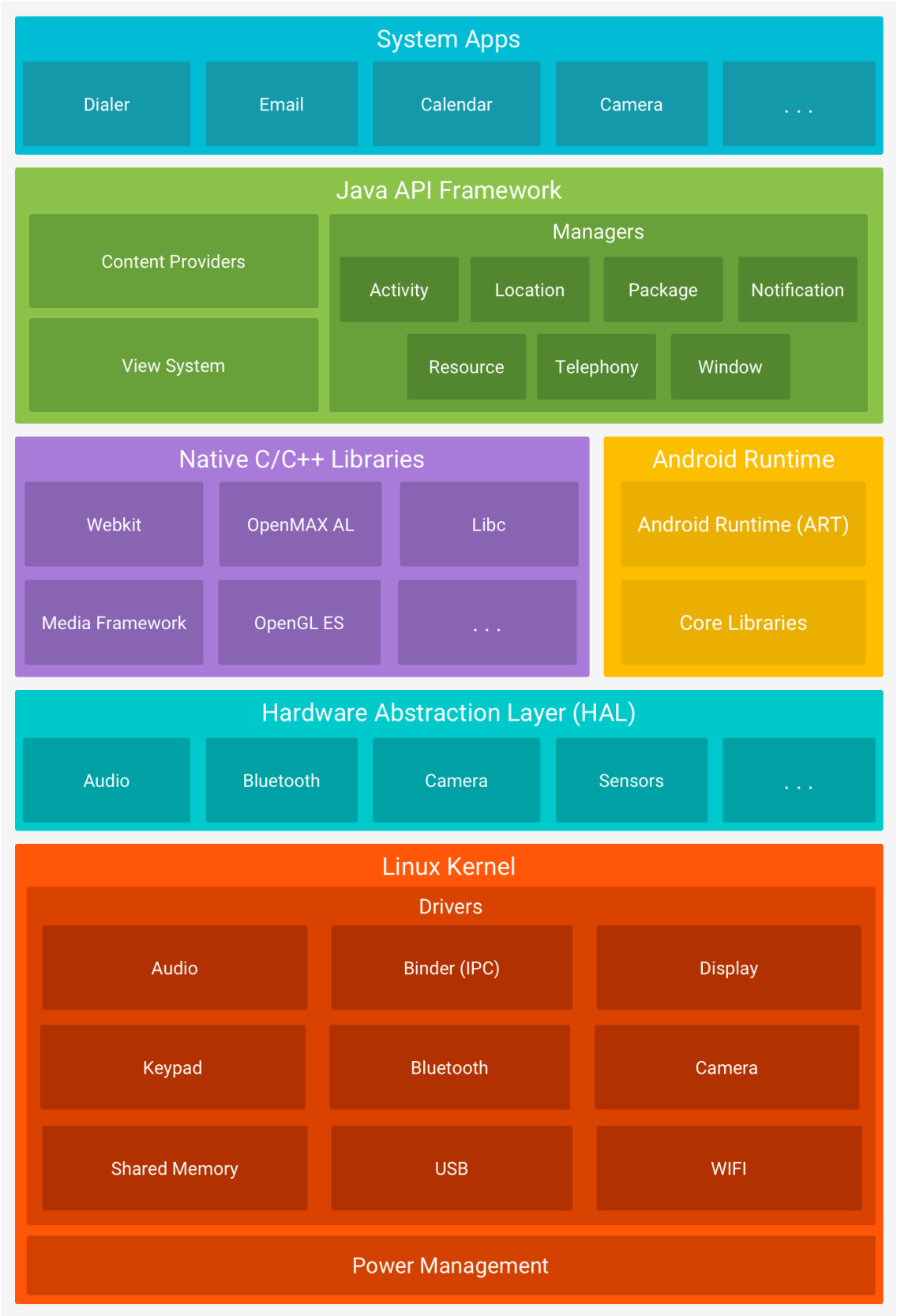
Open Handset Alliance to stowarzyszenie 84 firm w celu opracowania otwartych standardów dla urządzeń mobilnych. W śród firm członkowskich są takie korporacje jak : HTC, Sony, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Google, Samsung Electronics, LG Electronics, T-Mobile, Nvidia oraz Wind River Systems.

Rys 2.3. Logo systemu operacyjnego Android [3]

Zasadniczo system operacyjny znany jest jako Android Open Source Projekt. Jest to darmowe oprogramowanie, które każdy może modyfikować, używać i zmieniać w dowolny sposób, licencjonowanym głównie w ramach licencji Apache. Większość urządzeń działa jednak na zastrzeżonej wersji Androida opracowanej przez Google, która jest dostarczana z dodatkowym preinstalowanym oprogramowaniem, którego kod jest zamknięty, czyli niedostępnym dla każdego użytkownika. Takie rozwiązanie zostało zastosowane szczególnie w aplikacjach z rodziny Google Mobile Services, w skład której wchodzi między innymi Google Chrome oraz Google Play czyli odpowiednio platforma przeglądarkowa oraz serwis służący do dystrybucji cyfrowej.

Kod źródłowy Androida został wykorzystany do opracowania wariantów tego systemu operacyjnego na wielu innych urządzeniach elektronicznych, takich jak konsole do gier, aparaty cyfrowe, przenośne odtwarzacze multimedialne, czy też komputery PC. Należy zaznaczyć że każdy z wyżej wymienionych urządzeń ma indywidualnie wyspecjalizowany interfejs użytkownika. [4]

Na rysunku 4 została przedstawiona architektura systemu operacyjnego Android. Stos oprogramowania składa się z jądra Linuxa, warstwy abstrakcji sprzętu (ang. Hardware Abstraction Layer), środowiska wykonawczego, natywnych bibliotek C/C++, struktury interfejsu API Java oraz aplikacji systemowych. Wszystkie te aspekty zostały omówione pod rysunkiem 2.4.



Rys 2.4. Stos oprogramowania Androida [5]

Jądro Linuxa jest podstawą platformy Android. Korzystanie z jądra systemu Linux umożliwia systemowi Android korzystanie z kluczowych funkcji bezpieczeństwa i daje możliwość producentom urządzeń dla opracowania potrzebnych im sterowników. Środowisko wykonawcze opiera się na jądrze systemu Linux w zakresie podstawowych funkcji takich jak wątkowanie czy też zarządzanie pamięcią niskiego poziomu.

Warstwa abstrakcji sprzętu (HAL) zapewnia standardowe interfejsy, które udostępniają możliwości sprzętowe wyższemu poziomowi platformy Java API. HAL składa się z wielu modułów bibliotecznych, z których każdy implementuje interfejs dla określonego typu komponentu sprzętowego. Podczas wysyłania wywołania w celu uzyskania dostępu do sprzętu urządzenia przez interfejs API, system ładuje moduł biblioteki dla komponentu sprzętowego.

Środowisko wykonawcze Androida zostało stworzone w celu uruchamiania wielu aplikacji na urządzeniach o małej ilości pamięci, wykonując do tego maszynę wirtualną Dalvik zaprojektowaną i zoptymalizowaną pod kątem minimalnego zużycia pamięci.

Natywne biblioteki C/C++ obsługują wiele podstawowych komponentów i usług systemu Android, takich jak ART. I HAL. Platforma Android udostępnia interfejsy API platformy Java, które udostępniają aplikacjom funkcjonalność niektórych z tym natywnych bibliotek.

Cały zestaw funkcji systemu operacyjnego Android jest dostępny za pośrednictwem interfejsów API napisanych w języku Java. Te interfejsy tworzą elementy potrzebne do tworzenia aplikacji na system Android, upraszczając ponowne wykorzystanie podstawowych modułowych komponentów systemów i usług.

Android dostarczany jest z zestawem podstawowych aplikacji do obsługi poczty, wiadomości SMS, kalendarza, przeglądarki internetowej. Nie ma jednak odgórnego ustalenia co do korzystania z konkretnego rozwiązania dla danej funkcji systemowej. Użytkownik może posiadać swoją aplikację np. kalendarza czy przeglądarki z której będzie korzystał i to ona będzie aplikacją domyślną.

2.4. Środowisko programistyczne Android Studio

Rys 2.5. Logo środowiska Android Studio [6]

Środowisko Android Studio, którego logo zostało przedstawione na rysunku 2.5, to oficjalne zintegrowane środowisko programistyczne (IDE – ang. Integrated Development Enviroment) dla systemu operacyjnego Google Android, zbudowane na oprogramowaniu IntelliJ IDEA firmy JetBrains i zaprojektowane specjalnie w celu tworzenia oprogramowania na platformę mobilną Android Studio. Jest to zamiennik dla E-ADT (Elicpse Android Development Tools) jako podstawowe narzędzie do tworzenia natywnych aplikacji na Androida dostępne do pobrania w systemach operacyjnych Windows, macOS oraz Linux.

Językiem programowania, preferowanym przez firmę Google jest Kotlin, który to 7 maja 2019 roku zastąpił Javę. Nadal możliwe jest tworzenie aplikacji w Javie oraz C++, jednak kierunek amerykańskiego giganta technologicznego jest jednoznaczny i prowadzi do rozwijania swojego własnego języka w opozycji do Javy.

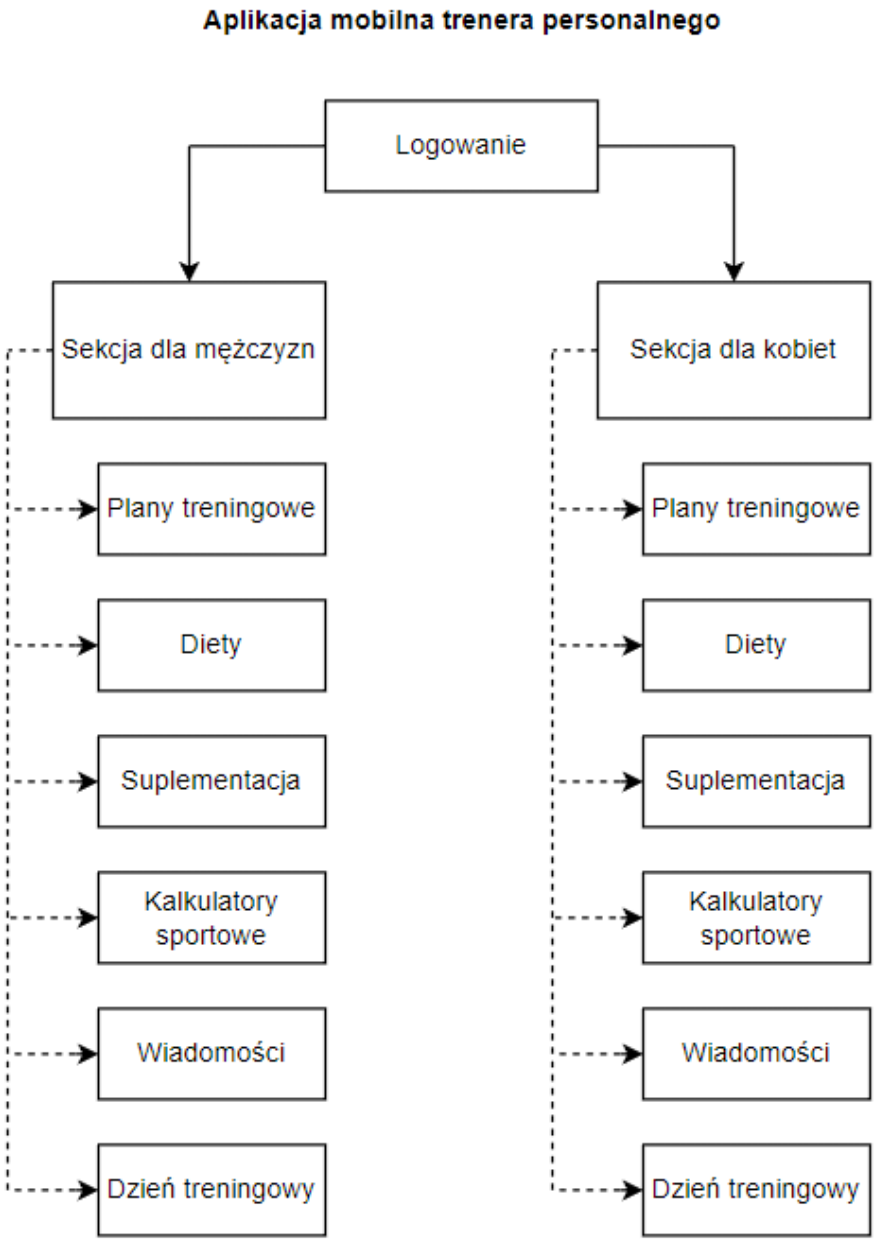
Android Studio ma wbudowane narzędzia projektowania widoków aplikacji w trybie WYSIYG (ang. What You See Is What You Get – To co widzisz jest tym co otrzymasz), który pozwala na uzyskanie rezultatu bardzo podobnego lub identycznego z tym co widać w środowisku programistycznym.

Środowisko pozwala również na testowanie zaprogramowanych rozwiązań bez dostępu do fizycznego urządzenia z Androidem poprzez wbudowane emulatory [7]

3. Opracowanie funkcjonalności aplikacji

3.1. Założenia projektowe aplikacji

Funkcjonalność aplikacji mobilnej trenera personalnego została skonstruowana z myślą o każdym użytkowniku bez względu na stopień zaawansowania umiejętności posługiwania się urządzeniem smartphone. Dzięki temu każda osoba zainteresowana będzie mogła skorzystać z rozwiązania realizowanego w ramach poniższej pracy dyplomowej i czerpać z niej maksymalne korzyści.



Rys 3.1. Funkcjonalność aplikacji

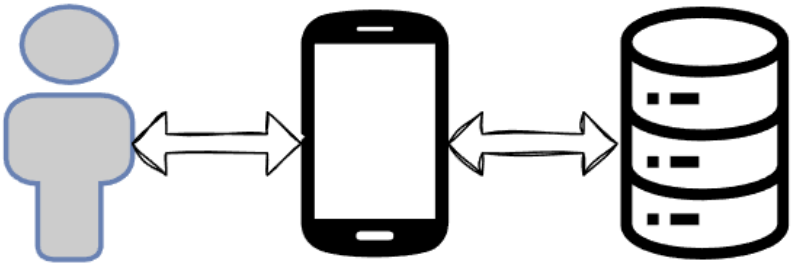
Na rysunku 3.1 została przedstawiona funkcjonalność aplikacji. Aplikacja mobilna trenera personalnego została podzielona na dwie sekcje – dla kobiet oraz dla mężczyzn. Podział został zastosowany w celu maksymalizacji komfortu użytkownika, ponieważ różnice biologiczne między mężczyznami i kobietami są na tyle znaczące, że wymagają osobnego podejścia treningowego. Użytkownik po zalogowaniu się do aplikacji będzie mógł wybrać jedną z dwóch przygotowanych sekcji. Zaimplementowanie logowania obejmuje również możliwość zarejestrowania się do systemu w celu możliwości korzystania z aplikacji.

Każda z sekcji posiada swoje podsekcje. Nazwy podsekcji w obu sekcjach są identyczne jednak różnią się one między sobą zawartością merytoryczną oraz wyglądem graficznym. Dla obu sekcji mamy : plany treningowe, diety suplementację, kalkulatory sportowe, wiadomości oraz dzień treningowy. Każda z tych podsekcji posiada swoją własną indywidualną funkcjonalność, która zostanie opisana w dalszej części rozdziału.

3.2. Struktura działania aplikacji

Aplikacja mobilna trenera personalnego jest przykładem aplikacji mobilnej samodzielnej. Dla takiego rodzaju aplikacji nie ma potrzeby uzyskiwania dostępu do zasobów Internetu w celu zalogowania się, a wystarczy jedynie urządzenie z zainstalowaną aplikacją. Przykładem takiego rozwiązania używanego powszechnie w urządzeniach smartphone jest powszechnie używana aplikacja kalkulatora.

Zastosowanie typu samodzielnego dla aplikacji mobilnej trenera personalnego nie powoduje żadnych ograniczeń w założeniach projektowych oraz funkcjonalności. Wszystkie funkcje oraz udogodnienia do których przyzwyczajony jest użytkownik podczas używania innych aplikacji zainstalowanych na swoim urządzeniu zostały zaimplementowane do wykonywania się lokalnie to jest na maszynie użytkownika jak zostało to zobrazowane na schemacie aplikacji przedstawionej na rysunku 3.2.

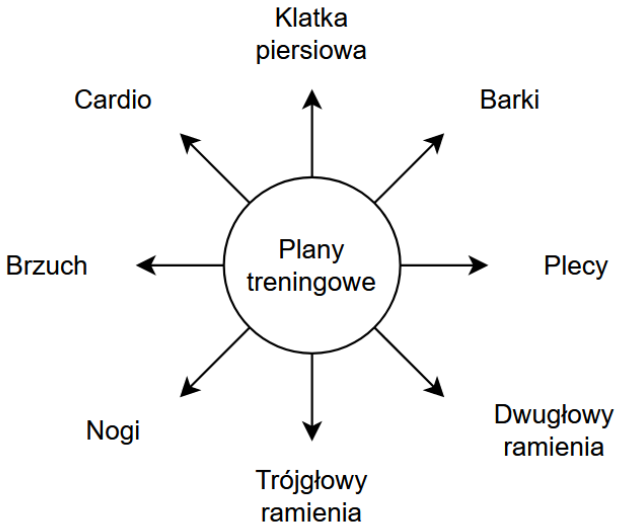


Rys 3.2. Schemat działania aplikacji

Zgodnie ze schematem przedstawionym poniżej użytkownik komunikuje się ze swoim urządzeniem mobilnym i to urządzenie mobilne przechowuje dane poprzez komunikację z bazą danych. Aplikacja samodzielna przechowuje bazę danych w swojej pamięci dzięki czemu użytkownik ma do niej dostęp bez konieczności łączenia się z siecią. Do wad takiego rozwiązania bez wątpienia należy zaliczyć zużycie miejsca, które generuje takie rozwiązanie, ponieważ dane muszą być przechowywane na urządzeniu co wymaga zasobów. Założenie aplikacji to jednak maksymalna wygoda i komfort dla użytkownika dlatego zdecydowano się podjąć ten krok kosztem dostępnej pamięci.

3.3. Plany treningowe

Pierwszą sekcją w aplikacji mobilnej trenera personalnego są plany treningowe dobrane odpowiednio w zależności od płci, dla której wybrana jest sekcja treningowa. Budowa ciała mężczyzn i kobiet, jak również układ hormonalny znacznie się różnią dlatego ważne jest odpowiednie dobranie metodyki treningowej. W przypadku mężczyzn będzie to nastawienie na trening siłowy górnych partii ciała, natomiast u kobiet zastosować można większą intensywność przy dolnych partiach ciała, nie zapominając jednocześnie o rozwoju pozostałych partii. W podsekcji plany treningowe użytkownik znaleźć może kolejno ćwiczenia dla mięśni : klatki piersiowej, barków , pleców, brzucha, nóg, dwugłowego i trójgłowego ramienia. W omawianej sekcji znaleźć można również trening cardio, czyli ćwiczenie aerobowe – o niskiej lub średniej intensywności.



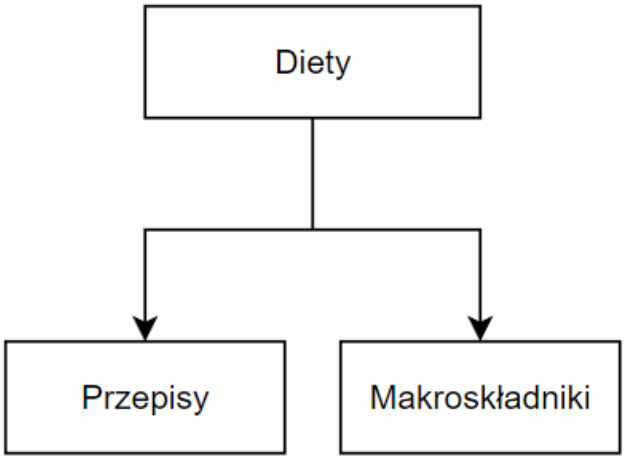
Rys 3.3 Schemat sekcji plany treningowe.

Zgodnie ze schematem z rysunku 3.3 sekcja plany treningowe zawiera osiem indywidualnych sekcji w której użytkownik może znaleźć plany treningowe dla każdej z partii mięśniowych. Implementacja sekcji polega na wyborze jednej z opcji dostępnej w postaci listy elementów z wykorzystaniem komponentu RecyclerView. Wybór z listy prowadzi do nowej aktywności zawierającej informacje na temat danej partii mięśniowej.

3.4. Diety

Dobrze zbilansowana dieta jest znacznie ważniejsza niż najlepiej wykonany trening siłowy. Zdrowe odżywanie pozwala osiągnąć lepszą sprawność fizyczną oraz umysłową, wpływa na samopoczucie oraz ogólny rozwój całego organizmu. Myśląc o rozpoczęciu swojej przygody ze sportami siłowymi, jak również będąc na zaawansowanym etapie należy mieć na uwadze utrzymanie dobrze zbilansowanej diety. Ilość poszczególnym makroskładników będzie zależała przede wszystkim od celu do osiągnięcia jaki stawia sobie dana osoba. Zawartość tłuszczu, białka oraz węglowodanów będzie się różniła w zależności od chęci zbudowania masy mięśniowej i docelowo przybrania na wadze czy też utraty tkanki tłuszczowej i tym samym utraty części masy mięśniowej z powodu zmniejszonej podaży kalorycznej.

Na poniższym schemacie na rysunku 3.4 została przedstawiona funkcjonalność sekcji poświęconej dietom.



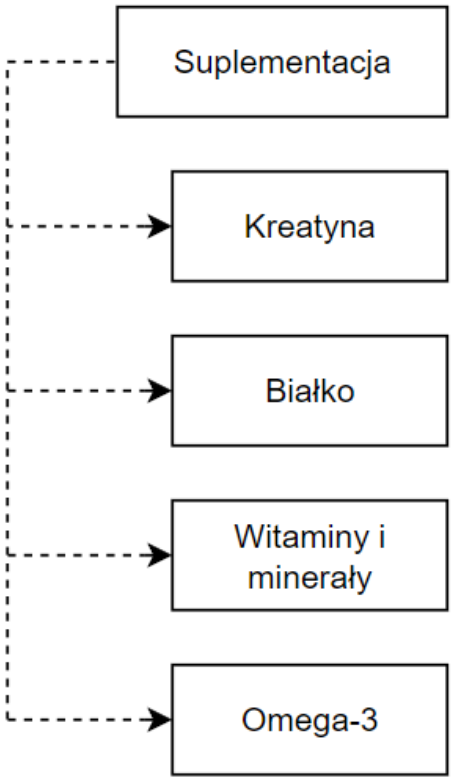
Rys 3.4 Schemat funkcjonalności sekcji diet

Sekcja diet składała się będzie z dwóch podsekcji. Pierwsza będzie poświęcona przepisom umożliwiającym dostarczenie odpowiedniej ilości makroskładników, natomiast druga to informacje odnośnie makroskładników z wyjaśnieniem co do ich stosowania oraz wszelkie zalecenia. Sekcja przepisów pozwoli wybrać przepis z listy i po kliknięciu elementu na liście użytkownik zostanie przeniesiony do linku z przepisem. Dzieje się tak również po naciśnięciu przycisku z napisem „makroskładniki". Użytkownik zostaje przeniesiony do strony internetowej opisującej to zagadnienie.

3.5. Suplementacja

W mediach suplementacja często traktowana jest jako cudowny środek, który natychmiast poprawi wyniki w sporcie, jak również pomoże bez wysiłku osiągnąć wymarzoną sylwetkę. Należy jednak mieć na uwadze, że suplementacja jest jedynie uzupełnieniem dobrze zbilansowanej diety i nie należy stosować jej jako zamiennik dla dobrze zbilansowanego posiłku.

Temat suplementacji na początku wzbudzał mieszane uczucia osób zajmujących się treningami siłowymi. Powodowała to niewystarczająca ilość badań, której substancje to zostały poddane. Dzisiaj jednak przy znacznych możliwościach technologicznych i zdecydowanemu upływowi czasu przeciętny użytkownik może bez problemu przy pomocy Internetu zdobyć niezbędną wiedzę popartą szeregiem badań naukowych.

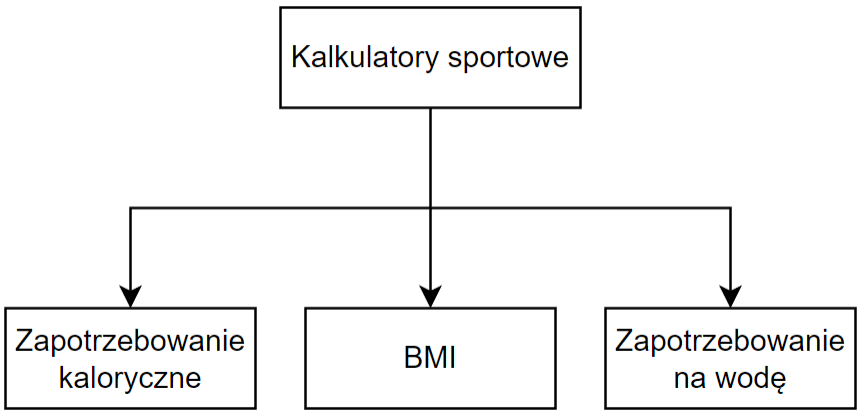


Rys 3.5 Schemat sekcji suplementacji

Na rysunku 3.5, przedstawionym powyżej, znajduje się schemat sekcji poświęconej suplementacji. Sekcja ta będzie zawierała przegląd najpopularniejszych i najbezpieczniejszych suplementów diety, które pomogą użytkownikowi na zdobycie niezbędnej wiedzy w tym zakresie. Z powyższej listy użytkownik może wybrać spośród 4 dostępnych opcji. Kliknięcie w element przenosi użytkownika do nowej aktywności, w której opisany jest dany suplement.

3.6. Kalkulatory sportowe

Kalkulatory sportowe odgrywają ważną role w początkach przygody ze sportami siłowymi. Dostarczanie makroskładników w codziennych posiłkach jest istotne, jednak należy mieć na uwadze ich ilość, żeby postawiony cel został osiągnięty. Podczas utraty tkanki tłuszczowej należy zmniejszyć podaż kaloryczną. Przeprowadzanie tego procesu metodą „na oko”, bez analizy i odpowiednich obliczeń może skończyć się pogorszeniem stanu zdrowia i prowadzić do wyniszczenia organizmu. Należy zatem skorzystać z odpowiednich kalkulatorów, które co prawda obarczone są pewnym błędem, ale pozwolą naświetlić kierunek działań dla osiągnięcia wcześniej postawionego celu.



Rys 3.6 Schemat sekcji kalkulatorów sportowych

Rysunek 3.6 przedstawiony powyżej zawiera schemat sekcji kalkulatorów sportowych. Sekcja ta składa się z kalkulatora zapotrzebowania kalorycznego, BMI oraz zapotrzebowania na wodę.

3.6.1. *Zapotrzebowanie kaloryczne*

Zapotrzebowanie kaloryczne to dzienne zapotrzebowanie człowieka potrzebne do utrzymania podstawowych funkcji organizmu. Zapotrzebowanie obliczane jest w sposób orientacyjny i może różnić się u poszczególnych osób ze względu na ich cechy indywidualne [8]

$$PPM = 10 * Waga[kg] + 6.25 * Wzrost[cm] - 5 * Wiek + 5 \quad (1.1)$$

$$PPM = 10 * Waga[kg] + 6.25 * Wzrost[cm] - 5 * Wiek - 161 \quad (1.2)$$

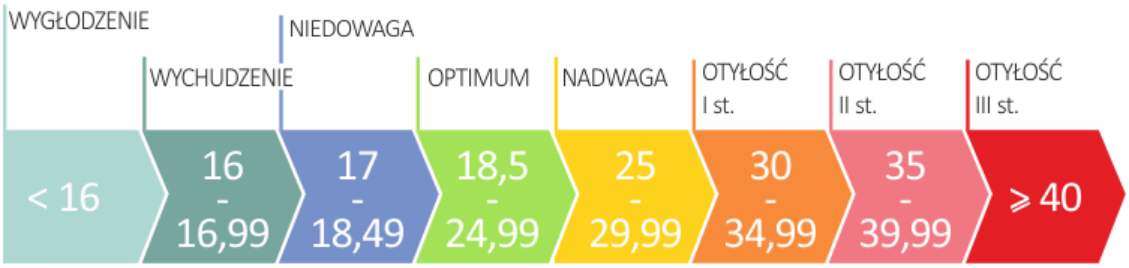
Wzory potrzebne do obliczenia zapotrzebowania kalorycznego zostały przedstawione powyżej. Wzór (1.1) odnosi się do zapotrzebowania dla mężczyzn, natomiast wzór (1.2) odnosi się do zapotrzebowania kalorycznego dla kobiet. Do obliczeń została zastosowana formuła Mifflina.

3.6.2. *BMI*

BMI (ang. Body Mass Index – wskaźnik masy ciała) to współczynnik pozwalający w ocenie zagrożenia chorobami głównie związanymi z nadwagą i otyłością. Wskaźnik BMI został opracowany wyłącznie dla osób dorosłych. Dla dzieci zostały opracowane tzw. Siatki centylowe, które pozwalają na obiektywną ocenę rozwoju fizycznego dzieci.

$$BMI = \frac{masa\ ciała\ [kg]}{wzrost^2[m^2]} \tag{1.3}$$

Wzór do obliczania indeksu BMI został przedstawiony powyżej w zależności (1.3). Indeks oblicza się dzieląc masę ciała podaną w kilogramach przez wzrost badanego podniesiony do kwadratu [9]. Na rysunku 3.7, przedstawionym poniżej zostały podane zakresy dla wskaźnika BMI.



Rys 3.7. Zakresy wskaźnika BMI [10]

3.6.3. *Zapotrzebowanie na wodę*

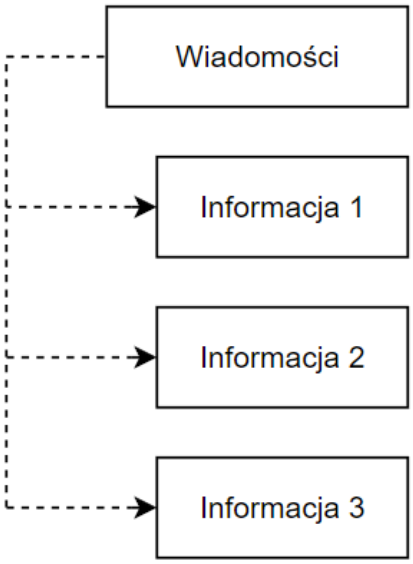
Człowiek składa się w około 78 procentach z wody. Dlatego istotne jest regularne i odpowiednie nawadnianie organizmu. Często słyszy się o przybliżonych ilościach jakie człowiek powinien wypić w trakcie dnia. Jednak są to zazwyczaj uproszczenia i przybliżenia które nie są odpowiednie dla każdej osoby.

$$DZNW\ [l] = masa\ ciała\ [kg] * 30\ [ml] \tag{1.4}$$

Zależność (1.4) opisuje dzienne zapotrzebowanie na wodę. Zależność wykorzystuje masę ciała i mnoży ją przez 30 ml co jest wartością sugerowaną do wypicia na jeden kilogram masy ciała [11].

3.7. Wiadomości

Sekcja poświęcona wiadomościom będzie dostarczała najnowsze informacje ze świata sportów siłowych a także podstawowe informacje, które mogą okazać się przydatne w rozpoczęciu przygody ze sportami siłowymi. Informacje będą prezentowane w postaci listy, która będzie wygodna i prosta w obsłudze dla użytkownika.

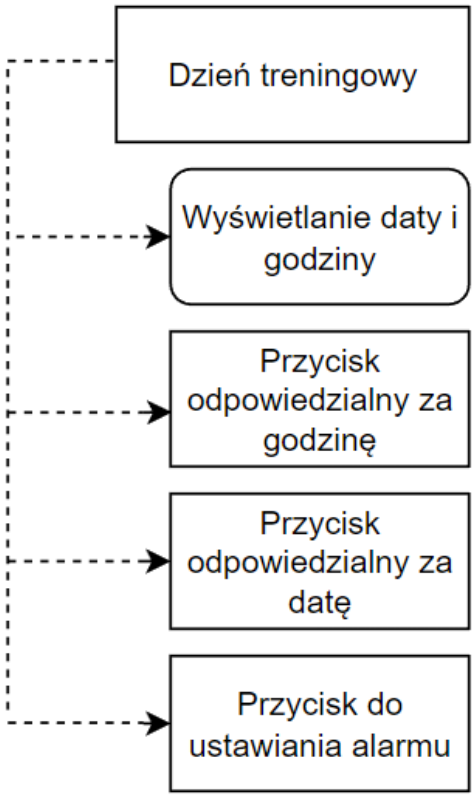


Rys 3.8. Schemat sekcji wiadomości.

Na rysunku 3.8 przedstawionym powyżej został zaprezentowany schemat sekcji wiadomości. Informacje w tej sekcji zostaną dodane za pomocą RecyclerView w celu stworzenia listy dynamicznej pozwalającej na prosty i czytelny wybór szukanej informacji. Kliknięcie w tytuł z informacją pozwoli na przekierowanie użytkownika do artykułu, w którym dana informacja została opisana.

3.8. Dzień treningowy

Sekcja o nazwie dzień treningowy odpowiada za przypomnienie o treningu. Użytkownik poprzez ustawienie godziny oraz daty treningu uzyska stosowne powiadomienie od aplikacji, że zbliża się pora treningu. Sekcja działa na zasadzie „remindera” (ang.remind – przypomnieć) i jest odpowiedzialna za przypominanie użytkownikowi nie tylko o dacie treningu, ale również o samej aplikacji przez co jest większa szansa, że wróci do niej w przyszłości. Bez interakcji użytkownika z aplikacją istnieje możliwość, że użytkownik zapomni o aplikacji, bądź też przestanie go ona interesować dlatego stosowanie różnego rodzaju interakcji stało się powszechną praktyką, w odniesieniu do programowania do aplikacji na platformy mobilne.



Rys 3.9. Schemat sekcji dzień treningowy.

Sekcja dzień treningowy będzie zawierała przycisk do ustawiania godziny treningu oraz osobny przycisk do ustawiania daty. W celu potwierdzenia ustawienia alarmu został zaimplementowany osobny przycisk wraz ze stosownym komunikatem. Po poprawnym ustawieniu alarmu zostanie wyświetlona wiadomości okienkowa na telefonie, co zostanie przedstawione w dalszej części pracy.

4. Opracowanie interfejsu graficznego użytkownika

Podczas opracowania interfejsu graficznego użytkownika dla aplikacji mobilnej trenera personalnego należało przyjąć podejście prostoty oraz przejrzystości obsługi aplikacji. Wszystkie elementy zostały opisane dokładnie w celu maksymalnego komfortu użytkowania. Aplikacja na bieżąco komunikuje się z użytkownikiem poprzez komunikaty „pop-up" (ang.pop-up – wyskakujące okienko), które informują do jakiej sekcji użytkownik został przekierowany, bądź też czy operacja, którą wykonuje użytkownik zakończyła się powodzeniem. Takie rozwiązanie pozwala zachować interakcje z użytkownikiem i daje poczucie, że aplikacja cały czas z nim współpracuje poprzez otrzymywany „feedback" (ang.feedback – informacja zwrotna).

Jak już zostało wspomniane założeniem aplikacji było maksymalne uproszczenie jej obsługi. Implementacja różnorodnych elementów graficznych ułatwia użytkownikowi poruszanie się pomiędzy danymi sekcjami i ewentualne ich zapamiętywanie w celu wygodniejszego poruszania się po aplikacji w przyszłości. Z myślą o osobach, które nie są biegłe w obsłudze smartphonów wykorzystano całą dostępną przestrzeń ekranu smartphone w celu implementacji dużych oraz widocznych przycisków. Zastosowano również szereg zabezpieczeń przed ewentualnymi błędami krytycznymi związanymi z użytkowaniem aplikacji o czym szerzej zostało opisane w rozdziale piątym dotyczącym weryfikacji eksperymentalnej aplikacji, w której to odniesiono się do zabezpieczeń oraz poprawności implementacji.

W celu urozmaicenia zastosowano motywy kolorystyczne dla obu sekcji z podziałem na płeć użytkownika. Sekcja dla kobiet została utrzymana w łagodniejszych i cieplejszych kolorach, natomiast sekcja dla mężczyzn bazuje na ciemniejszym motywie kolorystycznym.

4.1. Logowanie do aplikacji

W aplikacji zaimplementowany został system logowania oraz rejestracji. Po uruchomieniu aplikacji użytkownik zostaje przeniesiony do ekranu rejestracji jak zostało to przedstawione na rysunku 4.1 poniżej.

Rys 4.1. Ekran systemu rejestracji.

Użytkownik ma możliwość wprowadzić preferowaną nazwę oraz ustawić odpowiednie dla siebie hasło. W przypadku rejestracji wymagane jest wprowadzenie hasła dwukrotnie w celu weryfikacji poprawności jego wpisania. Po poprawnym wypełnieniu wszystkich pól dostępnych w tej aktywności użytkownik może się zarejestrować poprzez wciśnięcie przycisku „Rejestracja”. Spowoduje to dodanie jego konta (nazwy użytkownika oraz hasła) do bazy danych aplikacji, o której mowa będzie w następnym podrozdziale. W przypadku kiedy użytkownik ma już utworzone konto może przejść bezpośrednio do aktywności logowania poprzez wciśnięcie przycisku „Mam już konto, przejdź do logowania”.

```
1   btngotologin.setOnClickListener(){
2       val intent = Intent(this,LoginActivity::class.java)
3       this.startActivity(intent)
4   }
5   btnsignup.setOnClickListener(){
6       val unametext = uname.text.toString()
7       val pwordtext = pword.text.toString()
8       val pword2text = pword2.text.toString()
9       val savedata = db.insertdata(unametext, pwordtext)
10
11      if(TextUtils.isEmpty(unametext) || TextUtils.isEmpty(pwordtext) || TextUtils.isEmpty(pword2text)){
12          Toast.makeText(this, "Dodaj nazwe użytkownika, hasło i potwierdzienie hasła", Toast.LENGTH_SHORT).show()
13      }
14      else{
15          if(pwordtext.equals(pword2text)){
16              if(savedata==true){
17                  Toast.makeText(this,"Rejestracja pomyślna", Toast.LENGTH_SHORT).show()
18                  val intent = Intent(applicationContext,LoginActivity::class.java)
19                  startActivity(intent)
20              }
21              else{
22                  Toast.makeText(this,"Użytkownik już istnieje", Toast.LENGTH_SHORT).show()
23              }
24          }
25          else{
26              Toast.makeText(this,"Hasła nie są takie same",Toast.LENGTH_SHORT).show()
27          }
```

Listing 4.1 Algorytm rejestracji.

Powyżej w listingu pierwszym został przedstawiony algorytm odpowiadający za rejestrację użytkownika. Linie 1-4 odpowiadają za przekierowanie użytkownika do sekcji logowania w przypadku posiadania konta w systemie. Następnie w odwołaniu się do przycisku odpowiadającego za rejestrację jest instrukcja warunkowa sprawdzająca, czy wszystkie pola zostały wypełnione. Kolejna instrukcja, która jest zagnieżdżona w instrukcji głównej sprawdza, czy hasła są takie same. W przypadku błędu haseł zostaje wyświetlony stosowny komunikat oraz w przypadku poprawnej rejestracji użytkownik zostaje powiadomiony o dodaniu jego konta do bazy danych. W linii 16 następuje porównanie danych zawartych w bazie oraz loginu i hasła otrzymanego od użytkownika. Jeżeli w bazie danych znajduje się już dany użytkownik o takim samym loginie oraz haśle to na ekranie zostaje wyświetlony stosowny komunikat, który informuje o takim przypadku.

Użytkownik podczas tworzenia konta oraz ustanawiania hasła może posługiwać się dowolną długością oraz kombinacją znaków. Dla rejestracji nie zastosowano większych zabezpieczeń, niż wymaga tego poprawne i komfortowe korzystanie z aplikacji w związku z tym, że baza danych znajduje się na urządzeniu użytkownika i w każdej chwili ma do niej dostęp w celu sprawdzenia zarejestrowanych użytkowników oraz wedle własnych potrzeb przetwarzania tymi danymi.

Rys 4.2 Ekran systemu logowania.

Na rysunku 4.2, przedstawionym powyżej, został zamieszczony wygląd aktywności odpowiadającej za logowanie do aplikacji mobilnej trenera personalnego. Aktywność składa się z pola do wpisania nazwy użytkownika oraz hasła. Nie zastosowano tutaj możliwości przejścia do systemu rejestracji, ponieważ jest ona domyślnie wyświetlana jako pierwsza aktywność po włączeniu aplikacji. Użytkownik który ma konto, wie że należy przejść bezpośrednio do systemu logowania, natomiast dla nowych użytkowników jest to wyłącznie ułatwienie. Powrót do ekranu rejestracji z systemu logowania jest możliwy poprzez użycie klawiszy funkcyjnych znajdujących się na dole smartphona.

```
1       loginbutton.setOnClickListener(){
2           val useredtx = edituser.text.toString()
3           val passedtx = editpword.text.toString()
4
5           if(TextUtils.isEmpty(useredtx) || TextUtils.isEmpty(passedtx)){
6               Toast.makeText(this,"Podaj nazwę użytkownika i hasło", Toast.LENGTH_SHORT).show()
7           }
8           else
9           {
10              val checkuser = dbh.checkuserpass(useredtx,passedtx)
11              if(checkuser==true){
12                  Toast.makeText(this,"Logowanie pomyślne", Toast.LENGTH_SHORT).show()
13                  val intent = Intent(applicationContext,Wybor_mezczyzna_kobieta::class.java)
14                  startActivity(intent)
15              }
16              else{
17                  Toast.makeText(this,"Błędna nazwa użytkownika lub hasło",Toast.LENGTH_SHORT).show()
18              }
```

Listing 4.2. Algorytm logowania.

W listingu drugim przedstawionym powyżej znajduje się implementacja aktywności odpowiadającej za logowanie do aplikacji mobilnej trenera personalnego. W przypadku nie wpisania w pole loginu bądź hasła żadnej wartości użytkownik otrzymuje informację o, tym, że należy podać nazwę użytkownika oraz stosowne hasło. Po poprawnym wypełnieniu dane są sprawdzane z bazą co zostało zaprezentowane w linii 10. Jeżeli wartości są poprawne użytkownik zostaje przekierowany do aplikacji, natomiast po wpisaniu błędnych danych zostaje wyświetlona informacja zwrotna w postaci stosownego komunikatu.

4.2. Implementacja bazy danych

Baza danych do aplikacji została zaimplementowana na bazie SQLite. SQLite to relacyjna baza danych typu „open source” (ang.open source – ogólnodostępny kod źródłowy), która służy do wykonywania operacji na bazach danych dla urządzeń z systemem Android. Do operacji tych można zaliczyć przechowywanie, manipulowanie bądź pobieranie danych z bazy. Domyślnie baza danych SQLite jest osadzona w systemie Android i nie ma potrzeby jej konfigurowania. Do korzystania z bazy danych SQLite została stworzona specjalna klasa „SQLiteOpenHelper”, która służy do tworzenia baz danych i zarządzania wersjami. Wedle indywidualnych potrzeb klasa „SQLiteOpenHelper” ma w sobie wiele wbudowanych metod pozwalających na aktualizację wierszy czy też wstawianie dodatkowych rekordów do bazy. Dzięki temu osoba chcąca użyć bazy danych w swoim projekcie otrzymuje komfort w postaci gotowego kodu, możliwego do użycia.

```
override fun onCreate(p0: SQLiteDatabase?) {
    p0?.execSQL("create table Userdata (username TEXT primary key, password TEXT)")
}

override fun onUpgrade(p0: SQLiteDatabase?, p1: Int, p2: Int) {
    p0?.execSQL("drop table if exists Userdata")
}
```

Listing 4.3. Implementacja bazy danych.

Baza danych zaimplementowana w aplikacji składa się z trzech części. Część pierwsza przedstawiona w listingu 4.3, to metoda „onCreate” wywoływana podczas tworzenia bazy danych pierwszy raz. W tej metodzie tworzymy nową tabelę „Userdata” z dwoma kolumnami „username” oraz „password” odpowiadającą kolejno za nazwę użytkownika oraz hasło, przy czym username ustawiona jest jako klucz podstawowy bazy. Druga metoda to wcześniej wspomniana metoda do aktualizacji bazy danych „onUpgrade”. Metoda ta wywoływana jest w przypadku wprowadzenia zmian w bazie przykładowo podczas tworzenia nowego użytkownika.

```
fun insertdata(username: String, password: String): Boolean {
        val p0 = this.writableDatabase
        val cv = ContentValues()
        cv.put("username", username)
        cv.put("password", password)
        val result = p0.insert("Userdata", null, cv)

        if (result==-1 .toLong()){
            return false
        }
        return true
    }
```

Listing 4.4. Dodawanie użytkownika do bazy.

Metoda „insertdata” przedstawiona w listingu 4.4 służy za wstawienie nowego wiersza do tabeli Userdata z określonymi wartościami nazwy użytkownika oraz hasła, które zostały podane podczas rejestracji. Instrukcja warunkowa „if” sprawdza czy wartość wynikowa operacji wynosi -1. Zostało to zastosowane w celu sprawdzenia powodzenia operacji wstawienia rekordu do tabeli. W przypadku niepowodzenia zwraca ona wartość „false”, natomiast podczas powodzenia instrukcja warunkowa zostaje pominięta a cała metoda zwraca wartość „true” sygnalizującą powodzenie operacji.

```
fun checkuserpass(username: String, password: String): Boolean {
        val p0 = this.writableDatabase
        val query = "select * from Userdata where username= '$username' and password= " +
                "'$password'"
        val cursor = p0.rawQuery(query,null)
        if (cursor.count<=0){
            cursor.close()
            return false
        }
        cursor.close()
        return true
    }
```

Listing 4.5. Sprawdzanie użytkownika w bazie danych.

Metoda „checkuserpass” przedstawiona w listingu 4.5 służy do sprawdzania, czy w tabeli Userdata bazy danych istnieje dany użytkownik. Metoda przyjmuje dwa argumenty – nazwę użytkownika oraz hasło. Pierwszy wiersz metody tworzy zapisaną instanchę bazy danych poprzez wywołanie metody „writableDatabase” klasy „SQLiteOpenHelper”. Następny wiersz tworzy ciąg zapytań, który wybiera wszystkie wpiersze z tabeli „Userdata”, w których są zapisane nazwy użytkownika oraz hasła. Metoda „rawQuery” jest wywoływana na obiekcie „p0”, który jest instancją SQLiteDatabase reprezentującą wyszukiwaną bazę danych. Metoda „rawQuery” wykonuje zapytanie i zwraca obiekt „Cursor”, który zawiera zestaw wyników zapytania. Instrukcja „if” po zapytaniu sprawdza, czy obiekt kursora zawiera jakieś wiersze. Jeśli liczba wierszy jest mniejsza lub równa 0, oznacza to, że kombinacja nazwy użytkownika i hasła nie istnieje w tabeli, więc metoda zwraca wartość „false”. Jeśli w tabeli istnieje kombinacja nazwy użytkownika i hasła, metoda zwraca wartość „true”. Na koniec kursor zostaje zamykany, aby zwolnić zasoby używane przez ten obiekt.

Algorytm bazy danych został zaimplementowany wyłącznie w celu weryfikacji użytkownika. W pozostałych aktywnościach aplikacji nie ma potrzeby magazynowania danych wprowadzanych przez użytkownika w związku z tym ograniczono się tylko do jednej bazy. W związku z tym, że baza jest przechowywana na urządzeniu użytkownika wraz z całą aplikacją nie ma potrzeby uzyskiwania dostępu do zasobów Internetu w celu zalogowania się do aplikacji.

4.3. Wybór między sekcją dla kobiet i mężczyzn

Po zalogowaniu się do aplikacji użytkownik może wybrać między sekcją przeznaczoną dla kobiet oraz dla mężczyzn jak zostało to przedstawione na rysunku 17.

Rys 4.3 Wybór miedzy sekcją dla kobiet oraz mężczyzn

Każdy z tych przycisków przenosi użytkownika do osobnego panelu, który jest odpowiednio spersonalizowany pod daną płeć. Trening dla kobiet oraz dla mężczyzn znacząco różni się od siebie nie tylko pod względem ciężarów jakie można podnieść, ale również ze względu na samo podejście treningowe. W treningach kobiet uwaga skupia się bardziej na dolnych partiach ciała w przeciwieństwie do treningu mężczyzn. Również wszelkie wymagania oraz zastosowane obliczenia do kalkulatorów, o których będzie mowa w późniejszej części pracy, różnią się w zależności od płci.

```
1  pm.setOnClickListener(){
2      Toast.makeText(this,"Sekcja dla mężczyzn", Toast.LENGTH_SHORT).show()
3      val intent = Intent(applicationContext,sekcja_dla_mezczyzn::class.java)
4      startActivity(intent)
5  }
6
7  pk.setOnClickListener(){
8      Toast.makeText(this,"Sekcja dla kobiet", Toast.LENGTH_SHORT).show()
9      val intent = Intent(applicationContext,Sekcja_dla_kobiet::class.java)
10     startActivity(intent)
11 }
```

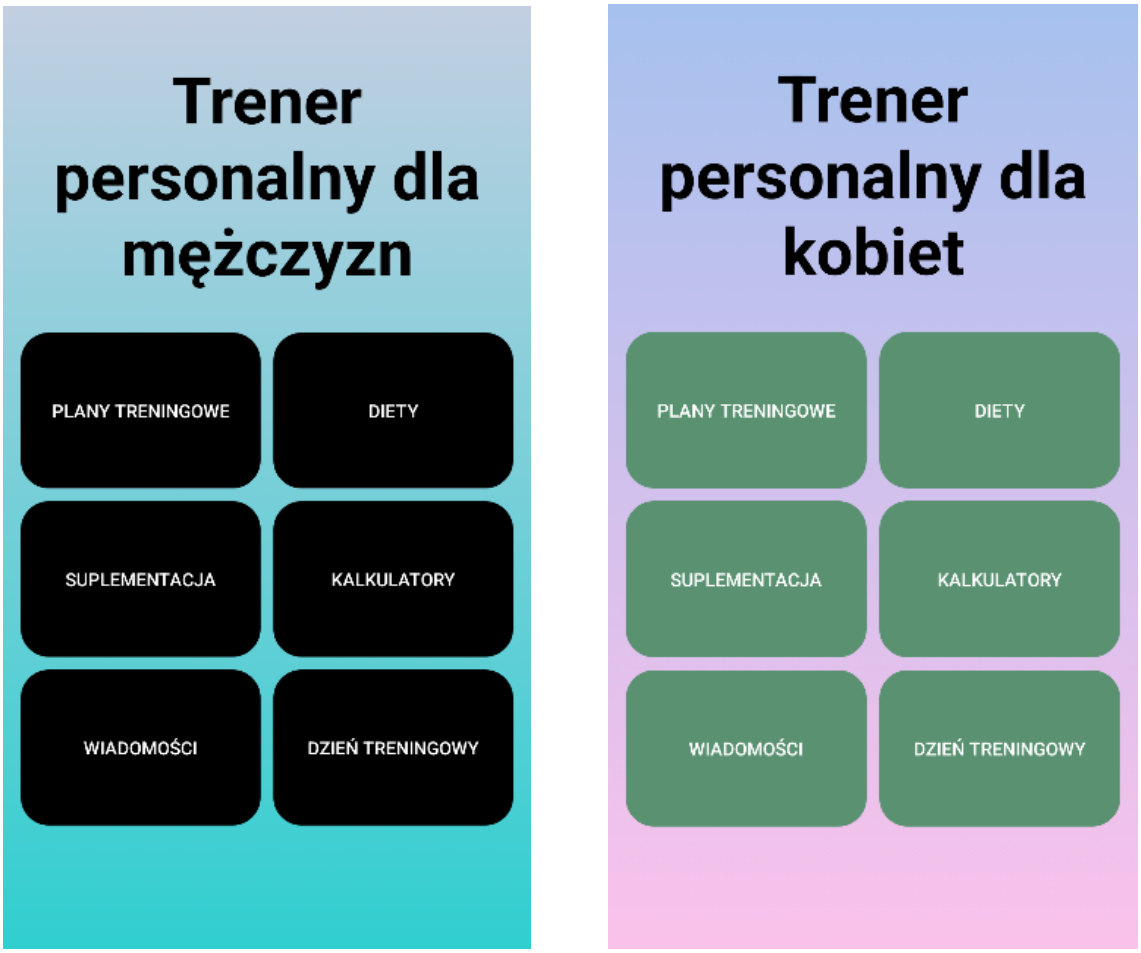
Listing 4.6. Wybór panelu.

Powyżej w listingu 4.6 została przedstawiona implementacja wyboru płci. Metoda obsługująca pierwszy przycisk została przedstawiona między liniami 1-5. Po wciśnięciu przycisku z napisem „Dla Mężczyzn" użytkownik zostaje przeniesiony do stosownego panelu poświęconego sekcji dla mężczyzn oraz zostaje wyświetlony komunikat „Sekcja dla mężczyzn" informujący o tym, że przejście do panelu zakończyło się powodzeniem. W przypadku obsługi metody odpowiadającej za przejście do panelu dla kobiet (linia 7-11), użytkownik po naciśnięciu przycisku „Dla Kobiet" zostaje przeniesiony do panelu poświęconego treningom dla kobiet oraz również jak w przypadku sekcji dla mężczyzn otrzymuje stosowny komunikat o powodzeniu się operacji.

4.4. Implementacja panelu głównego

Panel główny jest sercem całej aplikacji i służy do przechodzenia pomiędzy kolejnymi sekcjami. Z panelu głównego użytkownik może przejść bezpośrednio do każdej z sekcji opisanej w rozdziale 3.1 wedle schematu przedstawionego na rys.3.1. Dla każdej płci został dobrany odpowiedni design w celu urozmaicenia wyglądu aplikacji. Motywy prezentowane na panelu głównym zostały utrzymane w pozostałych sekcjach w celu zachowania spójności, co pomaga także użytkownikowi w przyzwyczajeniu się do danej sekcji pod względem kolorystycznym.

Wygląd poszczególnych paneli został przedstawiony poniżej na rysunku 4.4 Po lewej został przedstawiony panel główny dla mężczyzn, natomiast po prawej panel główny dla kobiet.



Rys 4.4 Wygląd panelu głównego – dla mężczyzn, po prawej stronie, oraz dla kobiet po lewej stronie.

Powyższe aktywności przedstawione na rysunku 4.4, składają się z sześciu funkcjonalnych przycisków. Każdy z nich przenosi użytkownika do osobno zaprojektowanej podsekcji, w której znajduje się zaimplementowane rozwiązanie danego tematu. Powrót do wyboru modelu treningowego z podziałem na płeć możliwy jest poprzez klawisze funkcjonalne. Po wybraniu odpowiedniej sekcji użytkownik zostaje poinformowany o pomyślnej operacji stosownym komunikatem wyświetlanym w formie wyskakującego okna zgodnego z nazwą sekcji do której się przeniósł.

Poniżej w listingu 4.7 został przedstawiony fragment implementacji sekcji dla mężczyzn. Implementacja ta jest identyczna w sekcji dla kobiet, różnią się wyłącznie nazwy zmiennych odpowiadających za przyciski dla danej aktywności.

```
pkPlany.setOnClickListener(){
    Toast.makeText(this,"Plany Treningowe", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,Dla_Mezczyzn_Plany_Treningowe::class.java)
    startActivity(intent)
}

pkDiety.setOnClickListener(){
    Toast.makeText(this,"Diety", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,Dla_Mezczyzn_Diety::class.java)
    startActivity(intent)
}
```

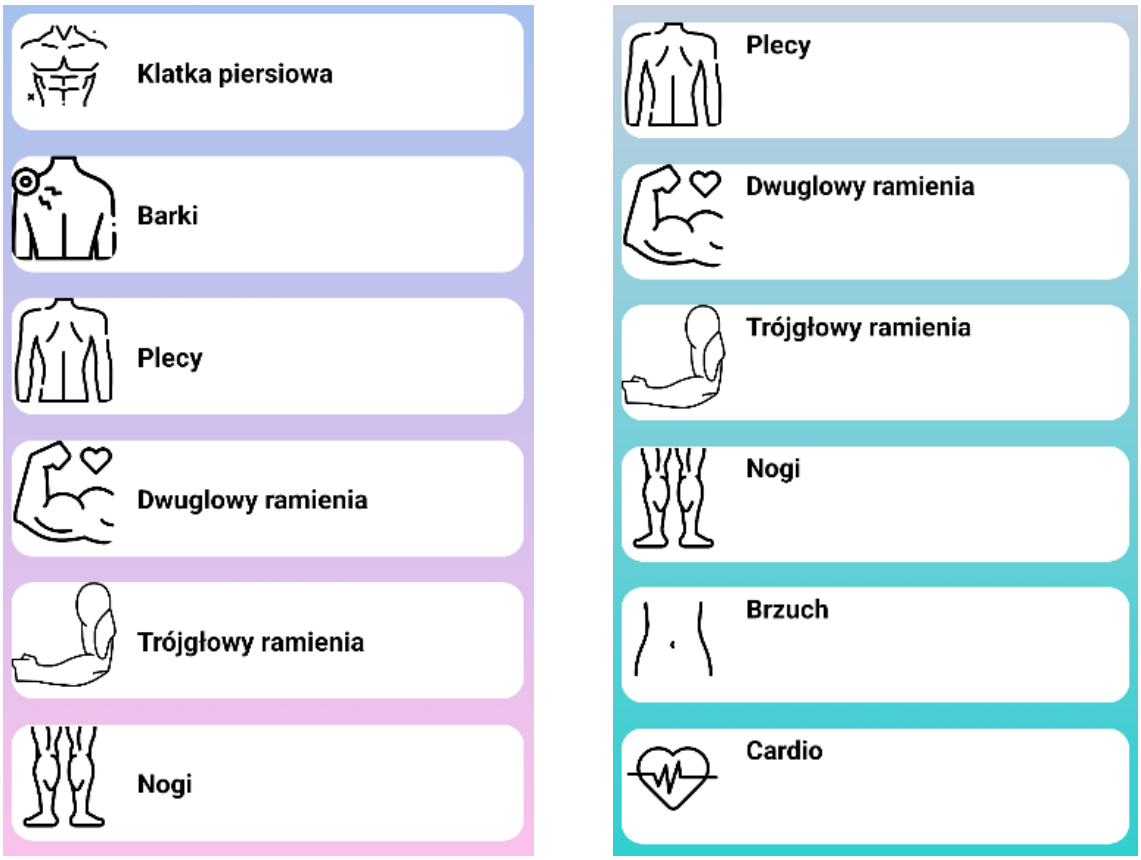
Listing 4.7. Fragment implementacji sekcji dla mężczyzn.

Powyższa implementacja przedstawia dwie metody będące fragmentem klasy odpowiadającej za obsługę sekcji dla mężczyzn. Powyższe metody odpowiadają za przeniesienie użytkownika do odpowiedniej aktywności zgodnie z jego wyborem. Wraz z przeniesieniem instrukcja „Toast.makeText" odpowiada za wyświetlenie stosownego komunikatu informującego użytkownika o nazwie sekcji do której został przekierowany. Powyższy fragment przedstawia implementację metod dla planów treningowych oraz diet. Pozostałe sekcje zostały zaprogramowane w sposób analogiczny, również w przypadku sekcji dla kobiet.

4.5. Sekcja planów treningowych

Sekcja planów treningowych jest pierwszą sekcją omawianą w ramach panelu głównego aplikacji mobilnej trenera personalnego. Została zaprojektowana zgodnie ze schematem przedstawionym na rysunku 3.3 oraz zgodnie z założeniami omówionymi w rozdziale 3.3, który poświęcony jest funkcjonalności sekcji planów treningowych. Zgodnie z wyżej wspomnianym schematem sekcja składa się z ośmiu podsekcji poświęconych każdej partii ciała oraz treningu cardio. Każda z wymienionych sekcji jest od siebie niezależna.

W celu utrzymania spójności wygląd koncepcyjny sekcji dla kobiet oraz dla mężczyzn jest identyczny. Zachowana została koncepcja kolorystyczna. Różniącym się istotnym aspektem w sekcji planów treningowych dla kobiet oraz dla mężczyzn jest zawartość merytoryczna w postaci innego rodzaju ćwiczeń.



Rys 4.5. Wygląd sekcji planów treningowych – dla kobiet z lewej strony oraz dla mężczyzn z prawej strony.

Na rysunku 4.5 przedstawionym powyżej został zaprezentowany wygląd graficzny sekcji planów treningowych. Sekcja składa się z listy ośmio-elementowej. Każdy element posiada swoją przestrzeń, nazwę oraz ikonę reprezentującą daną sekcję. W celu zachowania spójności aplikacji w obu sekcjach została wykorzystana ta sama grafika.

Lista została wykonana na zasadzie RecyclerView, który jest popularnym komponentem wykorzystywanym w systemie Andorid do wyświetlania listy elementów. Stosuje się go jako alternatywę do przedstawiania listy zamiast używania mniej elastycznych rozwiązań takich jak komponenty : ListView czy GridView. Główną zaletą RecyclerView jest rozwiązanie pozwalające na wyświetlanie tylko i wyłącznie elementów, które są potrzebne użytkownikowi w danym momencie na ekranie.

```
class AdapterDlaKobietPlan(private val newList: ArrayList<Plan>) :
    RecyclerView.Adapter<AdapterDlaKobietPlan.MyViewHolder>() {

    private lateinit var mListener: OnItemClickListener

    interface OnItemClickListener{
        fun onItemClick(position: Int)
    }

    fun setOnItemClickListener(listener: OnItemClickListener){
        mListener = listener
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(
            R.layout.list_item_dla_kobiet_plany_treningowe, parent, false)

        return MyViewHolder(itemView,mListener)
    }

    override fun getItemCount(): Int {
        return newList.size
    }
    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {

        val currentItem = newList[position]
        holder.titleImage.setImageResource(currentItem.titleImage)
        holder.Tytul.text=currentItem.heading

    }
    class MyViewHolder(itemView: View, listener: OnItemClickListener) : RecyclerView.ViewHolder(itemView){

        val titleImage : ShapeableImageView = itemView.findViewById(R.id.image_dla_kobiet_plany_treningowe)
        val Tytul : TextView = itemView.findViewById(R.id.Tytuł_dla_kobiet_plany_treningowe)
        init {

            itemView.setOnClickListener {
                listener.onItemClick(absoluteAdapterPosition)
            }
```

Listing 4.8. Implementacja Adaptera dla planów treningowych.

Adapter służący do wypełnienia listy RecyclerView został przedstawiony powyżej w listingu 4.8. Lista zostaje wypełniona obiektami o nazwie „Plan", które zwierają w sobie nazwę oraz numer identyfikacyjny dla zdjęcia, ponieważ operacje na zdjęciach takie jak kolejność czy wybór konkretnego zdjęcia dla danego obiektu odbywa się na liczbach całkowitych.

Klasa „AdapterDlaKobietPlan" zawiera w sobie konstruktor, który pobiera listę obiektów o nazwie Plan i inicjuje adapter z danymi. Poniżej mamy zainicjowany interfejs o nazwie „OnItemClickListener", który ma w sobie jedną metodę o nazwie „OnItemClick". Metoda ta służy do obsługi zdarzeń po kliknięciu w pojedynczy obiekt zaimplantowanego „RecyclerView", o którym będzie mowa w późniejszej części pracy.

Metoda „OnCreateViewHolder" służy do nakładania wyglądu na pojedyncze obiekty, który został stworzony w osobnym pliku XML file jako wzór wyglądu dla poszczególnych obiektów.

Za powiązanie danych z poszczególnymi pozycjami listy odpowiada metoda „onBindViewHolder” Pobiera dane z podanej pozycji w liście o nazwie „newList” i ustawia odpowiednie widoki w klasie „MyViewHolder” z danymi.

Metoda „getItemCount” zwraca rozmiar listy o nazwie newList.

Klasa „MyViewHolder” jest statyczną klasą wewnętrzną, która rozszerza „RecyclerView.ViewHolder” i przechowuje poszczególne widoki dla każdego elementu listy. Zmienne „titleImage” i „Tytul” zawierają odniesienia odpowiednio do widoków „ImageView” i „TextView”.

Blok „init” ustawia detektor kliknięć w widoku elementu i wywołuje metodę „onItemClick” elementu „OnItemClickListener” po kliknięciu elementu. Przekazuje pozycję klikniętego elementu jako argument do metody „onItemClick”.

W klasie głównej obsługującej sekcję planów treningowych znajdują się elementy odpowiedzialne za obsługę zawartości danych obiektów. Zostały zastosowane listy przechowujące zdjęcia, które pobrane zostały z folderu drawable, służącego do przechowywania elementów związanych z grafiką. W klasie tej zostały zdefiniowane również tytuły poszczególnych obiektów w postaci listy oraz zawartość pojedynczego obiektu, która wyświetla się użytkownikowi po naciśnięciu na ten obiekt. Opis obiektu został przechowany w folderze „values” w pliku „strings.xml”. Przykładowy wygląd zawartości „string.xml” dla treningu cardio dla kobiet został przedstawiony poniżej w listingu 4.9.

```
<string name="kobiety_cardio">

    W przypadku treningu cardio można zastosować dowolną maszynę jaka
    jest dostępna na siłowni, bądź jaką dysponujemy w domu.\n
    Istotne jest aby dostosować intensywność do swoich możliwości
    i nie wykonywać zbyt intensywnego treningu.\n
    Zalecane jest 5 min przed treningiem siłowym oraz 15 min po treningu siłowym.\n
    Czas ten może się różnić w zależności od przyjętych założeń oraz celu do jekiego dążymy.\n
    W przypadku redukcji tkanki tłuszczowej zalecana jest dłuższa sesja
    na urządzeniach jednak przy tej samej intensywności.\n

</string>
```

Lisitng 4.9. Wygląd obiektu w pliku strings.xml

```
imageId = arrayOf(
    R.drawable.body,
    R.drawable.shoulder2,
    R.drawable.back,
    R.drawable.muscle,
    R.drawable.triceps,
    R.drawable.leg,
    R.drawable.belly2,
    R.drawable.cardio
)

Tytul = arrayOf(
    "Klatka piersiowa",
    "Barki",
    "Plecy",
    "Dwuglowy ramienia",
    "Trójgłowy ramienia",
    "Nogi",
    "Brzuch",
    "Cardio"
)

Trening = arrayOf(
    getString(R.string.kobiety_klatka_piersiowa),
    getString(R.string.kobiety_barki),
    getString(R.string.kobiety_plecy),
    getString(R.string.kobiety_biceps),
    getString(R.string.kobiety_triceps),
    getString(R.string.kobiety_nogi),
    getString(R.string.kobiety_brzuch),
    getString(R.string.kobiety_cardio)
)
```

Listing 4.10. Opis zawartości dla klasy reprezentującej plany treningowe dla kobiet.

W listingu 4.10, przedstawionym powyżej została zobrazowana implementacja pobrania zawartości z folderu drawable oraz string.xml przechowujących dane dla obiektów listy. Tytuł dla poszczególnych obiektów został nadany bezpośrednio w klasie.

Pozostałe obiekty dla tej klasy, jak również dla klasy reprezentującej plan treningowy dla mężczyzn został wykonany w sposób analogiczny. Aspektami różniącymi się dla danych sekcji mężczyzn i kobiet jest zawartość merytoryczna poszczególnych obiektów, natomiast od strony programistycznej zostały wykorzystane te same rozwiązania. Pozwala to na szybkie oraz zrozumiałe odtworzenie wykorzystanego rozwiązania w przyszłości w związku z zachowaniem tego samego modelu postępowania dla obu schematów.

Listing 4.11, przedstawiony poniżej, prezentuje dalszą część klasy planów treningowych dla kobiet.

```
1         newRecyclerView = findViewById(R.id.Recycler_viev_dla_kobiet_plany_treningowe)
2         newRecyclerView.layoutManager = LinearLayoutManager(this)
3         newRecyclerView.setHasFixedSize(true)
4
5         newArrayList = arrayListOf<Plan>()
6         gerUserdata()
7
8     private fun gerUserdata() {
9
10        for (i in imageId.indices){
11            val temp = Plan(imageId[i],Tytul[i])
12            newArrayList.add(temp)
13        }
14
15        var adapter = AdapterDlaKobietPlan(newArrayList)
16        newRecyclerView.adapter = adapter
17        adapter.setOnItemClickListener(object :AdapterDlaKobietPlan.OnItemClickListener{
18            override fun onItemClick(position: Int) {
19                val intent = Intent(this@Dla_Kobiet_Plany_Treningowe,Plan_Dla_Kobiet::class.java)
20                intent.putExtra("heading",newArrayList[position].heading)
21                intent.putExtra("Trening",Trening[position])
22                startActivity(intent)
23            }
24        })
25    }
```

Listing 4.11. Opis klasy planów treningowych dla kobiet.

Powyższy kod inicjuje RecyclerView, ustawia menadżera układu, który odpowiedzialny jest za pozycjonowanie obiektów w „RecyclerView” oraz nadaje mu stały rozmiar. Następnie zostaje zainicjowany nowy obiekt „ArrayList” typu Plan oraz wywołana zostaje metoda „getUserdata”, służąca za wypełnienie listę danymi zadeklarowanymi w listingu 4.10.

Dla każdego obiektu zostaje również stworzony kolejny obiekt „OnItemClickListener”, który nasłuchuje zdarzenia dla każdego obiektu. W przypadku kliknięcia na obiekt element rozpoczyna nową aktywność, indywidualną dla każdego elementu i przekazuje mu dane odpowiednie dla każdego elementu – tytuł oraz zawartość znajdującą się w liście o nazwie Trening. Nowa aktywność pobiera dane przekazane w linii 20 oraz 21 i przekazuje je poprzez swoją klasę do struktury wyglądu aplikacji, dzięki czemu każdy obiekt posiada indywidualną zawartość w zależności od wybranego uprzednio obiektu w liście RecyclerView. Dzięki takiemu rozwiązaniu nie ma potrzeby tworzenia osobnej aktywności dla każdego obiektu RecyclerView, następuje tylko zamiana informacji na jednej wspólnej aktywności.

Rys 4.6 Wygląd zawartości obiektów sekcji planów treningowych – z prawej strony dla mężczyzn, z lewej strony dla kobiet

Na rysunku 4.6, przedstawionym powyżej został zaprezentowany wygląd obiektów listy z sekcji planów treningowych. Aktywność obiektu składa się z tytułu reprezentującego nazwę danej sekcji oraz zawartość merytoryczną opisującą dane zagadnienie. W przypadku dłuższych opisów zastosowany został element „ScrollView", który umożliwia przewijanie zawartości w pionie lub poziomie w zależności od orientacji urządzenia. Jest to przydatny element w przypadku wyświetlania zawartości, których objętość nie mieści się na ekranie urządzenia. Bez zastosowania tego widżetu, zawartość, która nie mieści się w ekranie użytkownika zostaje ucięta i nie może zostać w żaden sposób wyświetlona podczas działania aplikacji.

Dla poprawy wyglądu i estetyki zastosowano różne style do edycji tekstu, które pomagają w aspekcie wizualnym prezentowanej zawartości.

```
class Plan_Dla_Kobiet : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_plan_dla_kobiet)

        val Tytul : TextView = findViewById(R.id.T_Plan_dla_kobiet_rodzaj_treningu)
        val Tresc : TextView = findViewById(R.id.Tresc_Plan_dla_kobiet_rodzaj_treningu)

        val bundle : Bundle?= intent.extras
        val heading = bundle!!.getString("heading")
        val Trening = bundle.getString("Trening")

        Tytul.text = heading
        Tresc.text = Trening
    }
}
```

Listing 4.12. Zawartość klasy pojedynczego obiektu sekcji planów treningowych.

W powyższym kodzie zdefiniowane zostało działanie aktywności odpowiedzialnej za wyświetlanie zawartości dla poszczególnego obiektu sekcji planów treningowych.

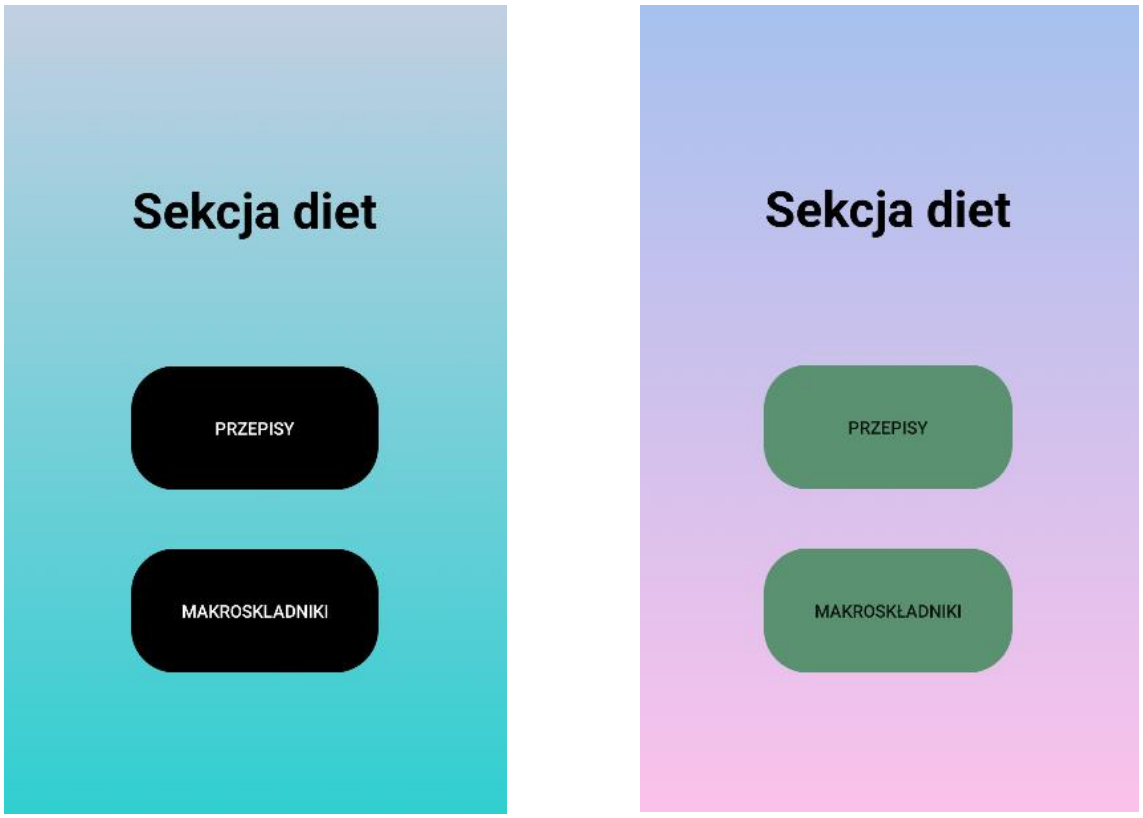
W klasie tej następuje tworzenie dwóch obiektów „TextView", które odpowiadają dwóm obiektom tego samego rodzaju, zdefiniowanym w pliku XML klasy, o zadanych identyfikatorach. Następnie następuje pobranie zawartości z klasy obsługującego listę obiektów dla sekcji planów treningowych i ich wyświetlenie dla odpowiedniego obiektu.

4.6. Sekcja diet

Sekcja diet została wykonana zgodnie ze schematem przedstawionym na rysunku 3.4. Sekcja ta została podzielona na dwie podsekcje, które zostaną omówione w poniższym podrozdziale.

Rozdział dotyczący przepisów został stworzony na zasadzie listy RecyclerView, która szczegółowo została omówiona w poprzednim podrozdziale dotyczącym planów treningowych. Dla sekcji diet został stworzony obiekt dla listy rozwijalnej RecyclerView, który za pomocą adaptera wyświetla tytuły przepisów wraz ze stosowną grafiką.

Na rysunku 4.7, przedstawionym poniżej zaprezentowana została grafika wykonana dla sekcji diet z podziałem na podkategorię przepisów oraz makroskładników.



Rys 4.7. Podział sekcji diet.

Zachowując ujednolicenie podział na sekcję diet został wykonany identycznie dla obu płci. Zastosowane zostały dwa przyciski przenoszące użytkownika do nowych niezależnych aktywności.

```
Przycisk_przepisy.setOnClickListener(){
    Toast.makeText(this,"Przepisy", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,Dla_Mezczyzn_Diety_Przepisy
    ::class.java)
    startActivity(intent)
}

Przycisk_zapo.setOnClickListener(){
    Toast.makeText(this,"Makroskładniki", Toast.LENGTH_SHORT).show()
    val intent = Intent(Intent.ACTION_VIEW, Uri.parse("https://www.maczfit.pl
    /blog/makroskladniki-czym-sa-jak-je-obliczyc/"))
    startActivity(intent)
}
```

Listing 4.13. Sposób implementacji sekcji diet.

Zgodnie z wyżej przedstawionym listingiem, przycisk z napisem przepisy odpowiada za przeniesienie użytkownika do listy zawierającej przepisy, natomiast przycisk makroskładniki przenosi użytkownika do strony internetowej, zawierającej informacje na temat makroskładników, jak zostało przedstawione na rys 4.8. [12].

Rys 4.8. Aktywność makroskładników.

Powyższa aktywność prezentuje podstawowe informacje na temat makroskładników. Przycisk przenosi użytkownika bezpośrednio do strony internetowej dlatego w tym przypadku użytkownik potrzebuje dostępu do Internetu.

Dla podsekcji przepisów została wykonana lista RecyclerView. Lista składa się z dziesięciu propozycji przepisowych z rozpisanymi makroskładnikami. Lista przepisów jest wspólna dla sekcji męskiej oraz żeńskiej. Nie ma potrzeby podziału w zależności od posiłku, w związku z tym, że różnica między mężczyznami, a kobietami jest w liczbie potrzebnych kcal, a nie w posiłkach w jakich jest dostarczana.

Wygląd listy przedstawiającej podsekcję przepisy w sekcji diet został zaprezentowany na rys 4.9, który został przedstawiony poniżej.

Rys 4.9. Wygląd sekcji przepisów. Po prawo – sekcja dla mężczyzn, po lewo – sekcja dla kobiet.

Każda z pozycji na liście, ma swój indywidualny tytuł oraz wspólną ikonę, kojarząca się z daną sekcją. Użytkownik po kliknięciu na dany obiekt zostaje przeniesiony do strony internetowej zawierającej dany przepis. W tym przypadku również użytkownik musi posiadać dostęp do zasobów Internetu. Zakładając jednak powszechną dostępność do tego medium zostało zastosowane tego rodzaju rozwiązanie. W przypadku braku Internetu użytkownik nadal może korzystać z aplikacji, jednak nie będzie dostępu do sekcji diet, ponieważ cała sekcja została oparta na tym samym rozwiązaniu – zarówno w sekcji dla kobiet jak i dla mężczyzn. Zakładając jednak, że użytkownik będzie korzystał z sekcji diet, głównie w domowej kuchni nie należy rozpatrywać dostępu do Internetu jako czynnika utrudniającego problem w korzystaniu, w związku z tym, że wiele popularnych aplikacji opartych jest na korzystaniu z zasobów Internetu.

```
imageId = arrayOf(
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24,
    R.drawable.baseline_food_bank_24
)

Tytul = arrayOf(
    "Makaron w pesto w stylu śródziemnomorskim",
    "Makaron spaghetti z parmezanem w pomidorach",
    "Makaron z łososiem wędzonym i szpinakiem",
    "Deser w stylu tiramisu z nieziemsko dobrym makro!",
    "Burger z buraka",
    "Jajecznica",
    "Owsianka",
    "Omlet",
    "Skyr waniliowy - kiedy nie ma w sklepie",
    "Wytrawna tarta z kurkami"
)
```

Listing 4.14. Implementacja obiektów sekcji przepisów.

Obiekty w sekcji przepisów składają się z ikony, która jest taka sama dla każdego obiektu, w związku z tym, że wszystkie obiekty dotyczą jedzenia, w przeciwieństwie do planu treningowego, gdzie każdy obiekt dotyczył innej partii mięśniowej. Dla każdego obiektu został nadany indywidualny tytuł dodany bezpośrednio w klasie odpowiadającej za przepisy. Nie ma potrzeby zawierania danych w pliku string.xml, w związku z faktem, że aktywność nie przekazuje z aplikacji żadnych dodatkowych informacji poza adresem URL(ang. Uniform Resource Locator - Jednolity lokalizator zasobów), do którego ma udać się dana aktywność.

Adapter jak również aktywność kliknięcia obiektu z nim związana zostały zaimplementowane zgodnie ze schematem, który został zaprezentowany dla adaptera stworzonego na potrzeby sekcji planów treningowych. Dotyczy to zarówno sekcji dla kobiet jak i dla mężczyzn.

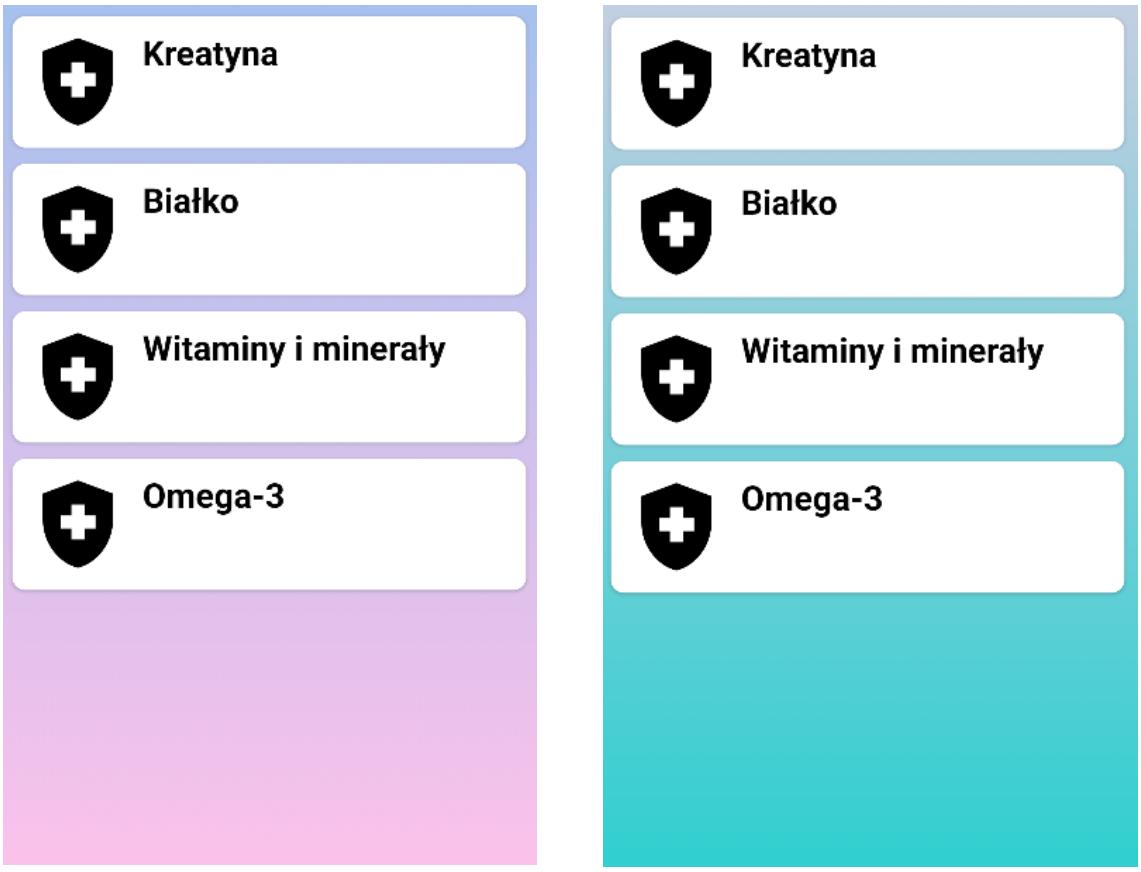
```
override fun onItemClick(position: Int) {
    if (position == 0 )
    {
        Toast.makeText(this@Dla_Kobiet_Diety_Przepisy, "Makaron w pesto w " +
        "stylu śródziemnomorskim", Toast.LENGTH_SHORT).show()
        val intent = Intent(Intent.ACTION_VIEW, Uri.parse("https://dieta-sportowca.pl" +
        "/2022/01/szybkie-przepisy-na-makaron-dla-sportowca/"))
        startActivity(intent)
    }
    if (position == 1 )
    {
    Toast.makeText(this@Dla_Kobiet_Diety_Przepisy, "Makaron spaghetti z" +
    " parmezanem w pomidorach", Toast.LENGTH_SHORT).show()
    val intent = Intent(Intent.ACTION_VIEW,Uri.parse("https://dieta-sportowca.pl" +
    "/2022/01/szybkie-przepisy-na-makaron-dla-sportowca/"))
    startActivity(intent)
    }
}
```

Listing 4.15. Implementacja aktywności dla obiektów w sekcji przepisy.

Zgodnie z algorytmem zamieszczonym w listingu 4.15, użytkownik po naciśnięciu na dany obiekt otrzymuje informację zgodną z tytułem na jaki obiekt kliknął oraz zostaje przekierowany zgodnie z zapisem linii 6-7 oraz 14-15, do nowej aktywności, która jest w rzeczywistości stroną internetową zawierająca informacje na wybrany temat. W celu identyfikacji poszczególnych obiektów zastosowano pozycjonowanie względem którego od góry liczone są kolejne elementy. Przykładowo, naciskając na pierwszy element z listy wybieralnej RecyclerView w sekcji przepisów, aktywowany zostaje warunek dla którego parametr position (ang.position – pozycja) przyjmuje wartość zero zgodnie z przyjętą indeksacją. Dzieje się tak dla pozostałych 9 zdarzeń w sposób analogiczny.

4.7. Sekcja suplementacji

Sekcja suplementacji zawiera podstawowe informacje na temat podstawowych suplementów diety dostępnych w sportach siłowych. W przeglądzie suplementów w ramach aplikacji mobilnej trenera personalnego skupiono się wyłącznie na przebadanych i sprawdzonych środkach, ponieważ nie wszystkie substancje używane w sportach siłowych mają tak szeroki zakres badań jak te, omawiane w ramach pracy dyplomowej. Dlatego na potrzeby pracy dyplomowej zostały wyselekcjonowane cztery podstawowe suplementy diety : kreatyna, białko, kwasy omega-3 oraz witaminy i minerały, które w zupełności wystarczą podczas swojego początku w przygodzie ze sportami siłowymi.

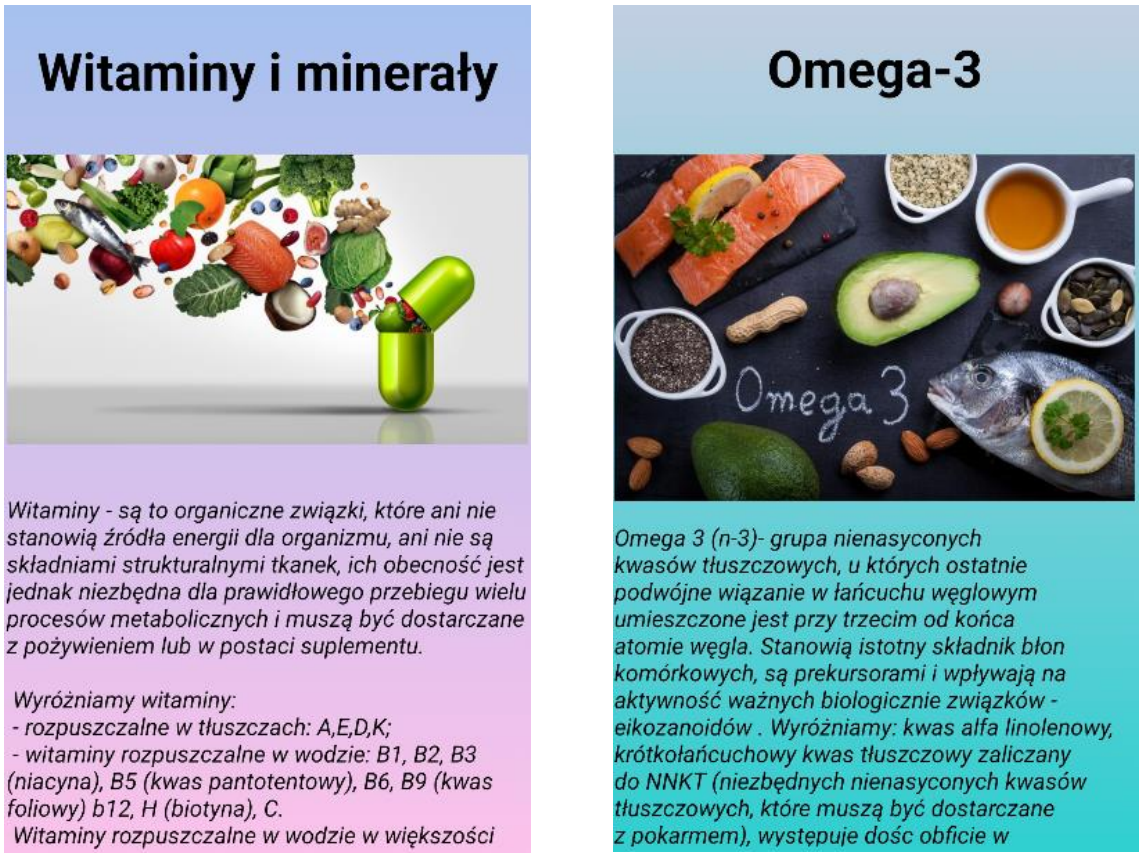


Rys 4.10. Wygląd sekcji suplementacji.

Jak już zostało wcześniej przedstawione sekcja składa się z czterech obiektów. Każdy obiekt jest niezależny od drugiego i przedstawia szczegółowe informacje na temat danego suplementu zgodnie z tytułem. Obiekt składa się z tytułu oraz ikony stosownej do omawianego zagadnienia.

Po naciśnięciu na dowolny obiekt znajdujący się w sekcji poświęconej suplementacji użytkownik zostaje przeniesiony do nowej aktywności, w której szczegółowo został omówiony dany problem. Aktywność składa się z tytuły, obrazu tematycznego towarzyszącemu tytułowi oraz treści.

Wygląd graficzny przykładowego obiektu w sekcji suplementacji został przedstawiony poniżej na rysunku 4.11. Po lewo znajduje się obiekt witamin i minerałów w sekcji dla kobiet, natomiast po prawo – obiekt kwasów omega 3 dla mężczyzn.



Rys 4.11. Wygląd obiektów sekcji suplementacji.

Zarówno w sekcji dla mężczyzn, jak i sekcji dla kobiet znajdują się te same informacje dotyczące omawianych suplementów diety. Nie ma rozgraniczenia płciowego między sposobem przyjmowania danych suplementów, istotne jest jedynie aby nie stosować ich jako zamienniki do odpowiednio zbilansowanej diety, a tylko jako dodatki, które pomogą nam przybliżyć się do zamierzonego celu.

Obiekty klasy reprezentujące sekcję suplementacji zostały przedstawione w listingu 16, przedstawionym poniżej. W klasie przekazywane są trzy parametry. Parametr tytuł jest zadeklarowany bezpośrednio w danej klasie i jest odpowiedzialny za wyświetlenie tytuły na obiekcie i w aktywności obsługującej dany obiekt. Parametr opis służy do wyświetlania opisu danego obiektu, natomiast parametr zdjęcie służy do wyświetlania obrazu między tytułem, a treścią w danym obiekcie.

```
Tytul = arrayOf(
    "Kreatyna",
    "Białko",
    "Witaminy i minerały",
    "Omega-3"
)

Opis = arrayOf(
    getString(R.string.kreatyna_opis),
    getString(R.string.bialko_opis),
    getString(R.string.Witaminy_opis),
    getString(R.string.Omega_opis)
)

Zdjecie = arrayOf(
    R.drawable.krea_foto_lepsze,
    R.drawable.bialko_foto,
    R.drawable.witaminy_foto,
    R.drawable.omega_foto
)
```

Listing 4.16. Obiekty dla sekcji suplementacji.

Powyższe opisy zostały umieszczone w pliku string.xml i wyglądają zgodnie ze schematem przedstawionym w listingu 4.9. Zdjęcia zostały umieszczone w folderze pomocniczym drawable, w którym znajduje się grafika użyta na potrzeby aplikacji mobilnej trenera personalnego.

```
adapter.setOnItemClickListener(object :AdapterDlaKobietSuplementy.OnItemClickListener{
override fun onItemClick(position: Int) {
    val intent = Intent(this@Dla_Kobiet_Suplementacja,Suplementacja_Dla_Kobiet::class.java)
        intent.putExtra("Tytul",newArrayList[position].text1)
        intent.putExtra("ZdjecieId",Zdjecie[position])
        intent.putExtra("Opis",Opis[position])
        startActivity(intent)
    }

})
```

Listing 4.17. Przekazanie obiektów do nowej aktywności dla sekcji suplementacji.

W listingu 4.17 przedstawionym powyżej został zaprezentowany sposób przesłania parametrów do nowej aktywności obsługującej wyświetlanie zawartości obiektów. W porównaniu do listingu 4.11, na którym również zostało przedstawione przekazanie parametrów została dodana dodatkowa instrukcja „insert.putExtra”, odpowiadająca za przekazanie zdjęcia zawartego w tej klasie do nowej aktywności. Parametr position informuje nas, że zdjęcia przesyłane są w kolejności indeksowanej od zera, tak jak zostało to omówione we wcześniejszych przykładach.

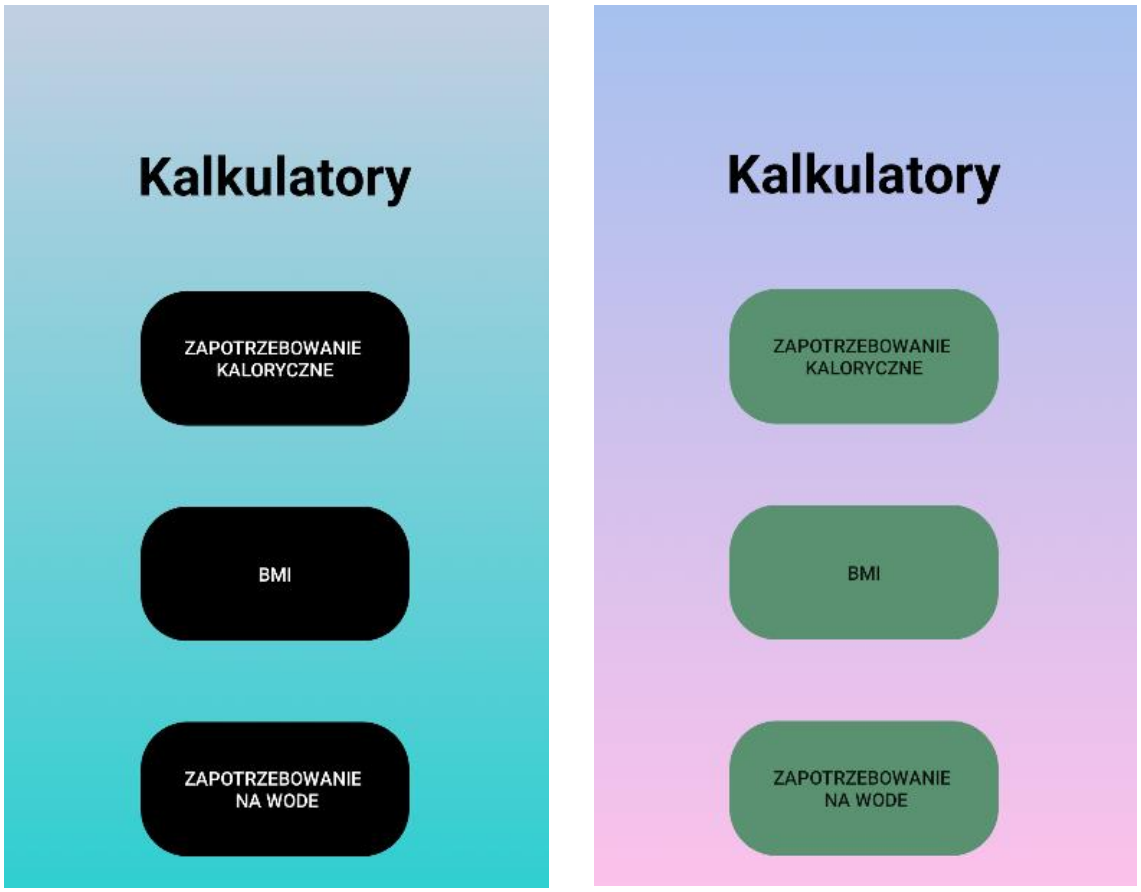
```
1  class Suplementacja_Dla_Kobiet : AppCompatActivity() {
2      override fun onCreate(savedInstanceState: Bundle?) {
3          super.onCreate(savedInstanceState)
4          setContentView(R.layout.activity_suplementacja_dla_kobiet)
5
6          val Tytul : TextView = findViewById(R.id.Tekst_suplementacja_layout)
7          val Opis : TextView = findViewById(R.id.Tres_suple_dla_kobiet)
8          val Zdjecie : ImageView = findViewById(R.id.Suplementacja_dla_kobiet_zdjecie)
9
10         val bundle : Bundle?=intent.extras
11         val Tytul2 = bundle!!.getString("Tytul")
12         val Opis2 = bundle.getString("Opis")
13         val Zdjecie2 = bundle.getInt("ZdjecieId")
14
15
16         Tytul.text = Tytul2
17         Opis.text = Opis2
18         Zdjecie.setImageResource(Zdjecie2)
19 }
```

Listing 4.18. Klasa obsługująca aktywność obiektów w sekcji suplementacji.

Powyższy listing przedstawia sposób przekazania parametrów z klasy poświęconej suplementacji do klasy obsługującej aktywność pojedynczego obiektu. W liniach 6-8 następuje deklaracja zmiennych, które odpowiadają obiektom umieszczonym w layoucie graficznym obsługującym daną aktywność. Następnie następuje przekazanie parametrów z klasy głównej suplementacji do klasy pojedynczego obiektu zgodnie z liniami 10-13. Ostatecznie zmienne obiektów zostają przypisane do parametrów z klasy suplementacji i przekazywane do layoutu graficznego w celu ich wyświetlenia.

4.8. Sekcja kalkulatorów sportowych

Sekcja kalkulatorów sportowych omawiana w ramach pracy dyplomowej została wykonana zgodnie ze schematem przedstawionym na rysunku 3.6. Sekcja będzie posiadała trzy podsekcje odpowiadające za poszczególne kalkulatory służące do obliczeń sportowych. Każda z sekcji jest niezależna i odpowiada za inną formę obliczeń. Dla każdego kalkulatora zastosowano powszechnie używane wzory zgodnie z którymi wyliczane są dane wartości, o których mowa była w rozdziale 3.6 poświęconym kalkulatorom sportowym. Zaprezentowane w tym rozdziale zależności matematyczne oraz zakresowe zostały zaimplementowane w aplikacji w celu możliwości komfortowej obsługi bez potrzeby znajomości wyżej omawianych równań matematycznych.



Rys 4.12. Sekcja kalkulatorów sportowych

Na rysunku 4.12, przedstawionym powyżej została zaprezentowana grafika sekcji kalkulatorów sportowych. Aktywność wyposażona jest w trzy przyciski, które przenoszą użytkownika do nowej aktywności zawierającej kalkulator z nazwą przedstawioną na przycisku.

```
calc_zapotrzebowanie.setOnClickListener(){
    Toast.makeText(this,"Kalkulator zapotrzebowania kalorycznego", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,Kalkulator_zapotrzebowanie_mezczyzni::class.java)
    startActivity(intent)
}

calc_BMI.setOnClickListener(){
    Toast.makeText(this,"Kalkulator BMI", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,KalkulatorBMI_m::class.java)
    startActivity(intent)
}

calc_woda.setOnClickListener(){
    Toast.makeText(this,"Kalkulator zapotrzebowania na wodę", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,Kalkulator_zapotrzebowanie_na_wode_m::class.java)
    startActivity(intent)
}
```

Listing 4.19. Implementacja przycisków sekcji kalkulatorów sportowych

Powyższy listing 4.19, przedstawia implementację przycisków z rysunku 4.12. Po naciśnięciu na przycisk użytkownik otrzymuje informację w postaci wyskakującego okna o sekcji do której został przeniesiony. Linie kodu z powyższego listingu : 3-4, 9-10 oraz 15-16 odpowiadają za przeniesienie do nowej aktywności obsługującej dany kalkulator.

4.8.1. *Kalkulator zapotrzebowania kalorycznego*

Kalkulator zapotrzebowania kalorycznego jest podstawowym kalkulatorem podczas omawiania zagadnień związanych ze sportami siłowymi. Odpowiednia podaż kalorii ma kluczowe znaczenie do rozrostu mięśni oraz ogólnego samopoczucia, które prowadzi do poprawy wyników na siłowni.

Rys 4.13. Grafika kalkulatorów zapotrzebowania kalorycznego.

Na rysunku 4.13, przedstawionym powyżej została zaprezentowana postać graficzna aktywności obsługującej kalkulator zapotrzebowania kalorycznego. Po lewo została przedstawiona wersja dla kobiet, natomiast po prawo – dla mężczyzn. Użytkownik jest proszony o podanie wagi w kilogramach, wzrostu w centymetrach oraz wieku w latach na którego podstawie zgodnie z zależnościami (1.1) oraz (1.2) w zależności od płci zostaje obliczone szacunkowe zapotrzebowanie kaloryczne. Wartość ta jest wyłącznie wartością przybliżoną i służy w celach orientacyjnych.

```
1  ButtonOblicz.setOnClickListener(){
2      val Wzrost = EditTextWzrost.text.toString()
3      val Waga = EditTextWaga.text.toString()
4      val Wiek = EditTextWiek.text.toString()
5
6      if(PustePole(Wzrost, Waga, Wiek))
7      {
8          val zapotrzebowanie = 10 * Waga.toFloat() + 6.25 *
9          Wzrost.toFloat() - 5 * Wiek.toFloat() - 161
10
11         val zaporzebowanie2Digits = String.format("%.2f",zapotrzebowanie).toFloat()
12
13         TextViewWyswietl_liczba.text = zaporzebowanie2Digits.toString()
14     }
15 }
```

Listing 4.20. Implementacja kalkulatora zapotrzebowania kalorycznego dla kobiet.

Na listingu 4.20, przedstawionym powyżej została zaprezentowana implementacja kalkulatora zapotrzebowania kalorycznego. Po naciśnięciu przycisku z napisem oblicz zostaje wykonana instrukcja warunkowa w linii 6. Warunek ten jest spełniony tylko w przypadku wypełnienia wszystkich pól wymaganych do wykonania obliczenia, co zostanie omówione w rozdziale 5 pracy. W instrukcji warunkowej zostaje wykonana formuła obliczeniowa dla kobiet zgodnie z wyrażeniem (1.2). Następnie wynik zostaje wyświetlony w postaci tekstu na ekranie użytkownika.

Listing 4.20 przedstawia sposób implementacji kalkulatora dla kobiet. Implementacja rozwiązania dla mężczyzn została wykonana w sposób analogiczny z uwzględnieniem zależności (1.1), która odpowiada zapotrzebowaniu dla mężczyzn.

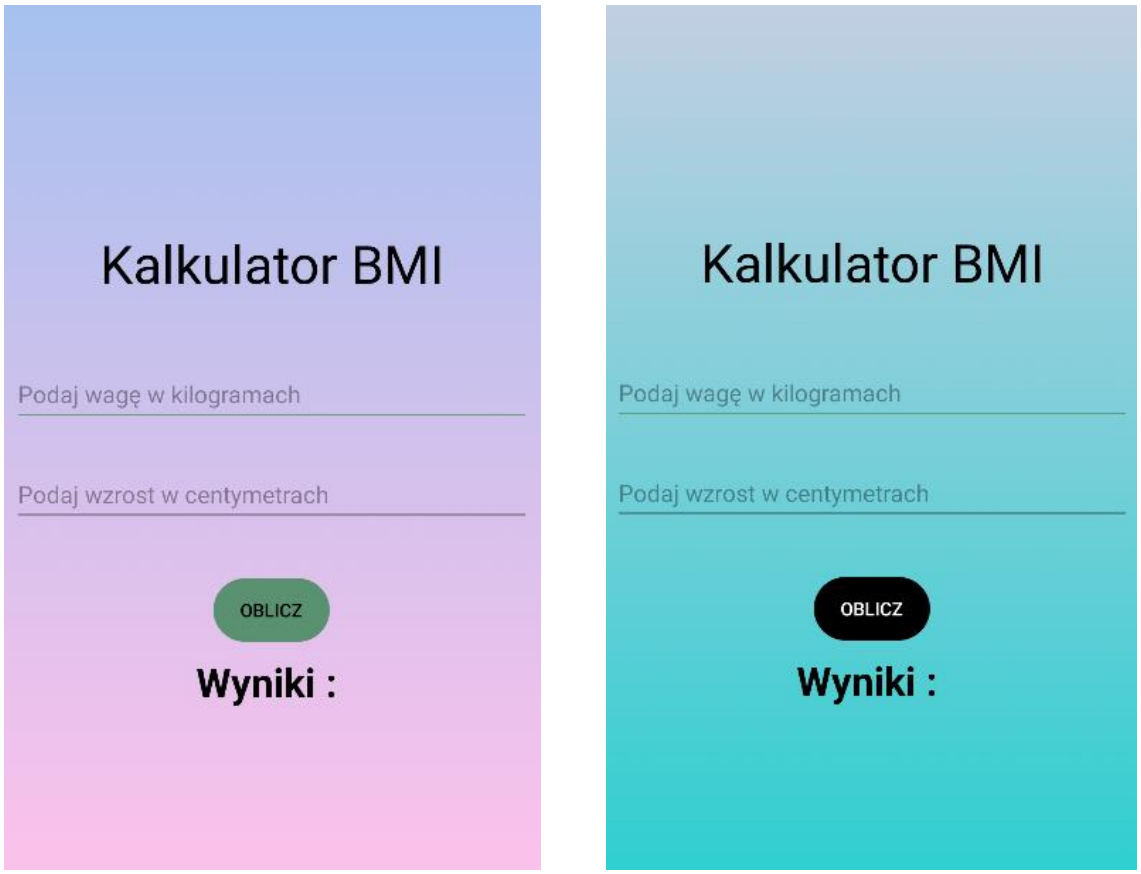
Na rysunku 4.14, przedstawionym poniżej zostało zaprezentowane działanie aplikacji dla przykładowych wartości. Aplikacja na podstawie formuł (1.1) oraz (1.2) wylicza zapotrzebowanie kaloryczne, które jest szacunkowym minimalnym zapotrzebowaniem dla danej płci. Należy wziąć pod uwagę, że w przypadku osoby trenującej należy zwiększyć podaż kaloryczną i nie można w takich sytuacjach sugerować się aplikacją. Po wpisaniu wartości oraz kliknięcia przycisku mamy wyświetlany wynik na podstawie zadanych wartości.

Rys. 4.14. Przykładowe działanie kalkulatora zapotrzebowania kalorycznego.

4.8.2. *Kalkulator BMI*

Kolejnym z kalkulatorów omawianych w ramach sekcji będzie kalkulator BMI służący do obliczenia wskaźnika masy ciała. Kalkulator BMI jest jednym z najpopularniejszych kalkulatorów sportowych dlatego dla wygody oraz zainteresowania użytkownika został on umieszczony w aplikacji. Po podaniu swojego wzrostu w centymetrach oraz swojej wagi w kilogramach użytkownik zgodnie z zakresem wskaźnika BMI, który został przedstawiony na rysunku 12 otrzymuje liczbę mieszczącą się w jednym z tych przedziałów.

Do obliczeń dla kalkulatora BMI posłużono się powszechnie znanym wyrażeniem (1.3), który jest ilorazem masy przez kwadrat wzrostu danej osoby. Zgodnie z wcześniej przytoczonym wzorem wzrost we wzorze brany jest w metrach, natomiast użytkownik podaje swój wzrost w centymetrach. Zostało to zrobione ze względu na powszechność podawania wzrostu w tej jednostce, a do obliczeń algorytm przelicza i zamienia jednostki na odpowiednie do wzoru.



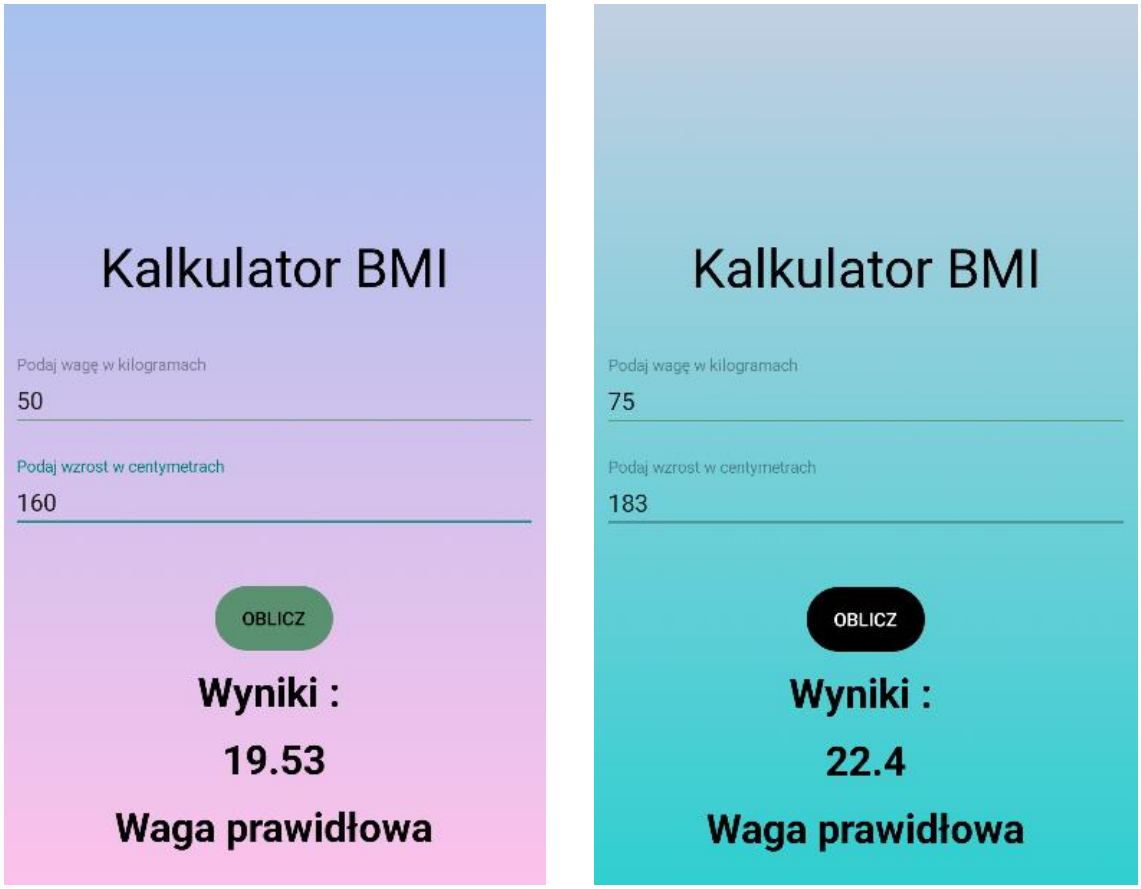
Rys 4.15. Wygląd graficzny kalkulatora BMI.

Na rysunku 4.15, przedstawionym powyżej został przedstawiony wygląd graficzny kalkulatora BMI dla sekcji kobiet oraz mężczyzn. Kalkulator posiada dwa pola tekstowe możliwe do wypełnienia przez użytkownika oraz jeden przycisk, który wykonuje obliczenia tylko i wyłącznie po uzupełnieniu wszystkich pól. Wynik wyświetla się w postaci liczbowej oraz słownej jako wynik równania oraz zakres z rysunku 12 w którym znajduje się użytkownik dla podanych wartości wagi oraz wzrostu.

```
1  ButtonOblicz.setOnClickListener(){
2      val Wzrost = EditTextWzrost.text.toString()
3      val Waga = EditTextWaga.text.toString()
4
5      if(PustePole(Wzrost,Waga))
6      {
7          val bmi = Waga.toFloat()/((Wzrost.toFloat()/100)*(Wzrost.toFloat()/100))
8          val bmi2Digirs = String.format("%.2f",bmi).toFloat()
9          Wystietl(bmi2Digirs)
10     }
11 }
```

Listing 4.21. Implementacja kalkulatora BMI.

Powyższy listing 4.21, przedstawia implementacje kalkulatora BMI. Linie 2-3 odpowiadają identyfikatorom dla layoutu graficznego który reprezentuje wygląd graficzny. Warunek w linii 5 wykonuje się tylko i wyłącznie kiedy oba pola są wypełnione. W tym przypadku następuje obliczenie wartości BMI w linii 7 zgodnie z zależnością (1.3) i jej wyświetlenie z dwoma miejscami po przecinku zgodnie z linią 8.



Rys 4.16 Przykład obliczeń kalkulatora BMI.

Na rysunku 4.16 przedstawionym powyżej zostało zobrazowane przykładowe działanie kalkulatora. Jak przykład została wzięta 50 kilogramowa kobieta o wzroście 160 centymetrów oraz 75 kilogramowy mężczyzna o wzroście 183 centymetrów. Wyniki obliczeń zostały zobrazowane w postaci liczby oraz zgodnie z przyjętym zakresem została również wyświetlona stosowna wiadomość informująca użytkownika w jakim zakresie się znajduje. Dzięki temu nie ma potrzeby sprawdzania indywidualnego przez użytkownika czy jego waga jest prawidłowa co działa na korzyść aplikacji ze względu na zwiększoną funkcjonalność.

```
1     private fun Wystietl (bmi: Float){
2
3         TextViewWyswietl_liczba.text = bmi.toString()
4
5         var resulttext = ""
6
7         when{
8             bmi<16.00 -> {
9                 resulttext = "Wygłodzenie"
10            }
11            bmi in 16.00 .. 16.99 -> {
12                resulttext = "Wychudzenie"
13            }
14        }
15        TextViewWyswietl_slowo.text = resulttext
16    }
```

Listing 4.22. Warunki dla kalkulatora BMI.

W listingu 4.22 przedstawionym powyżej został przedstawiony fragment implementacji przedziałów BMI zgodnych z rysunkiem 3.7. Metoda przyjmuje parametr bmi, który jest liczbą obliczoną według wcześniej wspomnianego równania. Parametr ten w zostaje sprawdzany w instrukcji „when" i przypisywany do przedziału w którym się zawiera. Jeżeli liczba należy do danego przedziału zmienna „resluttext" przyjmuje ciąg znaków zgodny z danym przedziałem liczbowym jak zostało to przedstawione w liniach 9 oraz 12 i zostaje przekazana do wyjścia po spełnieniu się wszystkich warunków.

4.8.3. *Kalkulator zapotrzebowania na wodę*

Woda jest składnikiem tkanek i płynów ustrojowych, które transportują w organizmie tlen oraz składniki odżywcze. Pomaga nam w regulacji temperatury ciała, procesie trawienia oraz wydalania produktów przemiany materii. Pełni również istotną funkcję ochronną dla mózgu [13]. Dlatego istotne jest aby dostarczyć odpowiednią ilość wody do organizmu.

Kalkulator zapotrzebowania na wodę powstał w celach informacyjnych, ale również aby uświadomić użytkownika ile tak naprawdę płynów należy dostarczyć w ciągu dnia w celu prawidłowego funkcjonowania organizmu.

Rys 4.17. Wygląd graficzny kalkulatora zapotrzebowania na wodę.

Na rys 4.17, który został przedstawiony powyżej, znajduje się wygląd graficzny kalkulatora zapotrzebowania na wodę. Kalkulator przyjmuje od użytkownika masę ciała i na jej podstawie zgodnie z zależnością (1.4) oblicza szacunkowe zapotrzebowanie na wodę. Wynik wyświetlający się użytkownikowi zostaje podany w mililitrach.

```
ButtonOblicz.setOnClickListener(){
    val Waga = EditTextWaga.text.toString()

    if(PustePole(Waga)) {
        val woda = 30 * Waga.toFloat()

        val zaporzebowanie2Digits = String.format("%.2f", woda).toFloat()
        TextViewWyswietl_liczba.text = zaporzebowanie2Digits.toString()
    }
}
```

Listing 4.23. Implementacja kalkulatora zapotrzebowania na wodę.

Listing 4.23 przedstawia implementację kalkulatora zapotrzebowania na wodę. Po naciśnięciu przycisku przez użytkownika kalkulator sprawdza warunek w linii 4. Funkcja „PustePole” sprawdza, czy pole do wpisania wagi zostało wypełnione i zostanie omówiona w 5 rozdziale pracy dyplomowej. Jeżeli warunek ten zostanie spełniony następuje obliczenie zapotrzebowania zgodnie z zależnością (1.4), co zostało przedstawione w linii 5 algorytmu. Następne dwie instrukcje to formatowanie wyniku do dwóch miejsc po przecinku oraz jego wyświetlenie na ekranie smartphone.

Rys 4.18. Przedstawienie działania kalkulatora zapotrzebowania na wodę.

Na rys 4.18. zostało przedstawione działanie kalkulatora na wodę. Analizując formułę (1.4) zauważyć można, że nie ma tutaj przypadku rozgraniczenia na płeć, a jedynie wartość ta zależy od masy ciała danej osoby. W powyższym przykładzie kobieta o wadze 60 kilogramów powinna wypić 1800 ml. płynów dziennie, natomiast mężczyzna o wadze 75 kilogramów około 2250 ml. Należy pamiętać, że wartości obliczone przez kalkulator są szacunkowe.

4.9. Sekcja wiadomości

Sekcja wiadomości powstała w celu dostarczenia użytkownikowi informacji ze świata sportów siłowych. Użytkownik w tej sekcji będzie mógł wybrać interesującą go informację z listy oraz przeczytać treść wybranego artykułu.

Rys 4.19. Wygląd listy sekcji wiadomości.

Na rysunku 4.19. przedstawionym powyżej znajduje się wygląd graficzny listy RecyclerView sekcji wiadomości. Po lewej stronie rys 4.19. został przedstawiony wygląd sekcji dla kobiet, natomiast po prawej wygląd sekcji dla mężczyzn.

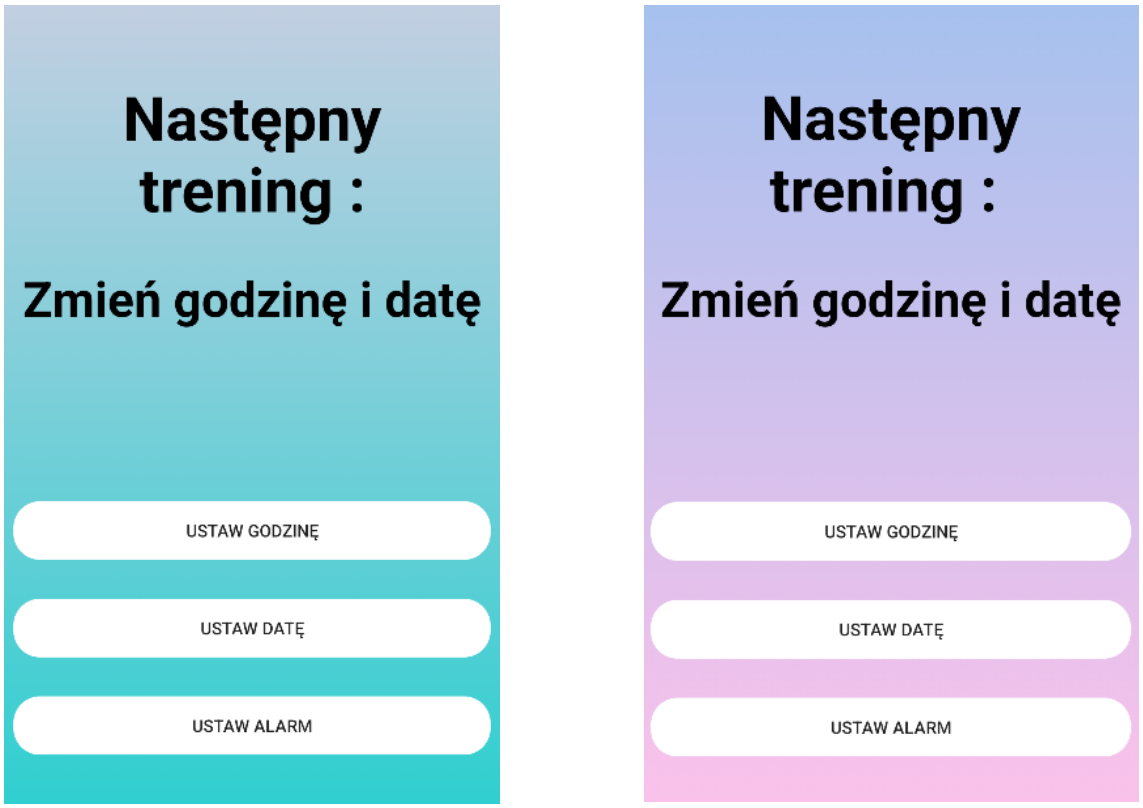
Użytkownik może wybrać z dostępnych 8 obiektów, które po kliknięciu przeniosą go do nowej aktywności. Obiekt posiada własną ikonę adektwaną do sekcji oraz tytuł informujący użytkownika o treści artykułu [14]

Rys 4.20. Przykładowy wygląd obiektów sekcji wiadomości.

Na rys. 4.20. został przedstawiony przykładowy wygląd przykładowych obiektów dostępnych w sekcji wiadomości. Sekcja wiadomości została wykonana na zasadzie RecyclerView, który został dokładnie omówiony w rozdziale 4.5. Nowa aktywność obiektu wybranego przez użytkownika składa się z tytułu zgodnego z nazwą wybranego obiektu, zdjęcia tematycznego oraz treści artykułu. Wykonanie aktywności przedstawionej na rys 4.20 zostało przedstawione w listingach 4.16, 4.17 oraz 4.18 i wykonane dla sekcji wiadomości w sposób analogiczny.

4.10. Sekcja dzień treningowy

Sekcja dzień treningowy powstała z myślą o organizacji treningu. Sekcja ta działa na zasadzie powszechnie znanego budzika i polega na przypomnieniu użytkownikowi o tym, że zbliża się jego trening. Aplikacje treningowe dostępne na rynku zostały wyposażone w możliwość ustawiania alarmów, dlatego w celu kompletności oraz dostosowania się do powszechnych praktyk, została zaimplementowana funkcja budzika.



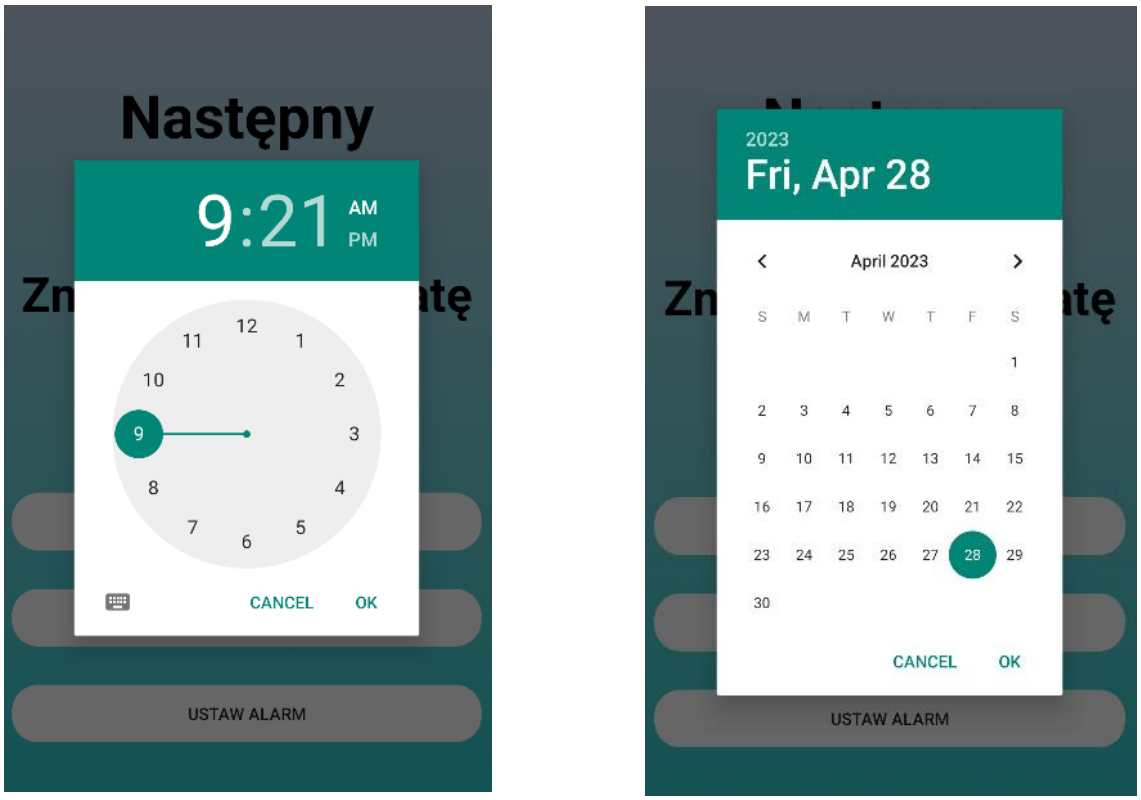
Rys 4.21. Wygląd graficzny sekcji dzień treningowy. Po lewo – sekcja dla mężczyzn, po prawo – sekcja dla kobiet.

Po wybraniu sekcji „dzień treningowy" z panelu głównego aplikacji użytkownik zostaje przeniesiony do sekcji, której wygląd graficzny został przedstawiony na rys. 4.21. Sekcja wyposażona jest w tytuł, informację o tym, że należy zmienić datę oraz godzinę a także trzy funkcjonalne przyciski. Odpowiadają one za ustawianie za godziny i daty treningu oraz alarmu.

Przycisk ustaw godzinę pozwala na wybranie godziny na zegarze z podziałem na czas liczony do południa (ang."Ante Meridem" – przed południem) oraz czas popołudniowy (ang. „Post Meridem" – po południu).

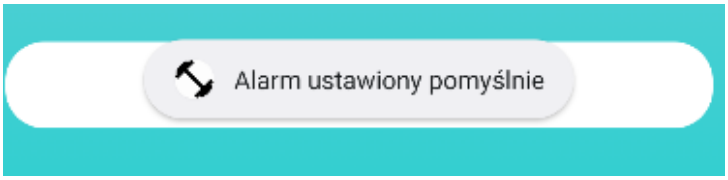
Przycisk ustaw datę pozwala na wybranie dogodnej daty z kalendarza.

Wygląd graficzny przycisków ustawiania daty oraz godziny został przedstawiony poniżej na rys 4.21. Po lewej stronie znajduje się ustawianie godziny treningu, natomiast po prawej stronie ustawianie daty.



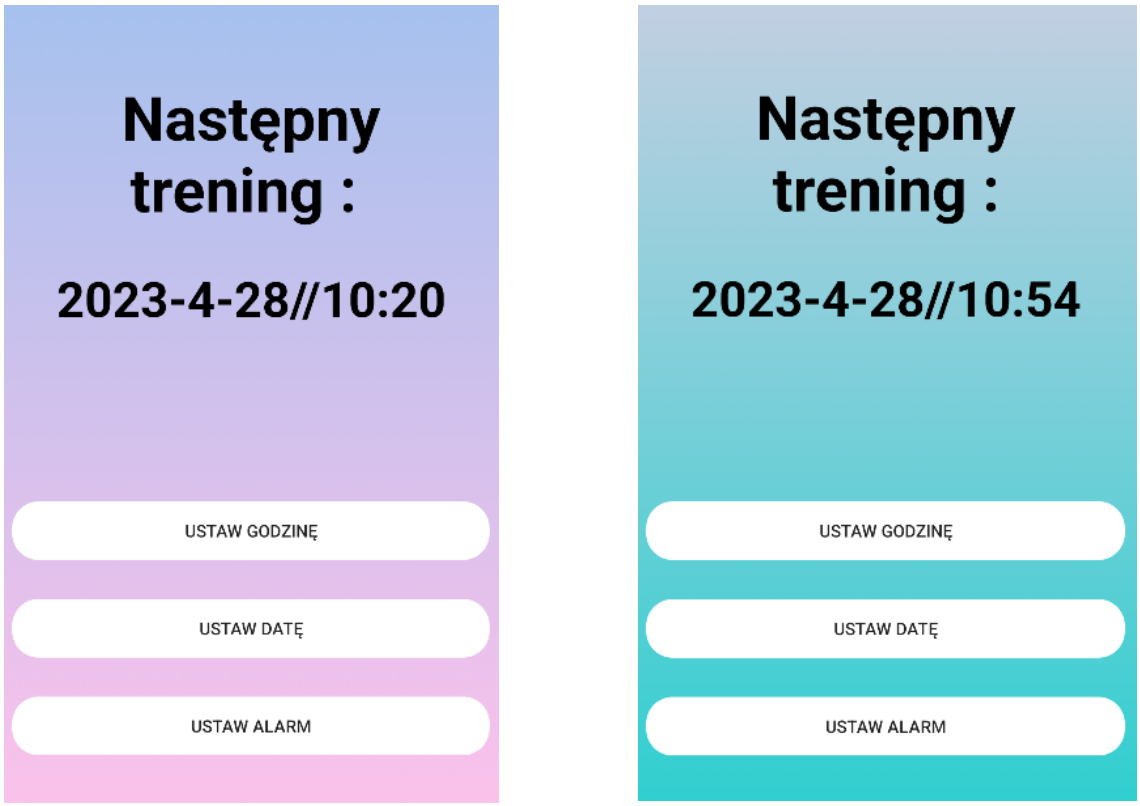
Rys 4.21. Wygląd graficzny ustawiania daty oraz godziny dla sekcji dnia treningowego.

Przycisk ustaw alarm służy do ustawiania alarmu po uprzednio wybranej godzinie oraz dacie. Pomyślne ustawienie daty oraz godziny skutkuje wyświetleniem komunikatu informującego o tej operacji jak zostało to przedstawione na rys 4.22.



Rys 4.22. Wygląd powiadomienia o pomyślnym ustawieniu alarmu.

Ustawienie alarmu skutkuje ustawieniem daty treningu w miejsce napisu „Zmień datę i godzinę” na wybrane przez użytkownika w formacie „rok-miesiąc-dzień // godzina::minuta”. Wygląd graficzny ustawionej sekcji dzień treningowy został przedstawiony na rys. 4.23, znajdującym się poniżej,



Rys 4.23. Ustawienie daty i godziny w sekcji „dzień treningowy”

Powyższy rysunek przedstawia grafikę użytkownika z poprawnie ustawionym alarmem. Użytkownik może swobodnie korzystać z pozostałych funkcjonalności aplikacji, a alarm będzie czuwał nad tym, żeby trening został wykonany o czasie.

Wygląd powiadomienia o treningu został przedstawiony na rys 4.24. O ustalonej godzinie użytkownik otrzymuje powiadomienie w postaci wyskakującego okna razem oraz dźwięku systemowego, tak samo jak w przypadku alarmu z budzika. Na powiadomienie składa się nazwa aplikacji oraz informacja „czas na trening”, która przypomina użytkownikowi o tym, że powinien wykonać swoje ćwiczenia.

Okno można zamknąć w poprzez przesunięcie go w lewo jak w każdej dostępnej aplikacji. Powiadomienie o treningu dostępne jest również w oknie zarządzania powiadomieniami dla aplikacji Android. Okno zarządzania powiadomieniami dostępne jest poprzez przesunięcie palca z góry ekranu na dół.

Rys 4.24. Wygląd powiadomienia w sekcji „dzień treningowy”

Zegar sekcji „dzień treningowy” obsługiwany jest bez potrzeby uruchamiania aplikacji w tle. Użytkownik może w pełni zamknąć aplikację zwalniając tym samym zasoby pamięci i może być pewny, że zegar powiadomi go o treningu w ustalonym czasie.

Sekcja została wyposażona wyłącznie w jeden zegar. Ustawienie dwóch zegarów spowoduje zmianę daty na ostatni ustawiony zegar. Jeżeli użytkownik ma dwa treningi jednego dnia sugerowane jest ustawienie zegara na pierwszy trening, a po odbyciu się pierwszego treningu na drugi.

Zaimplementowany zegar jest wspólny dla obu sekcji. Aplikacja mobilna trenera personalnego jest aplikacją indywidualną do użytku jednoosobowego, dlatego zostało przyjęte założenie, że dana osoba będzie korzystała z sekcji „dnia treningowego” adekwatnie do swojej płci.

```
1  class MyTimePickerDialog:DialogFragment() {
2
3      private val c = Calendar.getInstance()
4      private val hour = c.get(Calendar.HOUR_OF_DAY)
5      private val minute = c.get((Calendar.MINUTE))
6
7      override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
8          return TimePickerDialog(requireActivity(), activity as OnTimeSetListener,
9          hour, minute, DateFormat.is24HourFormat(activity))
10     }
11 }
```

Listing 4.24. Klasa obsługująca ustawienie godziny w alarmie.

Powyższy listing definiuje klasę o nazwie „MyTimePickerDialog", której celem jest stworzenie okna dialogowego selektora czasu. W klasie tej został zainicjonowany kalendarz oraz wartości bieżącej godziny oraz minuty. Metoda „OnCreateDialog" zwraca obiekt, który pomaga w obsłudze zdarzenia ustawienia czasu z bieżącą datą i godziną. Instrukcja „DateFormat.is24HourFormat" powoduje, że urządzenie użytkownika informuje czy obsługuje ono 24-godzinny format czasu, czy nie. W związku z używaniem wirtualnego urządzenia dostępnego w środowisku Android Studio, w przykładowym zrzucie na rys 4.21. jest obsługiwany 12-godzinny format.

```
1  class MyDatePickerDialog: DialogFragment() {
2
3      private val c = Calendar.getInstance()
4      private val year = c.get((Calendar.YEAR))
5      private val month = c.get(Calendar.MONTH)
6      private val day = c.get(Calendar.DAY_OF_MONTH)
7
8      override fun onCreateDialog(savedInstanceState: Bundle?):
9      Dialog {
10         return DatePickerDialog(requireActivity(),
11         activity as OnDateSetListener, year, month, day)
12     }
13 }
```

Listing 4.25. Klasa obsługująca ustawianie dnia w alarmie.

Powyższy listing definiuje klasę „MyDatePickerDialog", która umożliwia użytkownikowi wyświetlanie okna dialogowego odpowiedzialnego za wybranie daty. Podobnie jak w klasie służącej za ustawianie godziny, została zadeklarowana zmienna odpowiadająca za tworzenie kalendarza. Kolejne zmienne w liniach 4-6 odpowiadają za pobranie kolejno : roku, miesiąca i dnia. Analogicznie do przykładu z listingu 4.24 zostaje zwracany obiekt o nazwie „DatePickerDialog", który pozwala na wybranie dnia, miesiąca oraz roku przez użytkownika.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_dla_mezczyzn_dzien_treningowy)

    alarmManager = getSystemService(Context.ALARM_SERVICE) as AlarmManager
    alarmIntent = PendingIntent.getBroadcast(applicationContext,0,
        Intent(applicationContext,AlarmReceiver::class.java),
        PendingIntent.FLAG_MUTABLE)

    TimeSetDate = findViewById(R.id.time_set_date_m)
    TimeSetHour = findViewById(R.id.time_set_hour_m)
    TimeSetAlarm = findViewById(R.id.time_set_alarm_m)
    TEXTTIME = findViewById(R.id.textTIME_m)

    TimeSetHour.setOnClickListener {
        val dialog = MyTimePickerDialog()
        dialog.show(supportFragmentManager,"time_picker")
    }
    TimeSetDate.setOnClickListener {
        val dialog = MyDatePickerDialog()
        dialog.show(supportFragmentManager,"date_picker")
    }
    TimeSetAlarm.setOnClickListener {


        val date = Calendar.Builder()
            .setDate(year, month, dayOfMonth)
            .setTimeOfDay(hour,minute,0)
            .build()

        alarmManager.set(AlarmManager.RTC_WAKEUP,date.time.time, alarmIntent)
        TEXTTIME.text = "$year-${month+1}-$dayOfMonth//$hour:$minute"
        Toast.makeText(this@Dla_Mezczyzn_Dzien_treningowy,"Alarm ustawiony pomyślnie",
        Toast.LENGTH_SHORT).show()
        }
}
```

Listing 4.26 Implementacja klasy obsługującej sekcję „dzień treningowy”

W powyższej metodzie właściwości „alarmMenager” oraz „alarmIntent” są inicjowane do zarządzania i wyzwalania alarmu. Linie 10-13 służą do znalezienia identyfikatorów przycisków oraz wyświetlenia napisu, które są przypisane do layoutu obsługującego daną aktywność. „TimeSetHour.setOnClickListener” ustawia element „onClickListener” na przycisk, który po kliknięciu tworzy i wyświetla okno wyboru czasu. „TimeSetDate.setOnClickListener” ustawia element „onClickListener” na przycisk, który po kliknięciu tworzy i wyświetla okno wyboru daty. „TimeSetAlarm.setOnClickListener” ustawia element „onClickListener” na przycisk, który ustawia wybraną datę, godzinę i alarm oraz wyświetla je w TextView TEXTTIME w miejsce gdzie został umieszczony napis „Zmień datę i godzinę”. „onTimeSet” i „onDateSet” to metody wywołania zwrotnego, które są wywoływane, gdy użytkownik wybiera datę lub godzinę w odpowiednich oknach dialogowych. Te metody aktualizują właściwości przechowujące wybraną datę, godzinę i alarm.

W „TimeSetAlarm.setOnClickListener” tworzona jest nowa instancja „Calendar” z wybraną datą i godziną. „AlarmManager” jest następnie używany do ustawienia alarmu przy użyciu wybranej daty i godziny, a „TextView” jest aktualizowany, aby wyświetlić wybraną datę i godzinę. Wyświetlany jest również komunikat potwierdzający pomyślne ustawienie alarmu.

5. Weryfikacja eksperymentalna aplikacji

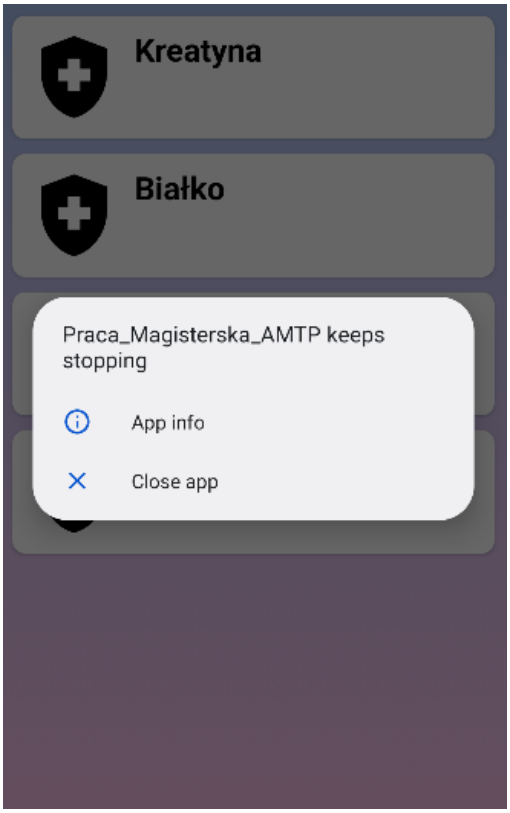
Weryfikacja eksperymentalna aplikacji została przeprowadzona poprzez wykonanie testów manualnych w celu sprawdzenia poprawności implementacji każdej sekcji, wykrycia ewentualnych błędów krytycznych oraz logicznych, a także błędów wynikających z użytkowania poprzez zastosowanie odpowiednich zabezpieczeń.

5.1. Poprawność implementacji

Błędy podczas wykonywania programu można podzielić na wiele sekcji, jednak na potrzeby pracy dyplomowej zostaną one podzielone na dwie grupy : błędy krytyczne oraz logiczne. Obie grupy zostały dokładnie opisane w niniejszym podrozdziale [15]

5.1.1. *Błędy krytyczne*

Błąd krytyczny uniemożliwia korzystanie z aplikacji bądź z jej części. Przykładem takiego błędu dla aplikacji mobilnej trenera personalnego byłaby niedziałająca sekcja w panelu głównym lub brak możliwości zalogowania się do aplikacji w związku z błędną implementacją bazy danych. Przykładowe zachowanie aplikacji podczas wystąpienia błędu krytycznego zostało przedstawione poniżej :



Rys 5.1. Przykład błędu krytycznego aplikacji

Błąd krytyczny przedstawiony na rys 5.1 został uzyskany poprzez celowe działanie mające na celu zatrzymanie działania aplikacji. W celu uzyskania błędu krytycznego dla sekcji suplementacji dla kobiet został za-komentowany jeden z obiektów przedstawionych na listingu 4.16, który odpowiada za przekazanie obrazu do aktywności wyświetlającej informacje o suplemencie „Omega-3”.

```
1  Zdjecie = arrayOf(
2      R.drawable.krea_foto_lepsze,
3      R.drawable.bialko_foto,
4      R.drawable.witaminy_foto,
5  //  R.drawable.omega_foto
6  )
```

Listing 5.1. Dodanie komentarza do jednego z elementów zdjęć sekcji suplementacji.

Sekcja składa się z czterech elementów, natomiast poprzez dodanie komentarza jak zostało to przedstawione powyżej w listingu 5.1 poprzez znaki „//”, obiekt zawiera trzy elementy przez co logika algorytmu się nie zgadza. Program szuka zdjęcia dla suplementu „Omega-3” jednak ono nie istnieje, co skutkuje przerwaniem działania aplikacji.

Błędy krytyczne zostały wyeliminowane w trakcie implementacji poprzez testowanie wszystkich funkcji na bieżąco po ich zaimplementowaniu. W trakcie wykonywania testów po w pełni skończonej aplikacji nie zostały wykryte żadne błędy krytyczne.

5.1.2. *Błędy logiczne*

Błąd logiczny występuje podczas błędnej implementacji logicznej aplikacji. Podczas błędów logicznych program może wykonywać inne operacje niż są mu zlecane, prowadzić użytkownika do innej aktywności bądź wyświetlać inne informacje niż te, o które prosił użytkownik.

Przykładem błędu logicznego dla aplikacji mobilnej trenera personalnego będzie naciśnięcie przycisku „Dla mężczyzn” w oknie wyboru modelu treningowego z podziałem na płcie, gdzie aplikacja przeniesie użytkownika do sekcji dla kobiet, mimo, że została wybrana inna sekcja.

Rys 5.2 Przykład błędu logicznego aplikacji

Rys 5.2, przedstawiony powyżej, zobrazowuje przykład błędu logicznego. Użytkownik w sekcji przedstawionej na rys. 4.3. dokonał wyboru sekcji dla mężczyzn, natomiast poprzez błąd w implementacji został przeniesiony do sekcji dla kobiet. O tym, że użytkownik chciał wybrać sekcję dla mężczyzn informuje komunikat przedstawiony na powyższym rysunku. Błąd ten wynika z logiki implementacji i umożliwia dalsze działanie aplikacji, dlatego nie można zaliczyć go do błędu krytycznego.

```
pm.setOnClickListener(){
    Toast.makeText(this,"Sekcja dla mężczyzn", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,Sekcja_dla_kobiet::class.java)
    startActivity(intent)
}

pk.setOnClickListener(){
    Toast.makeText(this,"Sekcja dla kobiet", Toast.LENGTH_SHORT).show()
    val intent = Intent(applicationContext,Sekcja_dla_kobiet::class.java)
    startActivity(intent)
}
```

Listing 5.2. Implementacja błędu logicznego aplikacji

Przykład błędu logicznego został przedstawiony na powyższym listingu 5.2. Porównując listing 5.2 z listingiem 4.6, gdzie implementacja została przedstawiona w sposób prawidłowy, linie 3 oraz 9 przenoszą użytkownika do tej samej sekcji. Dla przycisku dla mężczyzn, którego implementacja jest w wierszach 1-5, przycisk powinien przenieść użytkownika do sekcji dla mężczyzn zgodnie z przykładem z listingu 4.6

Powyższy błąd logiczny został spowodowany celowo dla zobrazowania przykładu. Wszelkie błędy logiczne zostały wyeliminowane na etapie planowania i implementacji. W testach nie wykryto błędów logicznych dla aplikacji mobilnej trenera personalnego.

5.2. Błędy wynikające z użytkowania

Zabezpieczenia zaimplementowane w aplikacji pozwalają użytkownikowi na bezproblemowe korzystanie z aplikacji. Wszelkie możliwe okna do wpisania danych – zarówno w sekcji logowania jak i sekcji kalkulatorów zostały zabezpieczone przed ewentualnymi błędami w przypadku niepoprawnego użytkowania.

5.2.1. *Zabezpieczenie sekcji rejestracji i logowania*

Zabezpieczenie dla sekcji rejestracji zostało wykonane w oparciu o potencjalne niewypełnienie wszystkich wymaganych poł w celu poprawnej rejestracji. Do zarejestrowania konta użytkownik zobligowany jest do utworzenia loginu oraz hasła i jego powtórzenia w celu sprawdzenia. Ewentualnie niewypełnienie któregoś z wyżej wymienionych pól mogłoby skutkować błędem krytycznym aplikacji.

```
1   if(TextUtils.isEmpty(unametext)
2     || TextUtils.isEmpty(pwordtext)
3     || TextUtils.isEmpty(pword2text)){
4
5         Toast.makeText(this, "Dodaj nazwe użytkownika, " +
6         "hasło i potwierdzienie hasła", Toast.LENGTH_SHORT).show()
7   }
```

Listing 5.3. Zabezpieczenie pól rejestracji

Powyższy listing prezentuje zabezpieczenie pól dla rejestracji. Instrukcja warunkowa zostaje spełniona w przypadku, gdy jedno z pól pozostaje puste. Po wypełnieniu danej instrukcji zostaje wyświetlony komunikat dla użytkownika o konieczności wypełnieniu wszystkich pól.

Rys. 5.3. Zabezpieczenie sekcji rejestracji.

W przypadku niewypełnienia pól użytkownik zostaje poproszony o podanie wymaganych danych. Informacja zostaje przekazana w postaci wyskakującego okna ze stosowną informacją. Informacja z rys 5.3 zostaje wyświetlona w przypadku niewypełnienia jednego, bądź wszystkich wymaganych danych.

Sekcja logowania została zabezpieczona w sposób analogiczny do sekcji rejestracji. Implementacja zabezpieczenia dla sekcji logowania została przedstawiona na listingu 5.4.

```
1   if(TextUtils.isEmpty(useredtx)
2     || TextUtils.isEmpty(passedtx)){
3
4     Toast.makeText(this,"Podaj nazwę użytkownika" +
5     " i hasło", Toast.LENGTH_SHORT).show()
6   }
```

Listing 5.4. Zabezpieczenie sekcji logowania.

Zgodnie z powyższym listingiem w przypadku niewypełnienia przynajmniej jednego z pól w sekcji logowania użytkownik zostaje powiadomiony o konieczności podania loginu i hasła jak zostało to zaimplementowane w liniach 4-5.

Rys 5.4. Zabezpieczenie sekcji logowania.

Na rys 5.4, przedstawionym powyżej została zobrazowana implementacja zabezpieczenia dla sekcji logowania. Naciśnięcie nieuzupełnionych pól powoduje pojawienie się stosownego komunikatu w postaci wyskakującego okna.

5.2.2. *Zabezpieczenie sekcji kalkulatorów sportowych*

Zabezpieczenia implementowane w sekcji kalkulatorów sportowych wzorowane były na zabezpieczeniach implementowanych dla sekcji rejestracji oraz logowania. Tutaj również należało wziąć pod uwagę ewentualne pozostawienie pustych pól do obliczeń przez użytkownika.

Rys 5.5. Przykład zabezpieczeń dla sekcji kalkulatorów sportowych

Na rys 5.5, przedstawionym powyżej, został zobrazowany przykład zabezpieczenia dla kalkulatorów sportowych. Na grafice po lewej stronie użytkownik nie wypełnił pola wieku dla kalkulatora zapotrzebowania kalorycznego. W takich przypadku został poproszony o jego podanie w stosownym komunikacie. Dla grafiki przedstawionej na prawej stronie tester zapomniał wpisać swojego wzrostu w celu obliczenia wskaźnika BMI. O fakcie został poinformowany w stosownym komunikacie.

Listing prezentujący implementacje powyższych rozwiązań został przedstawiony na następnej stronie. W przykładzie zostanie omówione zabezpieczenie dla kalkulatora zapotrzebowania kalorycznego w związku z największą złożonością pól. Rozwiązania dla kalkulatora BMI oraz zapotrzebowania na wodę zostały zaimplementowane analogicznie zgodnie z listingiem 5.5 dla obu sekcji treningowych.

```
1  private fun PustePole(Wzrost:String?,Waga:String?,Wiek:String?):Boolean{
2      return when{
3          Waga.isNullOrEmpty() -> {
4              Toast.makeText(this,"Podaj wagę",
5                  Toast.LENGTH_SHORT).show()
6              return false
7          }
8          Wzrost.isNullOrEmpty() -> {
9              Toast.makeText(this,"Podaj wzrost",
10                 Toast.LENGTH_SHORT).show()
11             return false
12         }
13         Wiek.isNullOrEmpty() -> {
14             Toast.makeText(this,"Podaj wiek",
15                 Toast.LENGTH_SHORT).show()
16             return false
17         }
18         else -> {
19             return true
20         }
21
22     }
23 }
```

Listing 5.5. Zabezpieczenia dla kalkulatora zapotrzebowania kalorycznego.

Funkcja „PustePole” wykonuje się w przypadku niewypełnienia przynajmniej jednej komórki w kalkulatorze zapotrzebowania kalorycznego zgodnie z implementacją przedstawioną w listingu 4.20. Funkcja ta sprawdza czy wszystkie pola zostały wypełnione kiedy przycisk „Oblicz” zostanie naciśnięty przez użytkownika. Jeżeli tylko jedno pole zostało puste to zostaje wyświetlony komunikat o informacji, które pole należy wypełnić jak zostało to przedstawione na rys 5.5 na grafice po lewej stronie. W tym przypadku został wykonany algorytm zaimplementowany w liniach 13-17 z komunikatem o podaniu wieku. Jeżeli więcej niż jedno pole pozostało by puste to kolejno od góry użytkownik otrzymywałbym informację z prośbą o wypełnieniu kolejnych pól. Przykładowo w przypadku pozostawienia wszystkich pól pustych najpierw pojawiłaby się informacja o podaniu wagi i dopiero po jej uzupełnieniu ukaże się informacja o braku wartości w polu wzrost. Po wypełnieniu pola wzrostu użytkownik zostałby poproszony o podanie swojego wieku, jeżeli pole to pozostawałby by w dalszym ciągu puste. Dzieje się tak ponieważ rozgałęzienie warunkowe „when” w języku Kotlin wykonuje się od góry do dołu zgodnie z przyjętą implementacją i w przypadku spełnienia się pierwszego warunku funkcja zostaje wykonana i pomijane są pozostałe warunki, mimo iż one też zostają spełnione.

```
1  <com.google.android.material.textfield.TextInputEditText
2      android:id="@+id/Waga_kalkulator_zapotrzebowanie"
3      android:layout_width="match_parent"
4      android:layout_height="wrap_content"
5      android:hint="Podaj wagę w kilogramach"
6      android:inputType="numberDecimal"
7  />
```

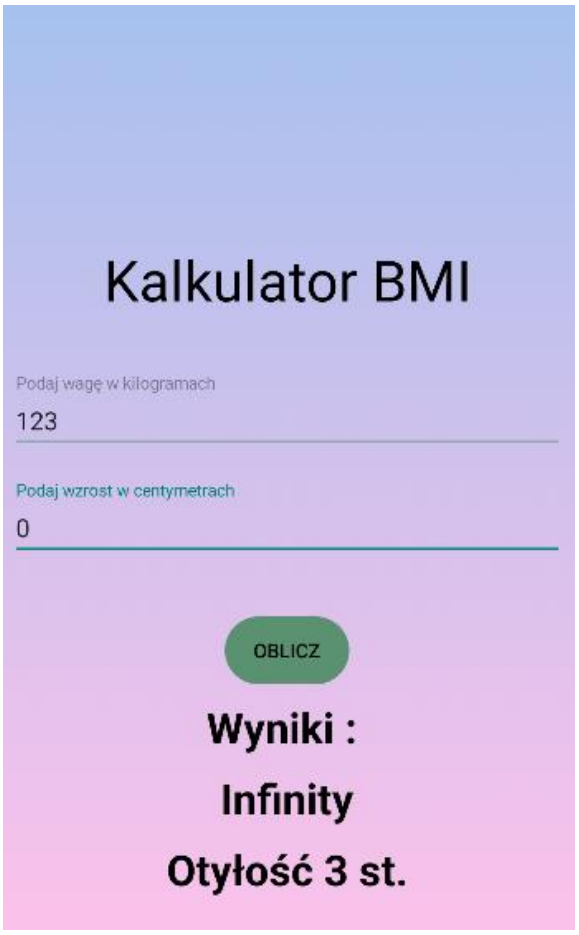
Listing 5.6. Implementacja ekranu numerycznego.

W listingu 5.6, który został przedstawiony powyżej, znajduje się wycinek algorytmu z pliku XML dla komórki odpowiadającej za wagę w kalkulatorze zapotrzebowania kalorycznego. Interesująca w tym przypadku jest linia 6 kodu, która powoduje ograniczenie dla użytkownika w postaci możliwości wprowadzenia tylko wartości liczbowych, w tym wartości dziesiętnych. Oznacza to, że po ustawieniu tego typu wprowadzania użytkownik może wprowadzać tylko cyfry i kropkę dziesiętną. Inne znaki, takie jak litery lub symbole, nie będą akceptowane.

Rys 5.6 Wygląd wprowadzania wartości dla pól w kalkulatorach sportowych.

Na rys 5.6 został przedstawiony wygląd ekranu użytkownika, w przypadku kiedy chce on wprowadzić wartość dla danej komórki w dowolnym kalkulatorze sportowym. Na jego ekranie pojawia się widok klawiatury numerycznej zawierającej kropkę dziesiętną co ogranicza możliwości użytkownika do wprowadzenia innych niechcianych znaków dla tych pól. Rozwiązanie to zostało zastosowane dla wszystkich rodzajów kalkulatorów w obu sekcjach.

W sekcji kalkulatorów sportowych należało wziąć pod uwagę również przypadek kiedy użytkownik będzie chciał do wzoru na BMI podstawić wartość zero w miejsce mianownika. Taka sytuacja z matematycznego punktu widzenia jest niemożliwa do policzenia dlatego również ten przypadek wymagał zastosowania dodatkowego algorytmu. Przypadek dla dzielenia przez zero dla wzoru (1.3) został przedstawiony poniżej.



Rys 5.7. Dzielenie przez zero w kalkulatorze BMI

Wynikiem dzielenia liczby 123 przez zero jest nieskończoność jak zostało to przedstawione na rys 5.7, zobrazowującym przypadek kiedy kalkulator nie posiada zabezpieczenia z dzieleniem przez zero.

```
1  if(PustePole(Wzrost,Waga)){
2      if (Wzrost.toFloat() != 0F) {
3          val bmi = Waga.toFloat() / ((Wzrost.toFloat() / 100)
4          * (Wzrost.toFloat() / 100))
5          val bmi2Digits = String.format("%.2f", bmi).toFloat()
6          Wystietl(bmi2Digits)
7      } else {
8          Toast.makeText(this, "Wzrost nie może być równy zero",
9          Toast.LENGTH_SHORT).show()
10     }
11 }
```

Listing 5.7. Implementacja zabiezpieczenia dla dzielenia przez zero

Powyższy listing przedstawia sposób implementacji zabezpieczenia dla dzielenia przez zero. Dla wartości wzrostu została dodana instrukcja warunkowa sprawdzająca czy wartość ta jest różna od zera. Jeżeli tak to obliczenia zostają wykonane, natomiast w przypadku wartości zero dla wzrostu zostaje wyświetlony stosowny komunikat informujący użytkownika o tym, że wzrost musi być różny od zera.

Błąd ten został wykryty na etapie testowania, dlatego implementacja zabezpieczenia nie została uwzględniona w algorytmie 4.21. W przypadku nieuwzględnienia zabezpieczenia dla mianownika kalkulator nadal będzie wykonywał swoje zadanie, jednak niektóre obliczenia nie będą poprawne zgodnie z przyjętymi zasadami matematycznymi.

6. Wnioski

W niniejszej pracy magisterskiej przedstawiono implementację wirtualnego trenera personalnego umożliwiającego użytkownikowi pozyskanie kompleksowej wiedzy z zakresu sportów siłowych. Aplikacja została wykonana dla platformy Android w języku Kotlin.

Praca obejmuje implementację systemu logowania wraz z dedykowaną bazą danych oraz dwóch panelów indywidualnych dla mężczyzn oraz kobiet. Każdy panel wyposażony jest w sześć tematycznych sekcji obejmujących osobny obszar wiedzy. Każdy z przedstawionych w ramach pracy paneli posiada swoje indywidualne funkcje pozwalające na interakcję z użytkownikiem oraz pozyskanie wiedzy w prostej i wygodniej formie.

Przeprowadzona weryfikacja eksperymentalna stwierdziła poprawność implementacji. W trakcie weryfikacji eksperymentalnej nie zostały wykryte żadne błędy krytyczne – uniemożliwiające korzystanie z aplikacji, czy błędy logiczne – utrudniające korzystanie z wirtualnego trenera. Pozostałe wykryte błędy podczas testowania oraz zastosowane zabezpieczenia zostały opisane w rozdziale piątym pracy dyplomowej.

Wyrazem podsumowania, przedstawiona powyżej praca jest przykładem wykorzystania środowiska Android Studio dla systemu operacyjnego Android jako narzędzia pozwalającego na zaimplementowanie osobistego wirtualnego trenera personalnego jako elementu wspomagającego trening siłowy. Powyższa aplikacja jest przykładem jednej z wielu aplikacji dostępnych na rynku, które obejmują podobny zakres tematyczny. Elementem charakterystycznym aplikacji trenera personalnego w porównaniu z innymi komercyjnymi rozwiązaniami jest prosta oraz intuicyjna obsługa aplikacji, która na bieżąco komunikuje się z użytkownikiem.

7. Bibliografia

[1] https://canalys.com/newsroom/global-smartphone-market-2022 (06.03.2023)

[2]https://www.telix.pl/sprzet/apple/2016/08/rekordowy-udzial-androida-na-rynku-smartfonow/ (08.03.2023)

[3] https://mobirank.pl/ (10.03.2023)

[4] https://en.wikipedia.org/wiki/Android_(operating_system) (10.03.2023)

[5] https://developer.android.com/guide/platform (08.03.2023)

[6] https://developer.android.com/ (10.03.2023)

[7] https://en.wikipedia.org/wiki/Android_Studio (10.03.2023)

[8]https://skalkuluj.pl/kalkulator-zapotrzebowania-kalorycznego-bmr(19.03.2023)

[9] https://dietetykpro.pl/kalkulatory/bmi (19.03.2023)

[10] https://www.zikodlazdrowia.org/blog/sprawdz-swoje-bmi/ (19.03.2023)

[11]https://www.budujmase.pl/kalkulatory/kalkulator-zapotrzebowania-na-wode.html (19.03.2023)

[12]https://www.maczfit.pl/blog/makroskladniki-czym-sa-jak-je-obliczyc/(25.04.2023)

[13] https://pacjent.gov.pl/diety/dlaczego-warto-pic-wode(28.04.2023)

[14] https://myfitness.gazeta.pl (28.04.2023)

[15]https://testerzy.pl/artykuly/bledy-podczas-tworzenia-kodu-zrodlowego (03.05.2023)