Rok akademicki 2014/2015

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

MAREK MAJDE

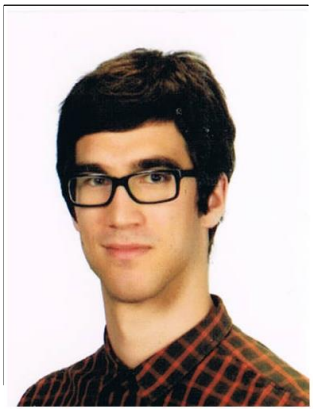
Aplikacja umożliwiająca rywalizację biegową w czasie rzeczywistym

Opiekun pracy
dr hab. inż. Piotr Gawrysiak

Ocena: .....

.....

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego



Kierunek: Informatyka

Specjalność: Inżynieria Systemów Informatycznych

Data urodzenia: 1992.02.06

Data rozpoczęcia studiów: 2012.02.20

Życiorys

Urodziłem się 6 lutego 1992 w Warszawie. Ukończyłem gimnazjum imienia Tadeusza Reytana w Warszawie oraz liceum imienia Jana Kochanowskiego, również w Warszawie. W lutym 2012 rozpocząłem studia na kierunku Informatyka, na wydziale Elektroniki i Technik Informacyjnych. Od listopada 2014 roku pracuję w firmie Kontomierz.pl, na stanowisku programista Java. Zajmuję się tam implementacją narzędzia do agregacji danych finansowych, z wykorzystaniem techniki „screen scraping”.

.....
Podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu ..... 20__ r

z wynikiem .....

Ogólny wynik studiów: .....

Dodatkowe wnioski i uwagi Komisji: .....

.....

.....

STRESZCZENIE

Praca przedstawia projekt aplikacji, umożliwiającej osobom biegającym rywalizację w czasie rzeczywistym. Aplikacja napisana jest na system Android. Do obsługi rywalizacji w czasie rzeczywistym wykorzystany jest serwer Google App Engine. Funkcjonalność aplikacji mobilnej opiera się o wykorzystanie systemu globalnego pozycjonowania, do pomiaru dystansu przebytego podczas biegu. W pracy inżynierskiej opisuję motywację do stworzenia projektu, opisuję problemy który musiałem rozwiązać oraz narzędzia których użyłem. Przedstawiona jest również implementacja najważniejszych funkcji aplikacji mobilnej.

Słowa kluczowe:
aplikacja mobilna, chmura obliczeniowa, bieganie, system czasu rzeczywistego, Google App Engine, Android, System Globalnego Pozycjonowania, rywalizacja, bieganie

---

SOFTWARE APPLICATION THAT ENABLES REAL TIME RUN COMPETITION

In this thesis I wanted to present a software, that enables real time competition between runners. There are many mobile applications on the market, but none of them provides described functionality. Mobile application is programmed on Android software, whereas server application is created with Google App Engine. Mobile functionality is based on Global Positioning System module, which measures the distance covered by runners. In following chapters I am describing my motivation to create this project. I am also writing about problems, which I had to solve and about the tools that were used to solve those problems. At the end I am describing the functionality, with most important code examples.

Keywords:
mobile application, cloud computing, real time system, Google App Engine, Android, Global Positioning System, rivalry, running

Spis treści

1 Wstęp

W ciągu ostatnich lat rynek urządzeń mobilnych przeżył metamorfozę. Jeszcze kilka lat temu nikt by nie pomyślał, że to telefon będzie nieodłącznym narzędziem towarzyszącym ludziom praktycznie przy każdej czynności. Nie jest już istotne czy potrzebujemy zadzwonić do znajomego, czy znaleźć przepis na zrobienie obiadu. Telefony komórkowe stały się naszymi prawymi rękami. Wraz ze wzrastającymi potrzebami użytkowników rośnie również liczba sposobów, na jakie można wykorzystać urządzenia mobile. Jedną z nich jest właśnie możliwość uprawiania sportu przy wsparciu technologii. Inteligentny telefon może informować użytkownika, jaki dystans przejechał na rowerze lub jak dużo spalił kalorii. Bardzo istotnym aspektem rozwoju aplikacji mobilnych są również portale społecznościowe. Jaką wartość ma pokonanie stu kilometrów na rowerze, jeśli nie możemy pochwalić się tą wiadomością ze znajomymi? Nie krytykuję tej formy dzielenia się własnymi sukcesami, wręcz przeciwnie, uważam za naturalne informowanie przyjaciół i bliskich o naszych sukcesach i zmaganiach. Chcę jedynie zwrócić uwagę na dynamicznie zachodzące zmiany, które dzieją się wokół nas.

Obok rynku aplikacji mobilnych powstaje w całości niezależnie inne potężne narzędzie. Chodzi o chmury obliczeniowe, które dają niezliczone możliwości wykorzystania. Umożliwiają one przechowywanie różnorodnych danych z komputera prywatnego na zewnętrznych serwerach, a co za tym idzie, nieograniczonego dostępu do nich z dowolnego miejsca na Ziemi. Dane są przesyłane przez sieć z prędkością, która umożliwia natychmiastowe odczytanie plików z chmury. Innym wykorzystaniem potencjału chmury obliczeniowej jest udostępnienie oprogramowania klientom. Może to być zarówno prosty program do edycji dokumentów tekstowych, jak i samodzielna maszyna, której klient może używać jako zdalnego komputera.

Dzięki połączeniu aplikacji mobilnej z chmurą obliczeniową, zyskujemy całkowicie nowe możliwości. Telefon komórkowy daje elastyczność i mobilność, a chmura obliczeniowa natychmiastowy dostęp do danych. W tej pracy chciałbym przedstawić połączenie tych dwóch narzędzi. Tematem mojej pracy inżynierskiej jest aplikacja, która umożliwia osobom biegającym rywalizację w czasie rzeczywistym. Idea jest bardzo prosta. Telefon komórkowy (smartfon) mierzy bieg użytkownika, natomiast chmura obliczeniowa umożliwia otrzymywanie natychmiastowej informacji o pozycji drugiego uczestnika. W

wersji podstawowej (pilotowej), aplikacja ma umożliwiać rywalizację ze znajomym lub obcą osobą w czasie rzeczywistym. Strona mobilna zaimplementowana jest na systemie Android. Serwer zaimplementowany jest z wykorzystaniem platformy Google App Engine.

Motywacją do wyboru takiego tematu była przede wszystkim możliwość nauki różnych narzędzi programistycznych. Po stronie serwera musiałem poznać wszystkie moduły potrzebne do zapisu danych użytkowników oraz do wymiany danych między użytkownikami. Już przed rozpoczęciem implementacji projektu mogłem się dużo nauczyć. Wynikało to z faktu, że zanim powstała jakakolwiek linijka kodu, przeanalizowane zostały różne dostępne rozwiązania. Często wymagało to zagłębienia się w dokumentację oraz w sposób użycia danego narzędzia. Wiedząc już jakie rozwiązania zostaną użyte w aplikacji, implementowałem małe fragmenty kodu niezwiązane z aplikacją, aby lepiej zrozumieć ich funkcjonowanie. To pozwoliło na pełne zrozumienie poszczególnych modułów.

Zaprojektowana aplikacja rozwiązuje realny problem, który ma wiele osób uprawiających sporty. Ja również zaliczam się do grona osób, którym w bieganiu brakuje rywalizacji. Jest to czynnik, który z jednej strony motywuje, z drugiej strony jest formą zabawy. Istniejące rozwiązania na rynku aplikacji mobilnych nie spełniają moich oczekiwań. Dlatego zdecydowałem, że w końcu powinien powstać projekt tego typu.

Przedstawiona praca inżynierska składa się z ośmiu głównych rozdziałów, z czego pierwszy to wstęp a ostatni to podsumowanie. W rozdziale drugim przedstawiono szczegółowe omówienie koncepcji. Rozdział trzeci zawiera opis rozwiązań istniejących na rynku. Tematem rozdziału czwartego są narzędzia i moduły wykorzystane przy tworzeniu projektu związane z serwerem oraz ze stroną mobilną aplikacji. W rozdziale piątym praca skupia się na omówieniu rozwiązań zastosowanych przy krytycznych aspektów projektu, natomiast rozdział szósty to opis implementacji tych rozwiązań na platformie Android. Opis implementacji dotyczy jedynie strony mobilnej aplikacji, implementację serwera wykonał Paweł Stępień, student wydziału Elektroniki i Technik Informacyjnych. W rozdziale siódmym opisane zostało testowanie projektu oraz logowanie zdarzeń.

2 Omówienie koncepcji

Tematem mojej pracy jest aplikacja mobilna umożliwiająca rywalizację biegową w czasie rzeczywistym. W tym rozdziale chciałbym szczegółowo przedstawić koncepcję projektu. Napiszę również o wymaganiach, które powinna spełniać aplikacja.

Jak już wspomniałem, zaprojektowana aplikacja dzieli się na dwa główne moduły, aplikację mobilną oraz serwer aplikacji. Aplikacja mobilna napisana jest na system Android w języku Java, natomiast serwer aplikacji zaimplementowany jest z wykorzystaniem narzędzia Google App Engine. System Android został wybrany ze względu na ilość użytkowników którzy z niego korzystają. Ponieważ urządzenia z systemem Android dominują na rynku mobilnym, istnieje spora szansa na dotarcie do większej ilości użytkowników, mając gotowy produkt. Im więcej osób korzysta z aplikacji, tym my jako twórcy, mamy więcej możliwości na dalszy rozwój produktu. Implementacja serwera przy pomocy Google App Engine posiada wiele zalet. Po pierwsze oprócz zwykłego serwera udostępniony jest również moduł Google Cloud Endpoint, który pozwala na łatwą integrację z różnymi klientami mobilnymi. Dodatkowo Google udostępnia bazę danych Datastore oraz moduł Memcache, które znacząco usprawniają działanie aplikacji tego typu. Szczegółowy opis działania tych modułów zawarty jest w rozdziale *4.1. Serwer*.

Podstawowa funkcjonalność jaką powinien zapewniać system, to możliwość rywalizacji ze znajomym lub obcą osobą w czasie rzeczywistym. Przy pierwszym użyciu aplikacji użytkownik zobowiązany jest do utworzenia konta. Do stworzenia konta wymagane jest podanie unikalnego loginu i hasła. Przy kolejnych uruchomieniach aplikacji, użytkownik nie będzie musiał podawać tych danych ponownie, ponieważ aplikacja korzysta z protokołu OAuth 2.0, który pozwala na automatyczne wykonanie uwierzytelnienia. Po zalogowaniu użytkownik wybiera czy chce rywalizować ze znajomym czy z obcą osobą. Po wybraniu opcji biegu ze znajomym są dwie możliwości. Można dołączyć do biegu stworzonego przez znajomego lub przygotować bieg. W przypadku chęci dołączenia do biegu już stworzonego, wystarczy wybrać dystans biegu i kliknąć odpowiedni przycisk. Wtedy po kilkunastu sekundach rozpocznie się bieg. Aplikacja umożliwia również stworzenie biegu. Ustala się wtedy dystans do pokonania oraz podajemy login znajomego, który ma uczestniczyć w rywalizacji. Podczas biegu na ekranie telefonu widoczna jest informacja, który uczestnik znajduje się na czele. Jeżeli jeden z uczestników biegu pokona

ustalony wcześniej dystans, bieg zostaje zakończony. Podczas rywalizacji z obcą osobą, cały proces przebiega bardzo podobnie, z wyjątkiem wyboru przeciwnika. Podczas biegu z losową osobą, informujemy aplikację, że chcemy wziąć udział w biegu, a ta znajduje przeciwnika.

Aby system funkcjonował poprawnie musi spełnić określone wymagania. Po pierwsze, musi mierzyć bieg użytkownika. Pomiar ten powinien odbywać się z jak największą dokładnością. Do tego system musi sobie radzić w sytuacjach wyjątkowych. Jako sytuację wyjątkową rozumiem działanie aplikacji przy zerwanym połączeniu z siecią telefoniczną (połączenie internetowe), przy słabym sygnale modułu GPS, przy złej pogodzie. Aplikacja mobilna powinna jednoznacznie identyfikować użytkowników oraz dawać im możliwość zapisywania informacji o ich znajomych. Aplikacja mobilna musi również komunikować się z serwerem. Przez sieć wysyłane są cząstkowe rezultaty biegów, zawierające przebyty dystans i czas od startu. Dane dotyczące rezultatu przeciwnika są odbierane od serwera. Tym samym aplikacja jest w stanie ocenić kto wygrywa podczas biegu. Aplikacja mobilna powinna mieć zaimplementowany graficzny interfejs użytkownika. W przedstawionej pracy zostało to jednak pominięte, praca skoncentrowana jest na funkcjonalności i intuicyjności aplikacji. Jako twórca, chciałbym aby użytkownik nie zastanawiał się nad tym, którą opcję w aplikacji ma wybrać. Do implementacji strony mobilnej korzystam przede wszystkim z oficjalnej dokumentacji systemu Android.

Serwer aplikacji musi umożliwiać aplikacji mobilnej zarejestrowanie użytkownika, udział w biegu ze znajomym oraz udział w biegu z obcą osobą. Powinien także zapisywać dane, które wysłała mu strona mobilna aplikacji, aby następnie móc je udostępnić rywalowi. Dane te mogą być zapisywane w pamięci serwera lub w bazie danych. Zapisując dane w bazie danych sprawiamy, że aplikacja działa wolno. Najdłuższe operacje podczas działania takiego systemu to komunikacja przez sieć oraz komunikacja z bazą danych. Rozwiązaniem przyspieszającym działanie systemu jest wykorzystanie pamięci serwera, jednak może to ograniczać skalowalność. Jeżeli mamy uruchomione wiele instancji serwera na różnych maszynach możemy przekierowywać ruch sieciowy, zmniejszając tym samym obciążenie. Dane zapisane w pamięci jednej maszyny nie są dostępne na innej maszynie, jest to więc duże ograniczenie. Serwer aplikacji powinien też umożliwiać łatwą integrację z systemami klienckimi, to znaczy z telefonami na system Android oraz iOS. Twórcy powinni

mieć też możliwość dodawania nowych funkcjonalności, bez zbytniej ingerencji w istniejący system. Ważne jest, aby aplikacja działała zarówno przy małej ilości użytkowników jak i przy tysiącach użytkowników. Możemy do tego użyć gotowych narzędzi, co pozwoli nam skupić się na funkcjonalności systemu. Ma nam w tym pomóc użycie chmury obliczeniowej. Rozwiązanie to jest oczywiście płatne, ale w momencie kiedy mamy bardzo małą ilość użytkowników koszt ten jest bardzo niski, a opłaty musimy wnosić jedynie za wykorzystanie bazy danych. Dopiero przy większej ilości użytkowników utrzymanie systemu może być kosztowne.

W pracy inżynierskiej opisałem podstawową funkcjonalność aplikacji, jednak z czasem można zaimplementować dużo więcej opcji. Kolejnym krokiem rozwoju aplikacji jest rywalizacja wieloosobowa, nie odbywająca się w czasie rzeczywistym. To znaczy, na przykład zapisujemy się do turnieju pięcioosobowego, ale wyniki poznajemy dopiero wtedy, kiedy przebiegną wszyscy użytkownicy. Mając naprawdę dużą bazę użytkowników, można rozważyć realizację turniejów wieloosobowych w czasie rzeczywistym. Mam na myśli uczestnictwo setek, a nawet tysiąca osób, podobnie do zawodów odbywających się w większych miastach na całym świecie. Różnica jest taka, że tutaj każda osoba biorąca udział w biegu, może być w zupełnie innym miejscu na Ziemi.

3 Rozwiązania istniejące na rynku

Jedną z pierwszych rzeczy, które należało sprawdzić po stworzeniu pomysłu, to istniejące rozwiązania na rynku aplikacji mobilnych. Często w głowie rodzi się wiele koncepcji, które wydają się być wyjątkowe. Jednak zawsze należy zweryfikować, czy wymyślone rozwiązanie, nie zostało wcześniej zrealizowane przez kogoś innego i czy przypadkiem nie odnosi już sukcesów na rynku. W tym rozdziale opisane będą dwie aplikacje komercyjne – Endomondo oraz Nike+ Running, które można pobrać na telefon komórkowy oraz jedną aplikację z wolnych źródeł oprogramowania - RunnerUp. Pomiędzy aplikacjami dostępnymi na rynku a projektem przedstawionym w tej pracy można znaleźć cechy wspólne, ale również różnice.

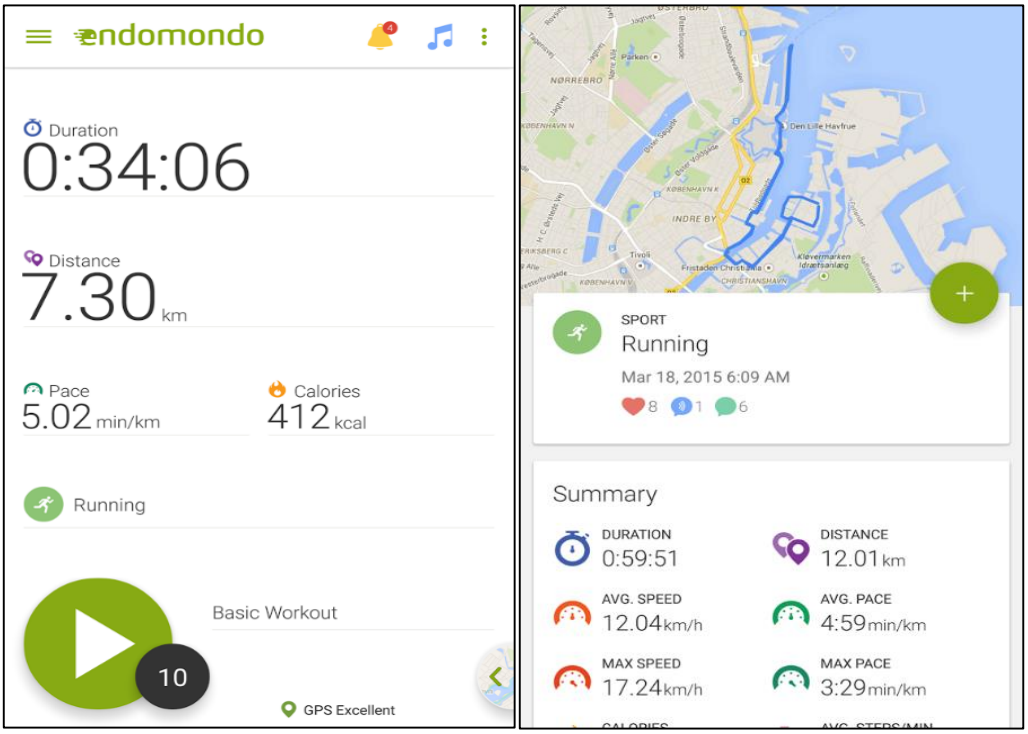
3.1 Endomondo

Endomondo jest produktem, który od kilku lat utrzymuje się na rynku. Aplikacja skierowana jest do szerokiego grona osób, uprawiających wszelkiego rodzaju sporty. Użytkownik może w Endomondo wybrać praktycznie każdą dziedzinę sportu, którą chce „mierzyć”. Słowo mierzyć jest tutaj nie bez powodu wzięte w cudzysłów. W praktyce, niektóre sporty nie mogą być mierzone wyłącznie telefonem. Na przykład, jeśli spędzamy czas na siłowni, aplikacja nie wie, jak intensywny jest nasz trening. Może jedynie zmierzyć czas naszej aktywności, co nie jest pełną informacją. Sporty które możemy wybrać, zaczynają się na bieganiu i jeżdżeniu na rowerze, a kończą na zajęciach z jogi lub fitnessu. Aplikacja liczy spalone podczas aktywności kalorie. Najprawdopodobniej opiera tą informację o dane użytkownika, podawane w profilu tzn. wagę, płeć, wzrost, czas treningu oraz jego typ. Ciężko jednak uwierzyć w skuteczność tych obliczeń, ponieważ telefon nie jest w stanie sprawdzić naszego pulsu, aby określić czy dany fragment treningu nie jest wykonywany z większą intensywnością. Podczas biegu, aplikacja wykorzystuje moduł GPS, ale wspiera się również danymi z wież telekomunikacyjnych. Pomiar biegu nie jest idealny, natomiast w przypadku zwykłego, rekreacyjnego biegu, jest zdecydowanie wystarczający.

Endomondo tworzy specjalne wydarzenia społecznościowe, które wprowadzają element rywalizacji. Polega to na tym, że np. użytkownik musi przez miesiąc wykonać więcej treningów niż inni użytkownicy aplikacji. Oczywiście aplikacja nie daje pełnej

możliwości kontrolowania tego, czy osoby rzeczywiście odbywały dane aktywności. Zachęceniem do brania udziału w takich wydarzeniach są nagrody, które rozlosowywano wśród najlepszych uczestników.

Na poniższych rysunkach 1 oraz 2 przedstawiony jest wygląd aplikacji. Główne aktywności prezentują się następująco (obrazki pobrane bezpośrednio ze strony „*https://play.google.com/store/apps/details?id=com.endomondo.android*”).



Rysunek 1 Główny ekran aktywności w aplikacji Endomondo (po lewej)

Rysunek 2 Rezultat biegu w aplikacji Endomondo(po prawej)

Na rysunku nr 1 widoczne jest główne okno aplikacji. Widać pole z pomiarem dystansu, pomiarem czasu, ilością spalonych kalorii oraz kilkoma dodatkowymi informacjami na temat odbywanej aktywności. Aktywność sportową możemy w każdej chwili zatrzymać lub zakończyć. Na rysunku nr 2 widoczny jest rezultat biegu. Na widoku mapy zaznaczona jest trasa oraz dodatkowe dane. Po biegu pokazane są standardowe informacje, dotyczące charakterystyki naszej aktywności. Aplikacja wyświetla trasę bez względu na to, czy tego chcemy, czy nie. Oznacza to również, że potrzebuje stałego kontaktu z modułami pobierającymi lokalizację.

Podczas wykonywania aktywności otrzymujemy głosową informację o naszych postępach. Po kilku miesiącach istnienia aplikacji, powstała również strona internetowa, na której można zalogować się używając danych z aplikacji mobilnej. Informacje są zsynchronizowane z danymi na telefonie. Jest to bardzo dobre rozwiązanie umożliwiające przeglądanie własnej historii treningów, statystyk i innych danych. Dodatkowo można obserwować profile znajomych, jest również możliwość bezpośredniego połączenia profilu z kontem na Google+ czy portalem Facebook.

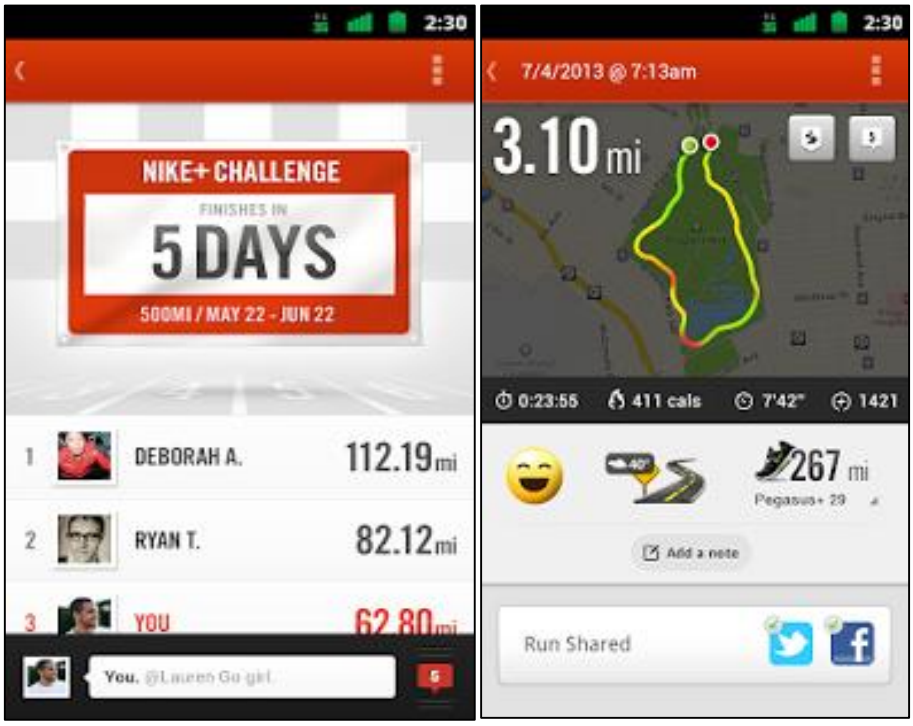
Endomondo to aplikacja która miała trudny początek na rynku mobilnym. Trudno jest zareklamować tego typu produkt, ponieważ każdy może używać go niezależnie i nie potrzebujemy do tego znajomych. Z tego powodu marketing jest utrudniony. W dodatku aby rozwijać system wymagane jest zainwestowanie czasu i pieniędzy. Dopóki nie wiadomo czy aplikacja odniesie sukces, nie warto się decydować na coś takiego. Dlatego też na początku, aplikacja pojawiła się w ograniczonej wersji, aby przez kolejne miesiące zebrać grono użytkowników. Następnie możliwy był rozwój oraz dodawanie płatnych opcji, dostępnych po wykupieniu abonamentu. Jest to aplikacja nastawiona na zysk z płatnych funkcjonalności. Spełnia ona potrzeby użytkowników. Mogą oni uprawiać sport, przechowywać historię swoich dokonań, a dodatkowo dzielić się tymi informacjami ze znajomymi.

3.2 Nike+ Running

Aplikacja Nike+ Running jest pod pewnymi względami bardzo podobna do Endomondo. Jej główną ideą jest pomiar biegu, który ma zmotywować użytkownika do regularnej aktywności. Aplikacja jest jednak bardziej profesjonalna. Nie umożliwia ona rywalizacji w dowolnej dziedzinie sportu, ale skupia się na bieganiu. Sprawia to, że firma Nike może skoncentrować się na rozwoju funkcji motywacyjnych w aplikacji. Nike+ Running wprowadza sporo elementów rywalizacji. Po pierwsze ma system osiągnięć. To znaczy jeśli przebiegniemy łączną, określoną ilość kilometrów, dostajemy specjalną odznakę. Taki system nagród jest standardowym sposobem motywowania użytkownika. Ma on wtedy poczucie, że jego działania nie idą na marne. Daje to również pewnego rodzaju świadectwo, że osiągnął on określony poziom umiejętności. Innym elementem motywującym jest system wyzwań. Polega on na tym, że w danym miesiącu musimy przebiec więcej kilometrów niż nasz znajomy. Robimy to w dowolnym czasie. Jest to

ciekawe rozwiązanie, musimy jednak oczekiwać na rezultat całej rywalizacji, a już dużo wcześniej może być widoczne kto wygra pojedynek. Porównując to rozwiązanie do sytuacji w realnym świecie, trzeba zaznaczyć, że na zawodach sportowych nie liczy się kto przebiegnie więcej kilometrów, tylko kto przebiegnie szybciej określony dystans. Rozwiązanie to tworzy więc potrzebę i nie zaspokaja już istniejącego wymagania.

Nike+ Running to dużo bardziej profesjonalny produkt. Widać to od razu po uruchomieniu aplikacji. Graficzny interfejs użytkownika jest zrealizowany na najwyższym poziomie. Dzieje się to nie bez powodu. Po pierwsze firma Nike dysponuje dużo większym kapitałem, niż twórcy Endomondo. Dodatkowo aplikacja ta ma służyć głównie celom marketingowym, to znaczy promocji marki Nike wśród osób biegających. Możliwe że sama aplikacja nie przynosi zysków z płatnych opcji dostępnych na telefonie, ale wyłącznie z promocji, reklam oraz odniesień do sklepu Nike w internecie. Trzeba jednak przyznać że aplikacja ta dużo lepiej mierzy przebyty dystans w porównaniu do aplikacji Endomondo. Zużywa jednak więcej energii baterii, co jest mniej istotne dla twórców aplikacji. (Obrazki pochodzą ze strony „*https://play.google.com/store/apps/details?id=com.nike.plusgps”*)



Rysunek 3 Ekran w aplikacji Nike+ Running przedstawiający wyzwanie (na lewo)

Rysunek 4 Ekran w aplikacji Nike+ Running przedstawiający rezultat biegu (na prawo)

Jak widać na rysunkach 3 oraz 4, wygląd znacząco różni się od interfejsu aplikacji Endomondo. Jest on bardziej agresywny, ma bardziej wyraziste kolory. Uważam że lepiej pasuje do aplikacji sportowej nastawionej na rywalizację. Na pierwszym zrzucie ekranu (**Błąd! Nie można odnaleźć źródła odwołania.**) widzimy przykładowe wyzwanie, oraz zapisane do niego osoby. Problemem może być to, że zbyt wcześnie widać, że ostatnia osoba nie ma zbyt wielu szans na poprawę swojej pozycji. Ciekawą opcją, którą dodano do aplikacji jest możliwość wsparcia naszego znajomego podczas biegu. Możemy zamieścić tekst motywujący naszego znajomego na jego profilu, a aplikacja wyświetli mu tę informację. Na drugim zrzucie ekranu (**Błąd! Nie można odnaleźć źródła odwołania.**) możemy zobaczyć zaznaczoną trasę po ukończeniu biegu. Wyświetlona trasa prezentuje gradient prędkości na danych fragmentach. Jeśli jakiś fragment biegu przebiegliśmy wyjątkowo wolno, zaznaczony on będzie na czerwono. Pokazuje to charakterystykę naszego biegu, co może nam się przydać jeśli chcemy poprawić swoje rezultaty. Jak widać aplikacja Nike+ Running jest lepszym produktem. Wpływa na to fakt, że jest ona skierowana do mniejszego grona odbiorców. Programiści mogli się więc skupić na dopracowywaniu istniejących funkcjonalności.

3.3 RunnerUp

Aplikacja RunnerUp dostępna jest w sklepie Google Play, ale ciekawsze jest to, że ten produkt może rozwijać każdy. Jest to rozwiązanie z kategorii wolnego oprogramowania. Oznacza to że każdy może rozwijać funkcjonalność tego projektu. Kiedy spojrzymy jednak w kod aplikacji, czasami trudno jest zrozumieć, jak działa dana funkcja. Nie wszyscy stosują się do dobrych praktyk programistycznych i miejscami kod jest trudny do przetworzenia. Szczególnie jeśli spojrzymy na funkcjonalność modułów, związanych z mierzeniem pulsu biegacza. Nie mniej jednak, każdy programista może się dużo nauczyć podczas pracy nad tą aplikacją. Przy okazji rozwija również sam projekt. Aplikacja RunnerUp ma dość prosty interfejs graficzny. Tak jak napisałem, celem programistów tworzących aplikację jest funkcjonalność, a nie dobrze sprzedający się produkt. Tym samym wiele funkcji, które w komercyjnych aplikacjach typu Endomondo czy Nike+ Running są płatne, tutaj udostępnione są za darmo. W dodatku każdy komentarz na sklepie Google Play, może być szybko rozpatrzony i zaimplementowany. W innych aplikacjach każda uwaga musi być

rozpatrzony przez firmę. Trzeba znaleźć programistę, który nie pracuje za darmo i dopiero wtedy dodana jest potrzebna funkcjonalność. Tutaj jeżeli jesteśmy programistami, to poświęcając własny czas możemy sami rozwinąć produkt i dodać to, czego nam brakuje.

3.4 Porównanie aplikacji

Analizując istniejące rozwiązania trzeba przyznać, że każde z nich wyróżnia się na rynku aplikacji mobilnych. Aplikacja Endomondo jest dostępna dla dużej ilości użytkowników, satysfakcjonując ich potrzeby. Nike+ Running to dobre rozwiązanie, jeśli ktoś chce biegać i rozwijać swoje umiejętności, ale nie zawsze znajduje do tego motywację. Ostatnia opisana przeze mnie aplikacja czyli RunnerUp, to rozwiązanie z kategorii otwartego oprogramowania, więc jej istnienie na rynku również jest uzasadnione. Może nie zachwyca ona wyglądem graficznym, ale daje użytkownikom to co dla nich najważniejsze, czyli obszerną funkcjonalność.

Cechą charakterystyczną wszystkich trzech aplikacji to pomiar biegu i reprezentacja wyników użytkownikowi. W moim rozwiązaniu motywacja jest jednak inna. W każdej przedstawionej aplikacji możemy wyświetlić trasę swojego biegu. Nakłada to pewne ograniczenia na implementację. W mojej aplikacji zamierzam wykorzystać dane o charakterystyce biegu, aby momentami móc zrezygnować z informacji pobieranych od modułu GPS. Tym samym będę w stanie oszczędzić na zużyciu baterii, a użytkownik nie zauważy różnicy.

Trudno powiedzieć, że opisane powyżej aplikacje byłyby dla mojego projektu konkurencją. Wystarczy pomyśleć, że moja aplikacja mogłaby działać równolegle z aplikacjami wymienionymi powyżej. W projekcie chciałbym skupić się wyłącznie na rywalizacji. Priorytetowo jest to rywalizacja w czasie rzeczywistym. Jeśli ktoś chciałby zmierzyć bieg, może uruchomić sobie aplikację Endomondo w tle.

Podstawową funkcjonalnością w projektowanej aplikacji, jest bieg jeden na jednego ze znajomym lub obcą osobą. Użytkownicy biegną określony wcześniej dystans. W trakcie biegu mogą zobaczyć kto jest szybszy. Tworząc moją aplikację chciałbym wypełnić lukę na rynku aplikacji mobilnych. Chciałbym także zaspokoić potrzeby wielu biegaczy, którzy chcieliby się ścigać z innymi, ale nie mogą tego robić jeśli są w innych miejscach na świecie. Jeśli spojrzymy na aplikację Nike+ Running, trzeba przyznać, że

wprowadza ona element rywalizacji. Tak jak napisałem, jest to jednak tworzenie potrzeb użytkownika, a nie zaspokajanie tych istniejących. Patrząc perspektywicznie, wszystkie elementy zawarte w komercyjnych rozwiązaniach można by dodać w mojej aplikacji. Byłby to jednak tylko dodatek. Trzonem projektu ma być rywalizacja w czasie rzeczywistym. Wykorzystując do tego chmurę obliczeniową mamy zapewnienie, że aplikacja będzie działać przy dużym obciążeniu. W dodatku rozwiązanie to nie jest rozwiązaniem komercyjnym, więc mogę skierować projekt w dowolnym kierunku, dając użytkownikowi to, czego naprawdę potrzebuje.

4 Narzędzia

W tym rozdziale chciałbym opisać wykorzystane w projekcie narzędzia. Rozdział podzieliłem na dwa główne podrozdziały. Pierwszy z nich zawiera opis narzędzi po stronie serwera. Drugi podrozdział zawiera opis narzędzi, bez których funkcjonowanie aplikacji mobilnej nie byłoby możliwe. Przy opisie nie wchodzę w szczegóły implementacyjne. Skupiam się na wysokopoziomowym opisie, który ma przybliżyć funkcjonalność i możliwości poszczególnych modułów.

4.1 Serwer

Po stronie serwera wykorzystywane są narzędzia, które wykonują większość pracy za programistę. Projektant aplikacji musi jedynie w odpowiedni sposób je skonfigurować oraz zapewnić prawidłową komunikację tych modułów.

4.1.1 Google App Engine

Google App Engine jest serwerem opartym o chmurę obliczeniową z zakresu *Platforma jako Usłu*ga (ang. *Platform as a Service*). Ten konkretny typ usługi daje klientowi możliwość stworzenia aplikacji i uruchomienia jej w chmurze. Rozbudowa aplikacji, utrzymanie, testowanie, a także wdrożenie, jest łatwe i szybkie. Dzięki Google App Engine twórca jest w stanie w kilka minut stworzyć serwer dostępny z dowolnego miejsca na świecie. Nie musi martwić się o jego niezawodność i dostępność. Może skupić się na stworzeniu aplikacji oraz jej właściwym działaniu, a nie na tym czy baza danych nie jest za mała, aby przechować określoną ilość informacji. Wykorzystanie silnika Google App Engine zapewnia skalowalność horyzontalną. Na początku z aplikacji korzysta nieduża ilość użytkowników. Jednak w momencie, gdy produkt zyskuje na popularności, wykorzystanie zasobów może znacząco wzrosnąć. Chmura obliczeniowa daje możliwość szybkiego i łatwego zwiększenia wykorzystywanych zasobów. Zarówno baza danych Datastore jak i sam serwer, dostosowane są do pracy z dużym obciążeniem. Najistotniejsze jest jednak to, że programiści, nie muszą martwić się kwestiami obciążenia serwerów i baz danych. Google App Engine robi to za nich. Można ten proces obserwować i reagować jeśli system działa inaczej, niż tego oczekujemy. Dodatkowo Google udostępnia wygodne narzędzie jakim jest

konsola administratora. Dzięki niej możemy obserwować jak działa nasz serwer oraz jaką ilość zasobów wykorzystuje. Google App Engine to elastyczne narzędzie jeżeli chodzi języki programowania. Mamy możliwość budowy silnika aplikacji internetowej w języku Java, Python, PHP lub Go. W przypadku tego projektu, współtworzonego z Pawłem Stępniem, jest to język Java.

4.1.2 Google Cloud Endpoint

Moduł Google Cloud Endpoint jest niezmiernie istotny dla integracji aplikacji mobilnej z silnikiem aplikacji internetowej. Jest to interfejs programistyczny aplikacji, który zawiera metody udostępniane dla klientów. Jako klientów mam na myśli aplikacje mobilne oraz przeglądarki internetowe. Google Cloud Endpoint dostarcza biblioteki i narzędzia, które umożliwiają proste generowanie nowych bibliotek dla aplikacji mobilnych. Przy integracji aplikacji mobilnej z chmurą obliczeniową, nie trzeba zajmować się implementacją komunikacji między tymi modułami. Udostępnione są nam gotowe moduły, opakowujące tę komunikację. Dzięki temu dużo łatwiej stworzyć cały system. Wystarczy że wygeneruje się bibliotekę, mając zaimplementowany interfejs programistyczny, a następnie zacznie się używać tej biblioteki po stronie mobilnej. Wymaga to oczywiście dodatkowej konfiguracji, ale nie różni się to znacząco od dodawania zwykłych bibliotek rozszerzających funkcjonalność programu.

Innym istotnym aspektem, który jest wykorzystany w Google Cloud Endpoint jest uwierzytelnianie użytkownika. Odbywa się ono w oparciu o protokół OAuth 2.0. Jest to bardzo rozbudowany protokół umożliwiający identyfikację użytkownika bez stałego podawania danych uwierzytelniających, z zapewnieniem poufności, integralności oraz dostępności. Cały proces uwierzytelniania odbywa się bez naszej ingerencji. My definiujemy tylko, które metody interfejsu programistycznego wymagać będą identyfikacji użytkownika.

Stworzenie programistycznego interfejsu aplikacji jest bardzo proste. W dokumentacji na temat Google Cloud Endpoint możemy dowiedzieć się jak to zrealizować. Poniżej przedstawione jest przykładowa definicja *API*.

```
@Api(name = "helloworld",
    version = "v1",
    scopes = {Constants.EMAIL_SCOPE},
```

```
    clientIds = {Constants.WEB_CLIENT_ID,
Constants.ANDROID_CLIENT_ID, Constants.IOS_CLIENT_ID},
    audiences = {Constants.ANDROID_AUDIENCE}
)
public class Greetings {}
```

Najważniejszym elementem jest użycie adnotacji Javy *@Api*. Wprowadza to nową definicję programistycznego interfejsu aplikacji. Istotne fragmenty definicji, to nazwa oraz stałe identyfikatory klientów. Przykładowo pole *Constants.ANDROID_CLIENT_ID* jest uzupełnione kluczem pobranym z konsoli do zarządzania projektem. Wcześniej musimy utworzyć taki projekt. Wszystkie te czynności wykonujemy po to, aby móc korzystać z funkcji naszego interfejsu w chmurze wyłącznie z naszej aplikacji. Klucz ten generowany jest z wykorzystaniem danych z kompilatora, którym kompilowana jest aplikacja mobilna.

4.1.3 Datastore

Datastore jest to baza danych z rodziny NoSQL. Modeluje ona dane w oparciu o obiekty, a nie o tabele. Dane powiązane są zależnością klucz, wartość. Architektura tego typu bazy danych pozwala na lepszą skalowalność horyzontalną. Datastore zapewnia również atomowość transakcji oraz spójność informacji w bazie danych. Dla programisty udostępniony jest dodatkowo interfejs Objectify. Pozwala on w prosty sposób tworzyć połączenie z bazą danych, modyfikować bazę oraz udostępniać dane w niej zawarte. Encje przechowywane w bazie danych Datastore, zawierają parametry, które pełnią analogiczną funkcję do pól w obiektach. Przy tworzeniu modelu danych istotne jest, abyśmy odpowiednio oznaczyli parametry, które będą pełniły funkcje kluczy, bądź parametrów wyszukiwania.

4.1.4 Memcache

Zadaniem modułu Memcache, podobnie jak bazy danych, jest przechowywanie informacji. Różnica między Memcache, a bazą danych, jest jednak taka, że moduł ten działa o wiele szybciej niż baza danych. Przechowuje on bowiem dane w pamięci. Używany jest on wtedy, kiedy dane potrzebne są natychmiastowo. Może znacząco przyspieszyć działanie aplikacji, szczególnie jeśli wykonujemy wielokrotne zapytania o te same dane, bądź potrzebujemy danych jak tylko pojawią się na serwerze. Musimy jednak wziąć pod uwagę

fakt, że dane z Memcache mogą być w każdej chwili usunięte. Nie możemy zatem bezgranicznie polegać na informacji zawartej w tym module.

Wykorzystanie modułu Memcache możemy przedstawić z użyciem następującego kodu.

Utworzenie instancji usługi Memcache:

```
    AsyncMemcacheService asyncCache =
MemcacheServiceFactory.getAsyncMemcacheService();
```

Ustawienie logowania zdarzeń przez aplikację:

```
     asyncCache.setErrorHandler(ErrorHandlers.getConsistentLogAndContinue
(Level.INFO));
```

Wydobycie obiektu i przechowanie go w sposób asynchroniczny:

```
    Future<Object> futureValue = value = (byte[]) futureValue.get();
    value = new Value();
    asyncCache.put(key, value);
```

Kilka prostych operacji pozwala nam wydobywać oraz zapisywać dane z użyciem Memcache. Jedyne co musimy zrobić aby poprawnie używać tego modułu to szczegółowe przeczytanie dokumentacji oraz wykorzystanie standardowych bibliotek z pakietu *com.google.appengine.api.memcache.*

4.2 Aplikacja mobilna

Aplikacja mobilna nie wykorzystuje zbyt wielu narzędzi, natomiast oparta jest o zestaw algorytmów dostosowanych do konkretnego problemu. Dogłębne zrozumienie działania głównych modułów aplikacji pozwala na lepszą implementację pomiaru biegu.

4.2.1 Moduł GPS

Moduł GPS jest najbardziej istotnym elementem całej aplikacji. Gdyby nie fakt, że w każdym nowym telefonie moduł ten jest dostępny, aplikacja nie mogłaby istnieć. Poprawny pomiar biegu jest warunkiem koniecznym do właściwego działania całej aplikacji. Aby zaimplementować pomiar biegu we właściwy sposób, należy najpierw dokładnie zrozumieć jak działa moduł GPS.

Moduł GPS w telefonie komórkowym wysyła sygnał do satelity. Satelita wysyła czas, w którym otrzymała sygnał. Różnica czasowa pozwala określić, jak daleko od satelity znajduje się nadawca sygnału. Biorąc pod uwagę informacje, które satelita wysyła dodatkowo, to znaczy pozycję na niebie w momencie odesłania sygnału, moduł GPS jest w stanie określić pozycję telefonu. Oczywiście jeśli mamy informację od jednego satelity, wiemy że znajdujemy się na okręgu z określonym promieniem otaczającym satelitę. Jeśli będziemy mieli kontakt z trzema satelitami, dostajemy informację o możliwych pozycjach dzięki trójstronności. Położenie ogranicza się wtedy do dwóch punktów. Czwarty satelita określa, w którym z tych dwóch punktów znajduje się telefon.

Inną istotną kwestią przy wykorzystaniu modułu GPS jest jego dokładność. Satelita ma wbudowany zegar atomowy, jednak odbiorniki w telefonach z oczywistych przyczyn takiego zegara nie posiadają. Dlatego w praktyce, dopiero około siedem satelitów pozwala poprawnie obliczyć naszą pozycję.

4.2.2 Krokomierz

Moduł krokomierza jest wymagany, aby zapobiegać oszustom, a tym samym zwiększyć wiarygodność aplikacji. Zastosowanie krokomierza umożliwia wykluczenie wpisów podejrzanych, które mogłyby wystąpić gdy użytkownik jechałby rowerem lub samochodem. Stosowanie takich rozwiązań jest naturalnym elementem w tego typu aplikacjach.

Oprogramowanie systemu Android w wersji 4.4 oraz nowsze posiada wsparcie do zupełnie nowych sensorów. Owe sensory to detektor kroków oraz licznik kroków. Muszą być wbudowane w urządzenie, ponieważ nie każdy telefon posiada możliwość ich wykorzystania. Aby użytkownicy starszych modeli mogli korzystać z krokomierza należy wykorzystać istniejące sensory oraz napisać odpowiednie oprogramowanie rozpoznające kroki. Krokomierz użyty jest w aplikacji również w celu poprawienia pomiaru biegu. Implementację ułatwia fakt, że użytkownik biegnie, można więc ustawić mniejszą czułość krokomierza. Kroki podczas biegu są łatwiejsze do odróżnienia od tych podczas zwykłego spaceru, ze względu na bardziej wyraźnie oddziaływanie na sensory. Jest więc duże prawdopodobieństwo, że krokomierz będzie perfekcyjnie mierzył kroki, a system będzie działał niezawodnie w wyjątkowych sytuacjach.

4.2.3 Aktywności

Podstawowym modułem wykorzystywanym podczas tworzenia aplikacji na system Android są aktywności. Jest to element, który posiada graficzny interfejs użytkownika. Pozwala on użytkownikowi na interakcję z aplikacją. Aktywności nie są ze sobą ściśle powiązane. Jedynym elementem wiążącym jest możliwość uruchomienia jednej aktywności z wewnątrz innej oraz przekazanie jej odpowiednich argumentów.

Do odpowiedniego użycia ważne jest zrozumienie cyklu życia aplikacji. Zapewnia on poprawne funkcjonowanie nie tylko projektowanej aplikacji, ale również całego telefonu. Dwie najważniejsze metody, które należy zaimplementować to *onCreate()* oraz *onPause()*. Definiuje się w nich kolejno co ma się wydarzyć, kiedy aktywność zostanie utworzona lub zatrzymana. Do tworzenia aktywności nie używa się konstruktorów. Metoda *onPause()* pozwala na implementację odpowiedniego zachowania aplikacji w sytuacji kiedy zostaje ona zatrzymana, na przykład poprzez uruchomienie innej aplikacji. Jest jeszcze kilka innych metod interfejsu aktywności, które zapewniają poprawne funkcjonowanie systemu.

Aby móc używać aktywności w aplikacji, należy dodać w pliku *AndroidManifest.xml* odpowiedni wpis. Plik ten jest głównym plikiem konfiguracyjnym aplikacji. Oprócz uprawnień, z jakimi aplikacja może korzystać z naszego telefonu, dodajemy w nim również informacje o dodatkowych bibliotekach, aktywnościach czy serwisach. Jeśli chcemy ustawić konkretną aktywność jako główną aktywność projektowanego systemu, również należy zrobić to w pliku *AndroidManifest.xml*.

Najważniejszym elementem aktywności pozostaje jednak odpowiednie zaimplementowanie funkcji związanych z cyklem życia aplikacji, tak aby na przykład po zamknięciu aplikacji nie było wycieków pamięci.

4.2.4 Serwisy Androida

Serwis jest to komponent aplikacji, który umożliwia uruchomienie operacji w tle. Zaletą serwisu jest możliwość bycia aktywnym nawet jeśli inna aplikacji jest uruchomiona na głównym ekranie telefonu. Serwisy nie udostępniają interfejsu użytkownika. Metody interfejsu tego komponentu wykorzystywane są aby przekazywać dane do aktywności w systemie Android. W projektowanej aplikacji serwisy wykorzystywane są między innymi do pomiarów biegów, ale również do ciągłej komunikacji z serwerem. Inny serwis pobiera

informacje o rezultatach naszego przeciwnika, a także wysyła rezultaty naszego biegu. Serwis może być wykorzystywany w dwóch formach, *Started* i *Bound.* Pierwszą formę wykorzystujemy do wykonania jednorazowej operacji, po której serwis się wyłączy. W tym przypadku nie wymagamy zwracania danych. Serwis taki może wysłać przekazane mu dane do sieci. Drugi typ serwisu, będący jednocześnie tym, który wykorzystywany jest w opisywanej aplikacji, to serwis *Bounded*. Ten typ serwisu wykorzystywany jest jeśli istnieje potrzeba powtarzania wielokrotnie danej operacji oraz potrzeba pobierania informacji o rezultatach operacji. Do komunikacji z serwisem wykorzystywany jest inny interfejs, *IBinder*. Przy pomocy obiektów implementujących ten interfejs, istnieje możliwość pobierania informacji z serwisu. Serwis działa tak długo, jak komponent z którym jest powiązany. Tym samym jeśli w przypadku zamknięcia aktywności w nieodpowiedni sposób, serwis będzie działał w tle aplikacji i wykonywał kosztowne dla procesora operacje.

5 Rozwiązania

W tym rozdziale chciałbym przedstawić wymagania aplikacji, zastosowane rozwiązania z użyciem konkretnych narzędzi oraz optymalizacje które zostały wprowadzone aby poprawić działanie całego systemu. W przypadku rozwiązań po stronie serwera opis nie jest szczegółowy, ponieważ implementacją tego modułu zajmował się Paweł Stępień.

5.1 Pobieranie informacji w czasie rzeczywistym

W chwili rozpoczęcia biegu przez dwie osoby, informacja o ich pozycji musi być jak najbardziej aktualna. Nie można wykluczyć, że dwie osoby, które korzystają z aplikacji, będą biegły obok siebie. Aplikacja powinna wtedy działać tak samo dla obu użytkowników. Taką możliwość daje nam połączenie silnika aplikacji Google App Engine, Memcache oraz Datastore.

W czasie biegu co pewien czas, stały i określony, telefon wysyła pozycję oraz czas w którym pokonany został dany dystans. Taka sama informacja, reprezentująca pozycję przeciwnika, może zostać pobrana z serwera. Tym samym istnieje możliwość poinformowania użytkownika, czy znajduje się na czele, czy przegrywa.

Wymianę informacji umożliwia Memcache oraz Datastore. Wysyłając paczkę danych do serwera, zapisywane są one zarówno w pamięci Memcache jak i w bazie danych Datastore. Dzieje się tak z dwóch powodów. Moduł Memcache wykorzystany jest ponieważ pamięć operacyjna działa dużo szybciej. Dlatego jeśli dane zostaną wysłane na serwer, a moduł Memcache nie zdecydował się ich usunąć zaraz po aktualizacji, przeciwnik może pobrać dane z tego modułu. Natomiast jeśli Memcache jest bardzo obciążone i zmuszone było usunąć informacje o pozycji użytkownika, kosztem informacji o pozycji innej pary biegaczy, to dane nadal można pobrać z bazy danych. Jest to dużo wolniejsze, jednak poprzez wykorzystanie bazy danych można być pewnym, że przeciwnik zawsze otrzyma informację o pozycji drugiego biegacza. W idealnym przypadku kiedy bieg trwa krótko, a serwer nie jest obciążony, dane będą wydobywane z pamięci serwera, a cała aplikacja będzie działać bardzo szybko.

5.2 Komunikacja z serwerem

Połączenie z internetem wymagane jest w aplikacji z kilku powodów. Przede wszystkim aby korzystać z aplikacji użytkownik musi być zalogowany. Dzięki temu może rywalizować ze znajomymi. Podczas każdego uruchomienia aplikacji sprawdzana jest również jej wersja. Posiadanie najnowszej wersji aplikacji umożliwia sprawiedliwą rywalizację pomiędzy użytkownikami. Połączenie z siecią jest również wymagane, aby znaleźć losowego przeciwnika. Niezależnie od tego czy uczestnik rywalizuje z obcą osobą czy ze znajomym.

Pobieranie wyniku przeciwnika oraz aktualizacja rezultatu użytkownika również jest wykonywana poprzez połączenie sieciowe. W tym przypadku istnieje możliwość zastosowania optymalizacji. Pobierając rezultaty przeciwnika, można tymczasowo zapisywać te dane na telefonie. Następnie korzystając z tych informacji można wyliczyć jego prędkość biegu, a następnie przewidzieć z pewną dokładnością jaki będzie kolejny rezultat przeciwnika. Pozwala to na zmniejszenie wykorzystania transmisji danych przez połączenie internetowe w aplikacji. Dodatkowo, jeśli połączenie z siecią zostałoby zerwane, urządzenie jest w stanie podawać szacowany rezultat biegu przeciwnika. Aplikacja nie może jednak bezgranicznie polegać na informacji obliczonej na podstawie historii poszczególnych wyników. Nie jest wykluczone, że przeciwnik zatrzyma się, w tym przypadku przewidując jego pozycję, użytkownika wprowadzony zostanie w błąd. Z tego powodu implementacja takiej funkcjonalności musi jedynie wspierać działanie podstawowych modułów.

W opisywanej aplikacji wybrano przesyłanie danych przy pomocy formatu *JSON*. Inne możliwości to użycie formatu *XML*. *XML* to według tłumaczenia „*Rozszerzalny język znaczników*”. Celem tego formatu jest możliwość utworzenia sposobu prostej wymiany danych w oparciu o określoną strukturę pliku. Tym sposobem istnieje możliwość przekazania danych, jeżeli odbiorca zna strukturę pliku. Posługując się znacznikami można wyciągnąć konkretne dane. Przykładowy plik w tym formacie ma następującą strukturę.

```
<przedmiot>
   <id>56</id>
   <nazwa>komputer</nazwa>
</przedmiot>
```

Posiadając bibliotekę do przetwarzania plików w formacie XML, można w prosty sposób wydobyć przedmiot o numerze identyfikacyjnym 56. Wadą tego rozwiązania jest redundancja informacji. Należy stworzyć znacznik zamykający dla każdego znacznika otwierającego. Nie dość, że informacje w takim pliku czyta się mało wygodnie, to jest konieczność przesyłania przez sieć większej ilości informacji, przez co rozwiązanie to jest wolniejsze, niż na przykład format *JSON*.

Nazwa JSON jest akronimem Javascript Object Notation. Tłumacząc na język polski jest to notacja bazująca na języku *Javascript*, oparta o obiekty. Podejście obiektowe pozwala na oszczędzenie ilości informacji transportowanych przez sieć, a także ułatwia analizę danych w tym formacie. Przekazywane dane wyglądają następująco.

```
{"przedmiot": {
   {„id": „56"}
}}
```

Można zauważyć, że dane przechowywane są jako tekst. Podobnie jak w przypadku plików *XML*, wystarczy że skorzystać z biblioteki do przetwarzania plików w formacie *JSON*, aby wydobyć konkretne dane.

5.3 Pomiar biegu

Mierzenie biegu użytkownika aplikacji odbywa się na podstawie informacji o jego pozycji. W systemie Android można to zrealizować na dwa sposoby. Jednym sposobem jest wykorzystanie systemu globalnego pozycjonowania, drugi to wykorzystanie wież telekomunikacyjnych. Wybierając najlepsze rozwiązanie należy wziąć pod uwagę kilka czynników. Po pierwsze każdy sposób ma zarówno wady jak i zalety. Istotne jest dostosowanie najlepszego rozwiązania do problemu. Trzeba również wziąć pod uwagę działanie danego rozwiązania, jeśli użytkownik porusza się. Ważne jest również zapewnienie odpowiedniej dokładności pomiarów.

Jedną z możliwości jest wykorzystanie wież telekomunikacyjnych oraz połączeń z odbiornikami Wi-Fi. Zaletą tego rozwiązania, jest działanie zarówno w pomieszczeniach jak i na zewnątrz. W prezentowanej aplikacji nie jest jednak istotne działanie w budynkach, ponieważ osoby do których skierowana jest aplikacja, to osoby biegające poza domem. Dużą zaletą pobierania lokalizacji z odbiorników Wi-Fi oraz wież telekomunikacyjnych jest

szybsze działanie tego rozwiązania, szczególnie w porównaniu do modułu GPS. Zmniejsza się również pobór baterii co może być bardzo istotne dla użytkowników. Telefony w dzisiejszych czasach potrafią mieć aktywne kilka aplikacji jednocześnie. Ilość wykonywanych działań przez aplikacje oraz duże wyświetlacze, sprawiają że sukcesem jest jeśli telefon wytrzymuje więcej niż jeden dzień, będąc cały czas aktywnym.

Drugie rozwiązanie to wykorzystanie modułu GPS. Właśnie to rozwiązanie zostało zastosowane w opisywanej aplikacji. Moduł GPS zapewnia większą dokładność określenia lokalizacji. W porównaniu do poprzedniego sposobu, rozwiązanie to ma kilka wad. Określenie lokalizacji odbywa się dużo wolniej, ponieważ urządzenie pobiera tę informację z satelity. Satelity znajdują się na orbitach ziemskich. Odbiorniki sieci telekomunikacyjnych znajdować się mogą kilka kilometrów od urządzenia. Wykorzystanie modułu GPS bardziej obciąża baterię w telefonie. Biorąc pod uwagę również pogodę, również gorzej radzi sobie system globalnego pozycjonowania. Jeżeli występuje duże zachmurzenie, urządzenie może mieć też problem z dokładnym określeniem lokalizacji. Największą zaletą globalnego systemu pozycjonowania jest jego precyzja. Zakładając, że użytkownik przemieszcza się, można zdefiniować pozycję użytkownika z dokładnością do około 6 metrów na 100 metrów. Jest to główny powód, dla którego użyto modułu GPS do pomiaru biegu w projektowanej aplikacji. Jednym z głównych celów projektu było zapewnienie użytkownika, że jego wyniki podczas rywalizacji z innymi osobami nie będą dziełem przypadku lub błędu pomiaru. Jeżeli byłaby potrzeba zmniejszenia dokładności, zyskując mniejszą konsumpcję baterii, zawsze można rozwinąć aplikację o taką funkcjonalność.

Niezależnie od wyboru sposobu pomiaru charakterystyka tego pomiaru nie powinna różnić się od zwykłego biegu. To znaczy jeżeli użytkownik określi, że chce brać udział w biegu na tysiąc metrów, to aplikacja powinna przerwać pomiar jeśli przebiegnie on dokładnie tysiąc metrów. Nie mniej, nie więcej. Działanie menedżera lokalizacji w telefonie z systemem Android charakteryzuje się tym, że najpierw tworzona jest instancja tego obiektu, przekazując mu określone parametry. Te parametry to między innymi informacja co jaką odległość powinno otrzymywać się aktualizację pozycji. Jeśli ustawi się tutaj wartość *n* metrów, to gdy nowa lokalizacja jest *n* metrów dalej niż poprzednio pobrana lokalizacja, otrzymamy aktualizację pozycji. Ustawiając tą wartość na 0, można narazić się na problem niedokładności modułu GPS. Ponieważ akceptowane będą każde zmiany

pozycji, stojąc w miejscu, naliczone zostaną przebyte metry, których użytkownik w rzeczywistości nie przebył. Jeśli ustawiona zostanie wartość n=20 metrów, dopiero po zmianie pozycji o tą odległość wystąpi aktualizacja od modułu GPS. Wydawałoby się, że w tym miejscu wystarczy przekazać taką wartość przy konstruowaniu obiektu i niczym więcej się nie martwić. Istnieje jednak problem zakończenia mierzenia po ustalonym wcześniej dystansie. Tak jak ostało wspomniane, jeśli użytkownik wprowadzi w aplikacji chęć przebiegnięcia dystansu 1000 metrów, to nie chce biec więcej. Z tego powodu idealnym rozwiązaniem jest zmniejszanie dokładności wraz ze zbliżaniem się do mety. W związku z powyższym w projektowanej aplikacji przyjęto początkową aktualizację pozycji po przebyciu 25 metrów, natomiast na ostatnich 5 procentach dystansu, rozpoczyna się zmniejszanie tej wartość o 33 procent w każdej iteracji. Obniżanie tej wartości zatrzymywane jest jeśli jest ona mniejsza niż 5. Tym sposobem użytkownik musi przebiec maksymalnie 5 metrów więcej niż określił to przed biegiem. Ostatecznie aproksymowane są ostatnie metry biegu i aplikacja zwraca czas pokonania całego dystansu. W przypadku wykorzystania gotowego rozwiązania systemu Android, należałoby ponownie utworzyć obiekt odpowiedzialny za aktualizację pozycji. Obiekt ten następnie musiałby nawiązać nowe połączenie z satelitami, co nie jest rozwiązaniem najbardziej optymalnym. Generuje ono także nowe problemy. Dlatego zdecydowano się na własny algorytm pobierania aktualizacji.

Innym ważnym elementem jest akceptowalna dokładność (niedokładność) pomiaru i powiązanie tego parametru z aktualizacją pozycji co określoną odległość. Używając standardowych bibliotek Androida wystarczy podać maksymalną niedokładność która akceptowana jest podczas definiowania lokalizacji. Jednak jeżeli implementujemy pomiar w sposób opisany powyżej, nie jest to już takie proste. Należy wziąć pod uwagę sytuację, w której urządzenie nie porusza się, natomiast moduł GPS przekazuje nowe dane. Można zwizualizować sobie ten problem na podstawie następującej sytuacji. Urządzenie znajduje się w danym punkcie *X*. Akceptowana jest niedokładność do 12 metrów, a pozycja aktualizowana jest co 15 przebytych metrów. Założenie jest takie, że informacja o nowej pozycji otrzymana będzie w odległości 9 metrów na północ od punktu *X* z niedokładnością 10 metrów. Pomimo że urządzenie nie zmieniło pozycji, jest to akceptowalna zmiana pod względem dokładności. Jednak urządzenie nie przebyło 15 metrów, dlatego odrzucana jest

ta informacja. Kolejna aktualizacja modułu GPS to 9 metrów na południe od punktu *X*. Jest to analogiczna sytuacja do poprzedniej. Ta zmiana pozycji również nie powinna być pobrana. Zaskakujący jest jednak fakt, że system Android weźmie pod uwagę tę aktualizację pozycji. Dlaczego tak się dzieje? Mianowicie przez to, że sprawdzając przebyty dystans akceptowane są wszystkie pozycje (pomijając te niedokładne). Dlatego poprzednia informacja, którą pobrał system Android jest taka, że urządzenie było 9 metrów na północy od punktu *X*, teraz jest 9 metrów na południe od punktu *X*, dlatego razem jest 18 metrów, co jest akceptowalne przez wprowadzone ograniczenia. Trzeba więc obliczać przebytą odległość od ostatniej prawidłowej pozycji.

5.4 Pomiar biegu w warunkach zmiennej pogody

Kolejnym wymaganiem stawianym aplikacji, jest działanie w każdych warunkach pogodowych. Jest to istotny podpunkt, ponieważ nie można zakładać, że aplikacja będzie uruchomiona przy idealnej pogodzie. A to właśnie warunki atmosferyczne, mają duży wpływ na dokładność określania pozycji użytkownika.

Jedno z rozwiązań, które można zastosować, to wykorzystanie kilku poprzednich pomiarów, do określenia kolejnej pozycji użytkownika. Podobne rozwiązanie dotyczy przewidywania rezultatu przeciwnika na podstawie jego poprzednich pozycji i obliczonej prędkości. Algorytm ten można także zastosować dla własnego biegu. Różnicą jest fakt, że w tym miejscu nie ma ograniczenia wykorzystania łącza internetowego, tylko zużycie baterii. Podobnie jak w przypadku przewidywania pozycji przeciwnika, ostatecznie również konieczne jest poleganie na pomiarze GPS. Można zastosować też zupełnie inne, niekonwencjonalne rozwiązanie.

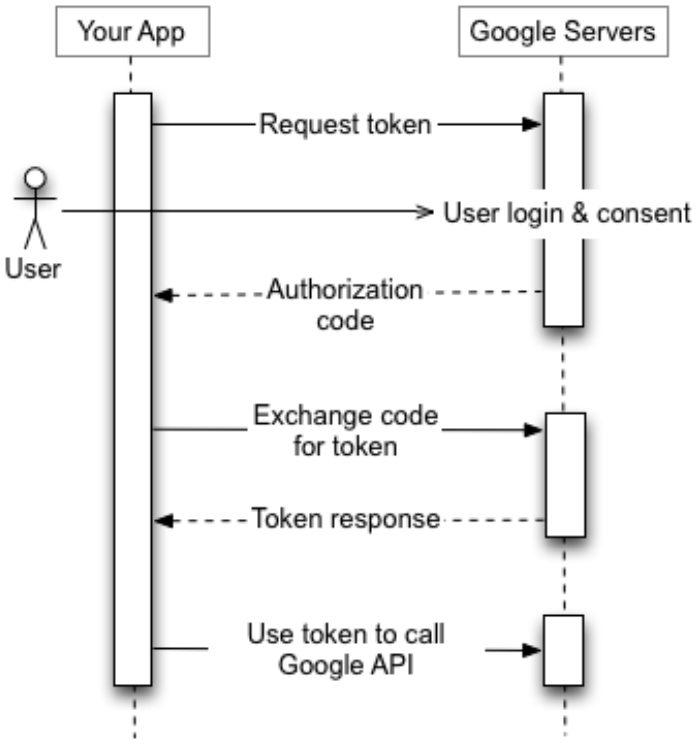
W wielu aplikacjach które mierzą bieg, kładziony jest nacisk na udostępnienie użytkownikowi jego trasy biegu. Dlatego stale potrzebna jest informacja o pozycji użytkownika na mapie świata. Dodatkowo aplikacje te często nie zapobiegają oszustom (nie wykorzystują krokomierza), ponieważ nie skupiają się na rywalizacji. Unikalnym rozwiązaniem jest połączenie danych pozyskanych z krokomierza, z danymi odbieranymi od modułu GPS. Jak miałoby to działać? Załóżmy ze poprzednie 50 metrów trasy pokonane zostało z najlepszą możliwą dokładnością. Wiadomo również, że krokomierz wyliczył 75 kroków. Prosty rachunek pozwala na wyliczenie, że użytkownik stawia krok o długości 0,66

metra w biegu. Należy pamiętać, że długość kroku użytkownika może się zmieniać w zależności od tego jak szybko biegnie. Wykorzystując jednak prosty algorytm na krótkim odcinku, można poprawić funkcjonowanie pomiaru biegu, a co za tym idzie całej aplikacji. Jeżeli wystąpi gorsza pogoda, również można zastosować to rozwiązanie. W idealnej sytuacji, zakładając perfekcyjne działanie owego algorytmu w rozbudowanej wersji, można mierzyć cały bieg z wykorzystaniem informacji od krokomierza. Ograniczałoby to wykorzystanie modułu GPS oraz oszczędzałoby konsumpcję baterii. Tak rozbudowaną wersję algorytmu należałoby jednak dogłębniej przeanalizować. Możliwe że wymagana byłaby większa baza z charakterystykami biegów użytkownika, informacja o jego wzroście, wadze i tym podobne. Wymagane byłyby również rozbudowane testy. Niemniej jednak, nawet w podstawowej wersji taki algorytm mógłby usprawnić działanie pomiaru biegu.

5.5 Uwierzytelnianie

Jeśli użytkownik chciałby ścigać się z obcą osobą, uwierzytelnianie nie byłoby wymagane. Jednak w sytuacji kiedy chciałby rywalizować ze znajomymi, system musi mieć możliwość identyfikacji użytkowników. Korzystając z aplikacji na system operacyjny Android użytkownik jest zmuszony do założenia konta w sklepie Google Play, aby móc pobrać aplikacje. Używane jest do tego konto w domenie pocztowej gmail. Dużym ułatwieniem dla programisty jest wykorzystanie w procesie uwierzytelniania protokołu OAuth 2.0.

W przedstawianym systemie również stosowane jest to rozwiązanie. Protokół OAuth zapewnia uwierzytelnianie użytkownika poprzez wymianę kluczy zwanych tokenami. Tokeny mają określony czas życia, a cały proces uwierzytelniania odbywa się ze spełnieniem konkretnych zasad, opisanych w dokumencie *„RFC 6749 - The OAuth 2.0 Authorization Framework”*.



Rysunek 5 Schemat autoryzacji [źródło: https://cloud.google.com/appengine/docs/python/oauth/].

Na rysunku nr 5 przedstawiony jest przebieg komunikacji z serwerem. Na początku klient musi wysłać żądanie uwierzytelniające do systemu, który ma uprawnienia do zapewniania dostępu do zasobów. Po otrzymaniu pozwolenia na ubieganie się o dostęp do zasobów, klient wysyła prośbę do serwera autoryzacyjnego o token, czyli unikatowy identyfikator. Po otrzymaniu tokenu, jest on używany przez klienta do korzystania z zasobów na danym serwerze.

W projektowanym systemie proces uwierzytelniania wykonywany jest bez ingerencji systemu Android, użytkownik musi jedynie zalogować się na konto powiązane ze sklepem Google Play. Wystarczy, że raz zaloguje się na swoje konto, natomiast cały proces wymiany informacji i uzyskiwania dostępu do określonych funkcji odbywa się bez jego ingerencji. To sprawia, że użytkownik nie musi za każdym razem podawać loginu i hasła po włączeniu aplikacji. Wystarczy że będzie zalogowany w systemie na konto, które wcześniej powiązane było z aplikacją.

Po stronie serwera Google App Engine również nie trzeba robić zbyt wiele, aby ograniczyć dostęp uwierzytelnionym użytkownikom. Wystarczy dodać pole *User* w sygnaturze metody. Google App Engine sam zadba o to, aby sprawdzić czy używany

token jest prawidłowy oraz stworzy obiekt, który będzie przechowywał adres email oraz kilka dodatkowych informacji o koncie użytkownika. Jeśli proces uwierzytelniania nie przejdzie pomyślnie, obiekt klasy *User* będzie miał wartość *null*, a metoda zwróci odpowiedni wyjątek.

6 Implementacja funkcjonalności

Rozdział zawiera szczegółowy opis implementację projektu. Na początku opisane są rozwiązania, które pozwalają aplikacji działać w środowisku wielowątkowym. Jest to bardzo istotna problematyka dla przedstawionego projektu. Następnie opisany jest sposób w jaki aplikacja mobilna komunikuje się z serwerem Google App Engine. Na końcu znajduje się wyjaśnienie, dotyczące obsługi rywalizacji biegowej w czasie rzeczywistym.

6.1 Współbieżność

Programując aplikacje na platformę Android można w pełni wykorzystać zalety współbieżności. Już od pierwszego kroku wykorzystywane jest to w omawianej aplikacji. W momencie gdy użytkownik chce się zalogować do aplikacji, nie należy blokować całego telefonu. Aplikacja powinna wysłać żądanie do serwera, a w momencie otrzymania odpowiedzi przejść do kolejnej aktywności. Wyświetlając kolejną aktywność, w tym przypadku jest to okno wyboru typu przeciwnika, również wykorzystywana jest współbieżności. Zaimplementowany jest bowiem oddzielny wątek odpowiedzialny za sprawdzanie połączenia z modułem GPS oraz kontrolujący połączenie z internetem. W opisywanej aplikacji wykorzystywane są dwa mechanizmy, bez których jej działanie nie byłoby możliwe.

Po pierwsze wykorzystywane są zadania asynchroniczne, czyli klasa *AsyncTask*. W oficjalnej dokumentacji systemu Android można przeczytać, że klasa ta umożliwia wykonywanie operacji w tle, bez ingerencji w interfejs użytkownika. Powinna być ona wykorzystywana do wykonywania krótkich operacji, na przykład do pobierania danych z internetu. Tworząc klasę dziedziczącą po *AsyncTask* jest możliwość wykorzystania metody *onPostExecute*. Jest ona wywoływana w momencie, kiedy aplikacja zakończy wykonywanie zadania w głównej metodzie (*doInBackground*). Dzięki temu aplikacja jest w stanie odpowiednio zareagować po zakończeniu zadania. W zaprojektowanej aplikacji wykorzystywane jest to do rozpoczęcia biegu lub zwrócenia informacji o nieudanej próbie utworzenia biegu. Między poszczególnymi metodami, które nadpisywane są w klasie dziedziczącej, można w prosty sposób przekazywać argumenty. Wystarczy, że każda funkcja będzie zwracać określoną wartość, a zwracany typ będzie zgodny z generycznym

typem podanym wcześniej przy definicji zadania asynchronicznego. Wykorzystanie zadań asynchronicznych staje się bardziej skomplikowane w przypadku chęci wykorzystania metody z klasy *Activity*, czyli z podstawowej aktywności, działającej w głównym wątku aplikacji.

Klasa *Activity* ma wiele metod, które pozwalają zarządzać interfejsem użytkownika oraz aktywnościami. Z wewnątrz zadania asynchronicznego, nie można bezpośrednio uruchomić kolejnej aktywności, jest to bowiem ingerencja w główny wątek aplikacji z wątku pobocznego. W opisywanej aplikacji, aby rozpocząć bieg ze znajomym użytkownik musi być zalogowany oraz wysłać żądanie do serwera *hostRunWithFriend*. Żądanie to zwraca:

- wartość -1 w przypadku błędu serwera,
- 0 w przypadku nieudanej próby rozpoczęcia biegu,
- wartość dodatnią w przypadku sukcesu.

Wartość dodatnia to ilość sekund, po których można rozpocząć bieg. W aktywności od razu po otrzymaniu odpowiedzi od serwera (wartości większej niż 0) rozpoczynana jest kolejna aktywność, czyli bieg z losową osobą. Aby było to możliwe, wykorzystany został mechanizm znany z innych języków obiektowych, na przykład C#, czyli delegatów. Przed wykonaniem zadania asynchronicznego przypisywany jest obiekt klasy do pola w tym zadaniu. Następnie po wykonaniu operacji w tle, wykonywana jest funkcja *onPostExecute*, w której wywoływana jest metoda należącą do przekazanego wcześniej delegata.

Implementacja wygląda następująco:

```
private class HostForFriendTask extends AsyncTask<Void, Void,
Boolean>  {

        public AsyncTaskResponse<Boolean> delegate = null;

        @Override

        protected Boolean doInBackground(Void... params) {

            String friendsLogin =
friendsLoginText.getText().toString();

            boolean runCreated = false;

            try {

                runCreated =
theIntegrationLayer.hostRunWithFriend(friendsLogin, preferences);

            } catch (Exception e) {
```

```
                e.printStackTrace();
                Logger.getAnonymousLogger().log(Level.SEVERE,
e.getMessage(), e);
            }
            return runCreated;
        }
        @Override
        protected void onPostExecute(Boolean runCreated) {
            super.onPostExecute(runCreated);
            delegate.onTaskFinish(runCreated);
        }
    }
    private class HostForFriendResponse implements
AsyncTaskResponse<Boolean> {
        @Override
        public void onTaskFinish(Boolean runCreated) {
            if (runCreated) {
                try {
                    waitForFriendToJoinAndStartRun();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

Klasa *AsyncTaskResponse* jest to interfejs, który zawiera metodę *onTaskFinish*. Tym samym w prosty i przejrzysty sposób można zarządzać aktywnościami i komunikacją z serwerem.

Powyższy mechanizm był jednym z dwóch, które wykorzystano w aplikacji do prawidłowego zarządzania współbieżnością. Kolejnym istotnym mechanizmem są operacje na wątkach. Wykorzystując klasę *Thread* można tworzyć wątki. Grupy wątków pozwalają zarządzać zadaniami wykonywanymi jednocześnie.

W wydaniu Javy w wersji 1.5 dodany został pakiet *java.util.concurrent*. W książce autorstwa Joshua Bloch pt. *Clean Code: A Handbook of Agile Software Craftsmanship,*

Robert C. Martin można przeczytać o szkielecie *Executor Framework* i prawidłowych praktykach jego wykorzystania. Opis jest następujący:

„*In release 1.5, java.util.concurrent was added to the Java platform. This package contains an Executor Framework, which is a flexible interface-based task execution facility. Creating a work queue that is better in every way than the one in the first edition of this book requires but a single line of code: ExecutorService executor = Executors.newSingleThreadExecutor(). Here is how to submit a runnable for execution: executor.execute(runnable);”.*

W powyższym fragmencie można przeczytać, że szkielet *Executor* jest elastycznym narzędziem bazującym na interfejsach, stworzonym do zarządzania wątkami. Tworzenie nowego wątku z jego wykorzystaniem jest dużo łatwiejsze, niż niskopoziomowe techniki. Fragment przedstawia też sposób tworzenia nowego wątku. Najpierw należy stworzyć nowy obiekt z klasy *ExecutorService*, a następnie wywołać metodę *execute,* przekazując nasz wątek jako parametr. Zagłębiając się w dokumentację można dowiedzieć się w jakiś sposób komunikować się pomiędzy wątkami oraz w jaki sposób kończyć pracę wątków. Jest to nie tylko dużo łatwiejsze niż operowanie bezpośrednio na wątkach, ale także bezpieczniejsze dla stabilności aplikacji.

Interfejs *Executor* zdejmuje wiele obowiązków z twórców aplikacji. Do tego należy do biblioteki Javy, więc można być pewnym, że nad tą biblioteką pracowało wielu doświadczonych programistów. Wykorzystanie tej biblioteki daje dużo większą szansę na poprawne działanie aplikacji. W przedstawianym w tej pracy programie wykorzystano opisane techniki. Do tej samej biblioteki należy jeszcze jeden typ zadań wykonywanych współbieżnie. Jest to klasa *ScheduledThreadPoolExecutor*. Zastępuje ona klasę *Timer,* z pakietu *java.util*. W tej samej książce (*Clean Code: A Handbook of Agile Software Craftsmanship, Robert C. Martin*) można znaleźć informacje o powodach, dla których wykorzystanie klasy z pakietu *java.util.concurrent* jest lepsze niż korzystanie ze zwykłego obiektu klasy *Timer*. Są one następujące:

„*While it is easier to use a timer, a scheduled thread pool executor is much more flexible. A timer uses only a single thread for task execution, which can hurt timing accuracy in the presence of longrunning tasks. If a timer’s sole thread throws an uncaught exception, the*

timer ceases to operate. A scheduled thread pool executor supports multiple threads and recovers gracefully from tasks that throw unchecked exceptions."

Autor stwierdza, że łatwiej jest użyć klasy *Timer*, jednak klasa *ScheduledThreadPoolExecutor* jest bardziej niezawodna. Pierwszym powodem jest jej dokładność podczas zadań wykonujących się dłuższy czas. Dzieje się tak, ponieważ klasa ta wykorzystuje wiele wątków. Drugim powodem jest obsługa wyjątków. Mianowicie obiekt klasy *Timer* zakończy swoje działanie w sytuacji wystąpienia wyjątku i zwróci błąd, natomiast obiekt klasy *ScheduledThreadPoolExecutor* dokończy swoje działanie.

Zarządzanie współbieżnością jest skomplikowane w dużych i nowoczesnych aplikacjach, jednak istnieje wiele mechanizmów, które ułatwiają zarządzanie wątkami. Programista musi tylko wiedzieć, którego narzędzia i w jaki sposób użyć.

6.2 Integracja z serwerem

W tym podrozdziale przedstawiony jest sposób w jaki zachodzi integracja pomiędzy aplikacją mobilną, a serwerem aplikacji.

Po pierwsze należy utworzyć programistyczny interfejs aplikacji, za pomocą Google Cloud Endpoint. Szczegóły opisane zostały to w rozdziale Google Cloud Endpoint. Do komunikacji z serwerem wykorzystano następujące biblioteki:

- *com.google.api.client.extensions.android.http.AndroidHttp*
- *com.google.api.client.extensions.android.json.AndroidJsonFactory*
- *com.google.api.client.http.HttpTransport*
- *com.google.api.client.json.JsonFactory*

Jak widać po ścieżce z importem bibliotek, wszystkie pochodzą z pakietu com.google.api.client.

Aby integracja z serwerem zachodziła poprawnie, należy poznać szczegóły funkcjonowania tych bibliotek. W tym rozdziale zamieszczony jest również kod z aplikacji, który pokazuje jak wygląda klasa odpowiedzialna za konfigurację połączenia z serwerem.

```
public class EndpointBuilder {
   public static final JsonFactory JSON_FACTORY =
       new AndroidJsonFactory();
```

```
    public static final HttpTransport HTTP_TRANSPORT =
        AndroidHttp.newCompatibleTransport();
    public static SitAndRunApi getApiServiceHandle() {
        SitAndRunApi.Builder helloWorld = new SitAndRunApi.Builder(
            HTTP_TRANSPORT, JSON_FACTORY, null);
        helloWorld.setRootUrl("http://localhost:8080/_ah/api/");
        return helloWorld.build();
    }
}
```

Tworzenie instancji obiektu jest następujące:

```
private SitAndRunApi endpointAPI;
public TheIntegrationLayer() {
    endpointAPI = EndpointBuilder.getApiServiceHandle();
}
```

Idąc zgodnie z kodem aplikacji, najpierw tworzona jest definicja klasy *EndpointBuilder*. Jest to klasa zawierająca wyłącznie metody statyczne, nie będzie bowiem potrzeby towrzenia instancji tej klasy. Następnie definiowane są statyczne finalne obiekty, które przekazywane są do metody *Builder* z klasy *SitAndRunApi.* Klasa ta pochodzi z interfejsu programistycznego i została wygenerowana z użyciem narzędzia do automatyzacji tworzenia projektów *Maven*. Pierwszym parametrem metody *Builder* jest klasa używana do transportu informacji przez sieć. W naszym przypadku jest to klasa do komunikacji z użyciem protokołu *HTTP*. Kolejnym parametrem jest format transportu informacji.

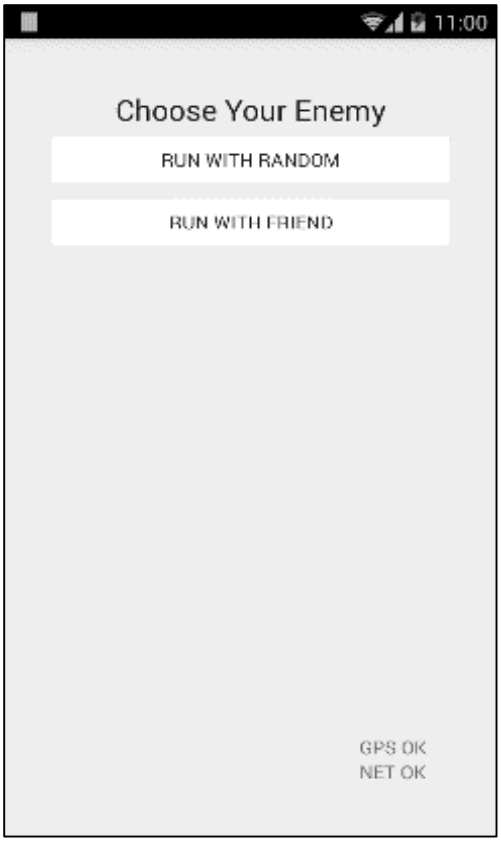
6.3 Obsługa biegu

Rozdział ten podzielony jest na kilka podrozdziałów, który reprezentują cykl życia aplikacji od wyboru przeciwnika, poprzez dwa dostępne typy rywalizacji, kończąc na właściwym biegu z przeciwnikiem. W rozdziale zawarte są również najważniejsze fragmentu kodu aplikacji, odpowiedzialne za realizację poszczególnych funkcjonalności związanych z biegiem.

6.3.1 Wybór przeciwnika

Aplikacja składa się z dwóch głównych aktywności, które implementują obsługę biegu. Każda z nich wykorzystuje odpowiednią funkcję z klasy *TheIntegrationLayer*.

Na początku użytkownik widzi okno aktywności *EnemyPickerActivity*. Jak wskazuje nazwa, może on wybrać rodzaj biegu. Dostępne są dwa typy rywalizacji. Jeden z nich to bieg ze znajomym, drugi to bieg z obcą osobą. Aktywność umożliwiająca wybór przeciwnika pojawia się natychmiast po zalogowaniu do aplikacji. Wygląd przedstawiony jest na rysunku nr 6.



Rysunek 6 Ekran wyboru przeciwnika

Jak wcześniej wspomniano, w projekcie skupiono się na intuicyjności aplikacji, tak aby użytkownik nie musiał się zastanawiać co oznacza dany przycisk i jaką czynność powinien w danej sytuacji wykonać. Dlatego też po zalogowaniu, użytkownik widzi jedynie dwa przyciski, *RUN WITH RANDOM* oraz *RUN WITH FRIEND*. Na dole ekranu widoczna jest też informacja o połączeniu z modułem GPS oraz internetem. Jeśli oba wskaźniki nie będą ustawione na wartość *OK*, użytkownik nie będzie w stanie uruchomić biegu. Jak opisano w

rozdziale 5, w aplikacji wymagane jest połączenie z internetem, aby móc rywalizować z przeciwnikiem. Moduł GPS jest potrzebny do pomiarów biegu. Wymagane jest poprawne połączenie z tymi modułami.

Informacja o połączeniu z modułem GPS zwraca jeden z trzech możliwych rezultatów. Po pierwsze moduł GPS może być wyłączony, wtedy na ekranie widoczna jest informacja *GPS OFF***.** Jeśli moduł ten jest włączony, są dwie możliwości. Połączenie jest prawidłowe (*OK*) lub nieprawidłowe (*BAD*). Decyzję o tym, czy połączenie jest poprawne, aplikacja podejmuje na podstawie dokładności wskazania lokalizacji. Jeśli dokładność jest mniejsza lub równa 25 metrów, wtedy uznaje się połączenie za prawidłowe. Taka dokładność odpowiada połączeniu z około 7 satelitami. Podczas biegu akceptowalna dokładność to około 32 metrów. Różnica między ograniczeniami wynika z dwóch czynników. Po pierwsze aby wykluczyć ewentualne wahania wskazań modułu GPS, dlatego uwzględniono różnicę kilku metrów między wstępną dokładnością, a końcową dokładnością. Po drugie, w sytuacji kiedy użytkownik jest w miejscu, wskazanie to powinno być dokładniejsze, niż kiedy jest w biegu. Dokładność tą można oczywiście zmienić. Jeśli użytkownik ma słabe połączenie z modułem GPS aplikacja powinna obniżyć próg dokładności po pewnym czasie. Nie należy bowiem uniemożliwić biegu użytkownikom ze słabszymi urządzeniami. Zmiana ta powinna się jednak odbywać po pewnym czasie i z uwzględnieniem pewnej wartości granicznej, poniżej której użytkownik nie może rozpocząć biegu.

Do sprawdzania połączenie z modułem GPS, wykorzystany jest interfejs *LocationListener*. Wymaga on zaimplementowania metody *onLocationChanged*. W tej metodzie, przy zmianie lokalizacji, można pobrać dokładność pomiaru.

```
@Override
public void onLocationChanged(Location location)
{  accuracy = location.getAccuracy();
   Toast.makeText(this, "Accuracy: " + accuracy,
   Toast.LENGTH_SHORT).show();
}
```

Wydaje się, że jest to bardzo prosta funkcja. Należy jednak wiedzieć, który interfejs wykorzystać. Do tego ważne jest użycie odpowiedniej funkcji z tego interfejsu oraz pola

obiektu klasy *Location*. Jeden z błędów, na który natknięto się podczas testowania aplikacji, powiązany był ze zwracaną wartością przez metodę *getAccuracy*. Aplikacja pozwalała użytkownikowi rozpocząć bieg w momencie, kiedy dokładność była mniejsza niż 25 metrów. W dokumentacji na temat wymienionej metody można przeczytać ciekawe informacje. Dowiadując się co tak naprawdę oznacza dokładność pobranej lokalizacji, otrzymuje się informację, że wartość długości i szerokości geograficznej jest z pewną dozą prawdopodobieństwa w środku koła o promieniu określonym przez wartość dokładności. Szansa na to, że urządzenie znajduje się w środku tego koła wynosi 68 procent. Wiemy również, że jest to wyłącznie dokładność horyzontalna. Na końcu podana jest informacja, że jeśli lokalizacja nie posiada dokładności, to zwracana jest wartość 0.0. To jest właśnie przyczyna błędu. Używana metoda zwraca bowiem wartość 0.0 przy próbie określenia lokalizacji bez dostatecznej ilości satelitów. To doprowadzało do sytuacji, że przy próbie uruchomienia aplikacji w pomieszczeniu, zwracana była wartość 0.0, co jest mniejsze niż 25. Tym samym uruchomienie biegu było możliwe. Ten błąd został jednak zauważony i po zmianie dodany jest warunek, że wartość dokładności musi znajdować się w środku określonego przedziału, pomiędzy 0 a wartością 25. Innym potencjalnym miejscem błędu jest moment ustawiania pola tekstowego, z informacją o dokładności wskazania modułu GPS. Metoda *makeText* klasy *Toast* przygotowuje jedynie pojawiające się pole z informacją, która ma wyświetlić się użytkownikowi. Jeśli nie zostanie dodane wywołanie metody *show*, informacja nigdy nie pojawi się na ekranie.

Oczywista nie jest również funkcja, która wyświetla informację o połączeniu z modułem GPS. Sprawdzanie jakości połączenia z tym modułem, musi odbywać się regularnie. Jednorazowe ustawienie informacji o prawidłowym połączeniu z danym modułem nie jest wystarczające. Wystarczy bowiem, że po chwili użytkownik wyłączy ten moduł. Dlatego istotne jest, aby sprawdzenie dokładności i ustawianie informacji o połączeniu odbywało się w oddzielnym wątku. Zarządzaniu wątkami poświęcono jeden z poprzednich rozdziałów pracy. Nie można z dowolnego wątku modyfikować graficznego interfejsu użytkownika. Zmiany te można wykonać tylko w wątku interfejsu użytkownika. Służy do tego specjalna metoda *runOnUiThread* z głównej klasy *Activity*. Przyjmuje ona jako parametr klasę implementującą interfejs *Runnable* lub wątek. Oba rozwiązania wymagają od nas implementacji bezargumentowej metody *run*, która wykonuje określoną

operację. Tym sposobem można z dowolnego wątku modyfikować interfejs użytkownika. Funkcja sprawdzająca jakość połączenia z modułem GPS oraz ustawiająca informację o połączeniu wygląda następująco:

```
private boolean accuracyValid() {
  final boolean valid = accuracy < 25 && accuracy > 0;
  runOnUiThread(new Runnable() {
      @Override
           public void run() {
               if (valid)
                   gpsInfo.setText("GPS OK");
               else
                   gpsInfo.setText("GPS BAD");
           }
       });
  return valid;
}
```

Wykorzystując ten mechanizm, można w dowolny sposób modyfikować interfejs użytkownika. Bardzo ważne przy programowaniu aplikacji na platformę Android jest oddzielenie wątków, które mają prawo modyfikować interfejs użytkownika, od tych które nie mogą tego robić. Dzięki temu można uniknąć zablokowania ekranu telefonu, jeśli aplikacja zapętliłaby się w określonym wątku.

Oprócz pobierania informacji o wskazaniach modułu GPS należy również sprawdzić połączenie z internetem. Cały proces przebiega analogicznie do kontrolowania połączenia z modułem GPS. Wykorzystano tutaj klasę *NetworkInfo*, która informuje, czy urządzenie jest połączone lub czy łączy się z internetem. Implementacja przedstawia się następująco:

```
ConnectivityManager cm =
   (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo netInfo = cm.getActiveNetworkInfo();
final boolean online = netInfo != null &&
   netInfo.isConnectedOrConnecting();
```

Jak widać najpierw tworzony jest obiekt klasy *ConnectivityManager*. Tworzenie wykorzystuje metodę *getSystemService*, z klasy *Activity*. Pozwala ona pobrać odpowiednie informacje o wybranych przez nas usługach. Aby korzystać z tej funkcjonalności, wymagane jest dodanie odpowiedniego pola w pliku *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Jeśli wartość ta nie zostanie ustawiona, aplikacja nie będzie miała dostępu do usługi odpowiedzialnej za zarządzanie informacją o połączeniu z internetem. Wywołanie metody w aplikacji spowoduje wystąpienie odpowiedniego wyjątku. Wydawałoby się, że implementacja opisanej funkcjonalności nie wymaga zbyt wielu linijek kodu. Jednak nawet przy tak prostych operacjach, trzeba być świadomym z jakich funkcji się korzysta i czego te funkcje wymagają od programisty.

6.3.2 Bieg z obcym i ze znajomym

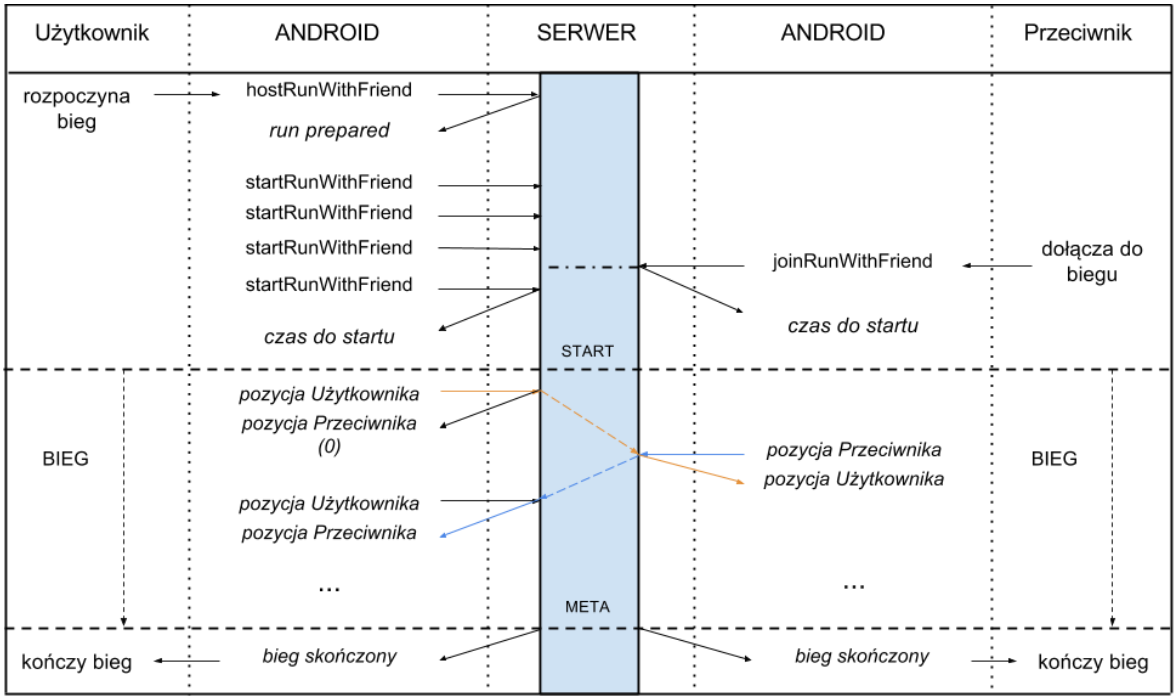
W tej pracy przedstawiono już implementację podstawowych funkcjonalności, które pozwalają zarządzać aplikacją. Pokazano również klasy, odpowiedzialne za integrację z serwerem i wysyłanie odpowiednich żądań. W tym rozdziale opisano w jaki sposób zaimplementowany jest bieg z losową osobą oraz bieg ze znajomym.

Aby rozpocząć bieg ze znajomym należy w aktywności *EnemyPickerActivity* wybrać przycisk *RUN WITH RANDOM*. Pojawiają się wtedy dwa pola. Jedno z nich nazwane jest *Desired Distance*, natomiast drugie *Acceptable Distance*. W pierwszym polu wymagane jest podanie dystansu, który użytkownik chciałby pokonać, natomiast w drugim polu podawany jest dystans graniczny, który jest akceptowany. Przeciwnik wypełnia pola w ten sam sposób. Dzięki temu wylosowany wspólny dystans będzie najbardziej pasował obu użytkownikom. W aplikacji wprowadzono ograniczenie minimalnego dystansu, którego wartość to 500 metrów. Wybrana została taka wartość, ponieważ przy biegach na małe dystanse, niedokładność pomiaru GPS mogłaby mieć zbyt duży wpływ na rezultat biegu. Po wybraniu dystansu, pojawia się przycisk, który umożliwia rywalizację, jeśli spełnione są wszystkie warunki. Te warunki to:

- dobre połączenie z modułem GPS,
- połączenie z internetem

- oraz dodatkowe warunki nałożone na podawany dystans.

Następnie wysyłane jest żądanie do serwera, a otrzymana wartość odpowiednio interpretowana. Jeśli zwrócona wartość to całkowita liczba dodatnia, rozpoczyna się odliczanie do rozpoczęcia biegu. Następnie podczas biegu, co stały czas, wysyłane jest żądanie do serwera, w którym podajemy nasz dotychczasowy rezultat, a w odpowiedzi otrzymujemy pozycję przeciwnika. Szczegóły zarządzania biegiem oraz zakończenia biegu opisano w dalszym rozdziale. Jeżeli użytkownik nie chce biec z obcą osobą ma inną możliwość, czyli bieg ze znajomym. W tym przypadku implementacja jest bardziej skomplikowana. Po wybraniu przycisku *RUN WITH FRIEND* użytkownik ma do wyboru dwa przyciski. Jeden z nich to przycisk *HOST*, a drugi przycisk *JOIN*. Podczas rywalizacji ze znajomym rozpoczęcie biegu jest bardziej problematyczne, ponieważ serwer nie może nam dobrać losowego przeciwnika.



Rysunek 7 Schemat komunikacji pomiędzy aplikacją użytkownika, serwerem a aplikacją przeciwnika.

W projekcie musimy znaleźć sposób, aby umożliwić rywalizację z konkretną osobą. Schemat komunikacji zaprezentowany jest na rysunku nr 7. Aby rozróżnić uczestników rywalizacji użyto określenia Użytkownik, dla osoby która tworzy grę. Przeciwnik to osoba, która dołącza do gry. Do utworzenia biegu wybrany został scenariusz

podobny do gier rozgrywanych w internecie, w czasie rzeczywistym. Opiera się ono na prostej zasadzie, że jedna osoba tworzy grę (potocznie „hostuje”, metoda *hostRunWithFriend*), a druga osoba dołączą do utworzonej gry. Użytkownik tworzący grę, musi podać login osoby, z którą chce rywalizować. Do tego, podobnie jak przy biegu z losową osobą, podaje się wymagany i akceptowalny dystans. Na serwerze tworzy się wpis, który oczekuje na dołączenie przeciwnika. Użytkownik dostaje natomiast informację zwrotną o udanym przygotowaniu biegu *(run prepared).* Po stworzeniu biegu, znajomy użytkownika musi wybrać parametry biegu, które mają z parametrami użytkownika część wspólną. Jeśli następnie rozpocznie bieg (*joinRunWithFriend),* serwer dopasuje obu uczestników biegu. Przeciwnikowi, który dołączał do biegu, serwer zwróci dodatnią wartość całkowitą, która oznaczać będzie czas do rozpoczęcia biegu. Jeżeli zaś chodzi o osobę tworzącą bieg, musi ona sprawdzić, czy przeciwnik dołączył do rywalizacji (wielokrotne wywołanie *startRunWithFriend* na schemacie komunikacji). W tym celu wykonywane są regularne zapytania do serwera, co około sekundę, aby otrzymać informację o dołączeniu przeciwnika do biegu. Jeśli przeciwnik dołączył do biegu, zwracana jest liczba sekund, określająca za ile bieg ma się rozpocząć. Oczywiście jest ona odpowiednio pomniejszona o czas, który minął od otrzymania odpowiedzi o dołączeniu przeciwnika do rywalizacji. Dzieje się tak aby obaj użytkownicy rozpoczęli bieg w tym samym momencie. Następnie każdy uczestnik wysyła swoje wyniki i odbiera pozycję przeciwnika. W momencie kiedy bieg jest skończony, serwer zwraca odpowiednie wartości, informując jednocześnie, który z biegaczy zwyciężył.

Opisana wyżej kolejność wysyłanych żądań pozwala w najprostszy sposób rozpocząć rywalizację ze znajomym. Istotne jest bowiem, aby bieg był zsynchronizowany pomiędzy uczestnikami. Modyfikacja, którą można by wprowadzić, to użycie Google Cloud Messaging, aby informować osobę która tworzy bieg, o dołączeniu do rywalizacji jej przeciwnika. Mechanizm ten pozwala wysłać wiadomość z serwera do aplikacji. W tym momencie jedynie aplikacja może wysyłać żądania do serwera Google App Engine. Przy użyciu GCM, aplikacja mobilna nie musiałaby wysyłać kilku żądań z zapytaniem, czy przeciwnik dołączył do biegu. Jednak biorąc pod uwagę ilość żądań, wysyłanych podczas biegu, jest to pomijalna wartość. Tutaj wysyłane jest od kilku do kilkunastu żądań, natomiast podczas biegu ta liczba znacząco rośnie. Bieg może trwać długo, a podczas biegu co stały

czas przekazujemy naszą pozycję do serwera. Zastosowane rozwiązanie upraszcza też aplikację, ponieważ implementacja modułu GCM jest skomplikowana zarówno po stronie serwera jak i stronie aplikacji mobilnej. Poniżej jedna z ważniejszych funkcji, która „odpytuje” serwer przed rozpoczęciem biegu:

```
TimerTask task = new TimerTask() {
    int requestCounter;
    @Override
    public void run() {
        try {
            RunStartInfo result =
                theIntegrationLayer.startRunWithFriend(preferences);
            requestCounter++;
            runOnUiThread(() -> Toast.makeText(getApplicationContext(),
                "Looking for friend", Toast.LENGTH_SHORT).show());
                if (result.getTime() > 0) {
                   startRun(result);
                   this.cancel();
                 }
                 else if (requestCounter == 10) {
                    runOnUiThread(() -> enemyNotFound());
                    this.cancel();
                  }
                 } catch (IOException e) {
                     e.printStackTrace();
                 }
             }
        };
        Timer timer = new Timer();
        timer.scheduleAtFixedRate(task, 0, 4000);
    }
```

W powyższym fragmencie kodu przedstawione jest także użycie klasy *TimeTask*, aby co określony czas wysłać żądanie do serwera. Metoda *scheduleAtFixedRate* pozwala określić opóźnienie, z jakim wysłane będzie pierwsze zapytanie, a także przedział czasowy, co ile ma być wykonywane zapytanie. Jest to bardziej eleganckie rozwiązanie niż wątek, do tego w prosty sposób możemy zatrzymać wykonywanie zapytań korzystając z metody *cancel*. Wysyłanie zapytań zatrzymywane jest w dwóch przypadkach. Jeden z nich to otrzymanie informacji o dołączeniu przeciwnika do biegu. Drugi przypadek, to przekroczenie ilości wykonanych zapytań. Nie chcemy bowiem aby aplikacja wysyłała żądania bez końca.

6.3.3 Zarządzanie biegiem

W rozdziale tym opisana jest implementacja biegu oraz komunikacja pomiędzy aktywnością wyświetlającą rezultat biegu, a klasami odpowiedzialnymi za pobieranie informacji o pozycjach użytkowników. Rozdział podzielony jest na odpowiednie podrozdziały.

6.3.3.1 Komunikacja pomiędzy aktywnościami

Po wyborze przeciwnika uruchamiana jest aktywność, która wyświetla użytkownikowi rezultat biegu. Interfejs nie wymaga od użytkownika jakiegokolwiek działania

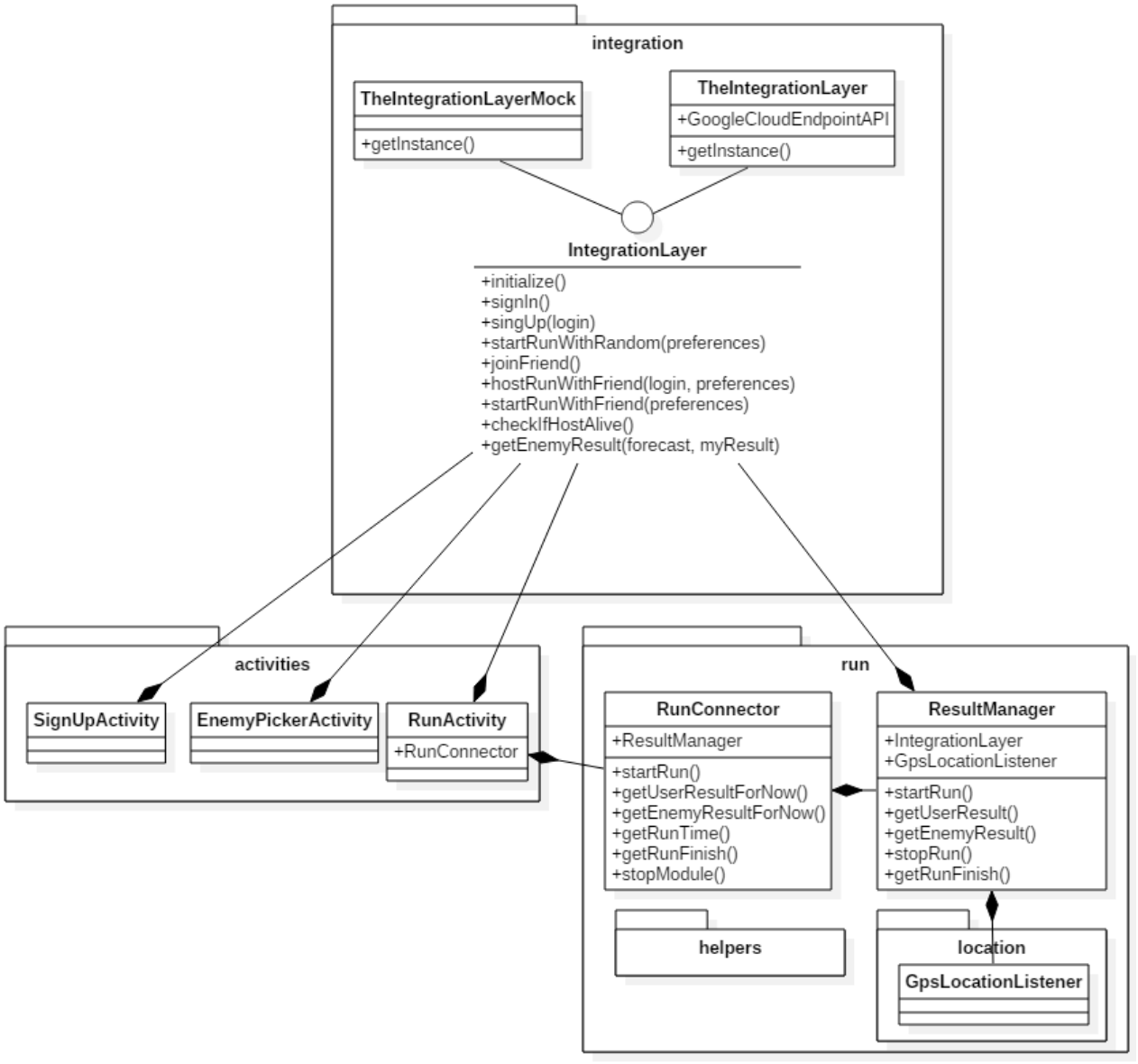
Rysunek 8 Ekran biegu

Po zestawieniu biegu, użytkownik widzi ekran biegu, na którym rozpoczyna się odliczanie do startu. Na samej górze wyświetla się dystans, który został wybrany przez algorytmy serwera, zgodnie z preferencjami użytkowników. W momencie rozpoczęcia biegu, pole *You* oraz *Enemy* wyświetla dystans użytkownika oraz dystans jego przeciwnika. Czas biegu jest wspólny dla obu uczestników. Po zakończeniu biegu, na dole ekranu, wyświetla się informacja o rezultacie biegu. To serwer informuje użytkowników o tym, kto zwyciężył podczas biegu.

Graficzny interfejs użytkownika jest nieskomplikowany. Zarządzanie biegiem nie jest już takie proste. W pierwszej kolejności istotne jest aby umożliwić użytkownikowi bieg, nawet jeśli aktywność działa w tle. Z tego powodu większość operacji zarządzających biegiem działa jako serwisy. Przy projektowaniu tej funkcjonalności starałem się oddzielić interfejs użytkownika od logiki aplikacji. W projekcie widać to poprzez podział klas na odpowiednie pakiety. Ogólna struktura wygląda następująco:

```
root
| activities
| integration
| model
| run
| | helpers
| | location
```

Można to również przedstawić na następujący schemacie:



Rysunek 9 Schemat podziału aplikacji

W pakiecie *activities* znajdują się między innymi aktywności *EnemyPickerActivity* oraz *RunActivity*. W pakiecie *run* znajduje się klasa *ResultManager*, która jest silnikiem zarządzającym biegiem. W tym samym pakiecie znajduje się również klasa *RunConnector* oraz pakiet *location*. Klasa *RunConnector* jest łącznikiem pomiędzy interfejsem użytkownika, a logiką aplikacji. Udostępnia ona metody dla aktywności. Dodatkowo jest odpowiedzialna za połączenie z klasą *ResultManager*, która jest serwisem. Przy takiej

implementacji, aktywność korzysta z funkcji zarządzających biegiem, jakby były one zwykłą klasą. Nie musi zajmować się uruchamianiem serwisu, sprawdzaniem czy jest on połączony czy zatrzymywaniem serwisu. Dzięki temu kod pisze się łatwiej, a klasa *RunActivity* nie jest aż tak obszerna. Takie rozwiązanie pozwoliło mi skupić się na konkretnych zadaniach klas, zachowując porządek w pisanym kodzie. Komunikację pomiędzy klasami można schematycznie przedstawić w inny sposób:

```
RunActivity <---> RunConnector <---> ResultManager <---> klasy pomocnicze
```

Klasa *ResultManager* korzysta ze wszystkich potrzebych klas pomocniczych, aby prawidłowo zarządzać biegiem. Interfejs programistyczny klasy *RunConnector* pozostaje bez zmian, więc nawet znaczące modyfikacje w klasie *ResultManager*, nie mają wpływu na zachowanie interfejsu graficznego oraz zaimplementowaną w nim funkcjonalność.

Jak wcześniej wspomniano, sercem aplikacji jest klasa *ResultManager*. Klasa ta dziedziczy po klasie *Service*. Mamy więc pewność, że wszystkie operacje będą wykonywały się nawet gdy aplikacja będzie działać w tle. W momencie kiedy aktywność rozpoczyna odliczanie do startu, wysyła też informację poprzez *RunConnector* do serwisu. Tworzone jest wtedy połączenie z serwisem, który jest przygotowany do działania. W momencie kiedy kończy się odliczanie, rozpoczyna się bieg. Wtedy do serwisu wysyłana jest informacja, o rozpoczęciu biegu. Należy tutaj zaznaczyć, że wszystkie dane związane z biegiem, które wyświetla aktywność, pobierane są z serwisu. Jest za to odpowiedzialna metoda *updateMainInformation*, która działa w jednym z wątków aktywności. Metoda ta wygląda następująco:

```
private void updateMainInformation() {
     runFinish = runConnector.getRunFinish();
     runTime = runConnector.getRunTime();
     userResultForNow = runConnector.getUserResultForNow();
     enemyResultForNow = runConnector.getEnemyResultForNow();
}
```

W tym momencie nie są istotne typy zwracane przez poszczególne metody. Należy podkreślić, że rezultat użytkownika, rezultat przeciwnika, czas biegu oraz informacje o zakończeniu biegu są pobierane z klasy *ResultManager*, poprzez wykorzystanie klasy

RunConnector. Aktywność nie wylicza dystansu, nie przybliża wyników, nie uruchamia też zegara odliczającego czas biegu. Wszystkie te dane pobierane są z serwisu. Jeżeli aktywność widzi, że użytkownik przekroczył metę, nie wyświetla żadnej informacji, dopóki metoda *getRunFinish* z klasy *RunConnector* nie zwróci takiej informacji.

6.3.3.2 Rozpoczęcie biegu

Jak wcześniej zaznaczono w momencie zakończenia odliczania przez aktywność, serwis rozpoczyna swoje działanie. Można powiedzieć że uruchamia wtedy wszystkie silniki. Wywoływana jest wtedy funkcja *startRun*, która wygląda następująco:

```
public void startRun() {
    theRunTimer.start();
    runThread = new RunThread();
    threadHandler.post(runThread);
    new GetEnemyResult().execute();
}
```

Widoczne są tutaj trzy istotne moduły, z których każdy jest pośrednio od siebie zależny. Po pierwsze konieczne jest rozpoczęcie odliczanie czasu, za co odpowiedzialna jest klasa *TheRunTimer* oraz obiekt *theRunTimer*. Zastosowano tu podobne nazewnictwo jak przy implementacji warstwy integracji z serwerem. Przedrostek „The” informuje, że klasa jest zaimplementowana zgodnie z wzorcem singleton. Wykorzystano ten wzorzec, aby każda klasa korzystała z tego samego zegara odliczającego czas biegu. W metodzie *startRun* wywoływane są również funkcje odpowiedzialne za rezultat przeciwnika oraz użytkownika. Rezultat użytkownika aktualizowany jest w wątku *RunThread*, natomiast rezultat przeciwnika aktualizowany jest w asynchronicznym zadaniu *GetEnemyResult*. Od tego momentu bieg jest rozpoczęty, a aktywność *RunActivity* wyświetla rezultaty.

6.3.3.3 Otrzymywanie rezultatów biegaczy

W aplikacji istnieją dwa moduły, które są odpowiedzialne za pobieranie rezultatów uczestników biegu. Rezultaty przechowywane są w kolekacjach typu *List*, odpowiednio *userResults* dla użytkownika oraz *enemyResults* dla przeciwnika. Każdy moduł wypełnia odpowiednią listę danymi typu *RunResult*. Obiekt klasy *RunResult* zawiera dwa pola. Jedno z nich to całkowity przebyty dystans w metrach, przechowywany w zmiennoprzecinkowym

typie *float.* Drugie pole reprezentuje całkowity czas biegu, wyrażony w milisekundach. Wartość przechowywana jest jako typ całkowity.

Za pobieranie rezultatu użytkownika odpowiedzialny jest wątek *RunThread,* z klasy *ResultManager*. Na początku pobierana jest nowa lokalizacja. Wykorzystana jest do tego klasa implementująca interfejs *LocationListener.* Przed wyliczeniem nowego dystansu sprawdzane jest, czy nowa lokalizacja jest wystarczająco dokładna (metoda *isAccuracte)* oraz czy jest w odpowiedniej odległości od poprzednio pobranej pozycji (metoda *isFarEnough*). W kodzie aplikacji wygląda to następująco:

```
Location newLocation = gpsLocationListener.getCurrentLocation();
if (newLocation != null && isAccurate(newLocation) &&
   isFarEnough(newLocation))
```

Jeśli lokalizacja nie jest dokładna, do kolekcji z wynikami użytkownika wstawiany jest poprzedni dystans, a czas biegu pobierany jest z klasy *TheRunTimer.* W przeciwnym przypadku, wyliczany jest całkowity dystans przebyty przez użytkownika na podstawie nowej pozycji oraz poprzedniej lokalizacji. Wartość ta wstawiana jest na początek listy z wynikami użytkownika. Pozostała implementacja wątku, odpowiedzialnego za pobieranie pozycji użytkownika przedstawia się następująco:

```
    if (userLocation != null)
        userDistance += userLocation.distanceTo(newLocation);
    userLocation = newLocation;
    userResults.add(HEAD,
      new RunResult(userDistance, theRunTimer.getRunDurationMillis()));
    float edge =
        (distanceToRun - userDistance) / (float) distanceToRun;
    if (edge < 0.05) {
        lowerLocationUpdateDistance(0.33f);
    }
} else {
    userResults.add(HEAD,
        new RunResult(
            userDistance, theRunTimer.getRunDurationMillis()));
```

```
    }
if (distanceToRun - userDistance <= 0 || runFinish.isRunOver()) {
    gpsLocationListener.stop();
    return;
}
threadHandler.postDelayed(this, 1000);
```

Istotna jest tutaj funkcja *lowerLocationUpdateDistance.* Jest ona odpowiedzialna za zmniejszenie granicznej wartości dystansu, po przebyciu której akceptujemy nową lokalizację. Pod koniec wątku *RunThread* znajduje się również fragment kodu odpowiedzialny za zakończenie biegu co wyjaśnione zostanie w dalszej części pracy.

Pozycja przeciwnika pozyskiwana jest oczywiście w ten sam sposób. Jednak od strony aplikacji mobilnej, wygląda to inaczej. Wewnątrz klasy *RunManager* zdefiniowana jest klasa *GetEnemyResult.* Dziedziczy ona po klasie *AsyncTask.* Jest ona odpowiedzialna za pobieranie informacji o pozycji przeciwnika oraz za wysyłanie pozycji użytkownika na serwer. Pozycję przeciwnika pobraną z serwera dodaje ona do listy *enemyResults.* Aby pobrać informację o pozycji przeciwnika musimy także przesłać naszą pozycję na serwer. Aby serwer poprawnie przetworzył dane, muszą być one różne od poprzednio wysłanych danych. Jeśli wielokrotnie przesłane będą te same dane, zostanie zwrócony błąd serwera. Dlatego przed pobieraniem pozycji przeciwnika, sprawdzane jest, czy pozycja użytkownika zmieniła się. Pozostała implementacja związana jest przetwarzaniem rezultatu przeciwnika oraz zakończeniem biegu co wyjaśnione zostanie w dalszej części pracy.

6.3.3.4 Zwracanie rezultatów

Jak opisano we wcześniejszych rozdziałach, aktywność *RunActivity* odpowiedzialna jest jedynie za wyświetlanie informacji o biegu. Wszelkie obliczenia wykonywane są w klasie *ResultManager.* Pozycja przeciwnika pobierana jest z wyprzedzeniem. Dlatego w momencie kiedy aktywność chce wyświetlić rezultat przeciwnika, wywoływana jest funkcja *getEnemyResult.* Podobnie w przypadku użytkownika, to aktywność decyduje kiedy chce wyświetlić pozycję biegacza. W związku z tym, w większości przypadków konieczne jest wyliczanie pozycji na podstawie pobranych danych. Należy rozpatrzyć kilka przypadków. Po pierwsze można nie mieć żadnych danych o pozycji biegacza. Zwracany jest wtedy dystans równy zero oraz czas równy czasowi

trwania biegu. W innych przypadkach, wyliczana jest pozycja na podstawie dwóch ostatnich rezultatów. Tutaj również należy wziąć pod uwagę dwa przypadki. Czas biegu może plasować się pomiędzy dwoma najnowszymi czasami lub po tych czasach. Jeśli czas biegu znajduje się pomiędzy czasami, wyliczana jest średnią prędkość na danym odcinku, co pozwala obliczyć dodatkowy dystans po danym czasie. W drugim przypadku algorytm jest bardzo podobny, z wyjątkiem zmiennych, które wykorzystane są do obliczenia średniej prędkości. Implementacja dla bardziej skomplikowanego przypadku, czyli pobierania pozycji przeciwnika przedstawia się następująco:

```
public RunResult getEnemyResult() {
        int resultsSize = enemyResults.size();
        if (resultsSize == 0)
            return new RunResult(0,
theRunTimer.getRunDurationMillis());
        else if (resultsSize > 0) {
            RunResult latestResult = enemyResults.get(HEAD);
            RunResult nextResult;
            if (enemyResults.size() == 1)
                nextResult = new RunResult(0, 0);
            else
                nextResult = enemyResults.get(1);
            float s1 = nextResult.getTotalDistance();
            float s2 = latestResult.getTotalDistance();
            long t1 = nextResult.getTotalTimeMillis();
            long t2 = latestResult.getTotalTimeMillis();
            long currentRunTime =
theRunTimer.getRunDurationMillis();
            float distanceForTheMoment;
            if (currentRunTime < t2) {
                float deltaDistance =
                  ((s2 - s1)/(t2 - t1))*(currentRunTime - t1);
               distanceForTheMoment = s1 + deltaDistance;
            } else {
                float deltaDistance =
```

```
                    ((s2 - s1)/(t2 - t1))*(currentRunTime - t2);
                distanceForTheMoment = s2 + deltaDistance;
            }
            if (distanceForTheMoment >= distanceToRun) {
                return new RunResult(distanceToRun, currentRunTime);
            }
            distanceForTheMoment = roundUp((int)
distanceForTheMoment);
            return new RunResult(distanceForTheMoment,
currentRunTime);
        }
        throw new RuntimeException();
    }
```

W momencie gdy wyliczony dystans jest większy niż dystans do pokonania, zwracany jest dystans do pokonania. Oznacza to jedynie, że bieg jest bliski ukończeniu, a odpowiednia informacja od serwera nie została jeszcze zwrócona. Na końcu funkcji rzucany jest wyjątek *RuntimeException,* ponieważ zgodnie z obecnymi założeniami funkcja nie powinna dojść do tego fragmentu kodu.

6.3.3.5 Zakończenie biegu

O zakończeniu biegu informuje serwer. Jeżeli przy pobieraniu pozycji przeciwnika zwrócona zostanie wartość -1, oznacza to że bieg się zakończył. Ustawiania jest wtedy informacja w klasie *RunFinish,* o ukończeniu biegu. Serwer zwraca również drugą wartość, która informuje użytkowników o zwycięstwie lub przegranej. Całą funkcjonalność przedstawia następujący fragment kodu:

```
RunResultPiece enemyResultPiece =
    theIntegrationLayer.getEnemyResult(FORECAST_SECONDS,
userResult);
    if (enemyResultPiece != null) {
      Integer enemyDistance = enemyResultPiece.getDistance();
       if (enemyDistance > 0)
          enemyResults.add(HEAD, new RunResult
                (enemyDistance, enemyResultPiece.getTime() * 1000));
```

```
        else if (enemyDistance == -1 && enemyResultPiece.getTime()
>= 0){
           runFinish.setRunOver(true);
             if (enemyResultPiece.getTime() == 0) {
               enemyResults.add(HEAD, new RunResult(
                     distanceToRun,
theRunTimer.getRunDurationMillis()));
                runFinish.setUserWon(false);
             }
             else if (enemyResultPiece.getTime() == 1) {
                runFinish.setUserWon(true);
             }
          }
```

Jeśli drugi parametr zwrócony przez serwer w klasie *RunResultPiece* jest równy 0, oznacza to że przeciwnik wygrał. Informacja ta jest zapisywana w logach aplikacji, dodatkowo ustawiany jest parametr *boolean* w obiekcie *runFinish*, który pozwoli aktywności wyświetlić informację o tym, kto zwyciężył. Tak jak wcześniej wspomniano, aktywność nie jest odpowiedzialna za logikę aplikacji. Jeśli zwrócona wartość jest równa 1, oznacza to że użytkownik wygrał.

Istotny jest jeszcze fragment kodu w wątku *RunThread.* Jeśli użytkownik przebiegnie zaplanowany dystans, ale serwer nie zwróci informacji o zwycięzcy, wątek zostaje naturalnie zatrzymany. Dodatkowo wyłączane jest pobieranie danych o lokalizacji. Poniższy fragment kodu pochodzi z klasy *RunThread:*

```
if (distanceToRun - userDistance <= 0 || runFinish.isRunOver()) {
    gpsLocationListener.stop();
    return;
}
```

Fragment ten zostanie wykonany również w przypadku, kiedy serwer zwrócił informację o zakończeniu biegu.

7 Testy i logowanie zdarzeń

7.1 Testy

Aby dowolne oprogramowanie mogło trafić w ręce użytkowników, powinno być odpowiednio przetestowane. Wiele firm informatycznych oprócz programistów zatrudnia również testerów. Są oni odpowiedzialni za napisanie przypadków testowych oraz ich przeprowadzenie. Przypadek testowy musi mieć zdefiniowany cel, warunki wstępne oraz kroki, na kroki testu składa się opis kroku oraz opis oczekiwanego rezultatu. Często tworzone są również raporty z przeprowadzonych testów oraz wielopoziomowe plany testów, na które składa się bardzo wiele różnych przypadków testowych.

Testy możemy podzielić na kilka rodzajów. Najbardziej podstawowe to testy jednostkowe oraz integracyjne. W przypadku testów jednostkowych, testowana jest wyłącznie dana funkcjonalność. Przykładowo w aplikacji mobilnej istnieje klasa, która odpowiedzialna jest za przewidywanie pozycji użytkownika, na podstawie jego dotychczasowych wyników. Można taką funkcję wydzielić do odpowiedniej klasy pomocniczej. Następnie dla tej metody, powinniśmy zaimplementować odpowiednią klasę testującą. Testy powinny sprawdzać jak najwięcej przypadków. Istotne jest również sprawdzenie przypadków brzegowych, ponieważ zwykle wtedy programiści popełniają najwięcej błędów. Są one także najtrudniejsze do obsługi. Po napisaniu odpowiednich przypadków testowych możemy zająć się implementacją funkcjonalności. Jeśli wszystkie testy przechodzą pozytywnie, możemy być pewni, że dana funkcjonalność działa poprawnie w wybranym obszarze. Jeśli w aplikacji pojawi się błąd, z łatwością możemy wykluczyć miejsce jego wystąpienia. Warto dodać, że testy jednostkowe są najczęściej uruchamiane podczas budowania kolejnej wersji aplikacji czy też programu. Dzięki temu wyklucza się przekazanie wersji systemu z niedziałającymi podstawowymi funkcjami. Innym typem są testy integracyjne. Sprawdzają one czy wykorzystane moduły działają poprawnie i czy poprawnie się ze sobą komunikują. W prostych przypadkach test może sprawdzać czy w serwerze zostały zapisane dokładnie te dane, które wysłało urządzenie.

W przypadku omawianej aplikacji pisanie przypadków testowych nie jest takie proste. Problem wynika z tego, że większość scenariuszy testowych sprowadza się do użycia

danego modułu bezpośrednio na telefonie. Przy takich testach pomocne mogą być dedykowane narzędzia, takie jak *Robot Framework*, *Selenium* czy *Appium*.

Biorąc pod uwagę fakt, że strona mobilna implementowana była samodzielnie, zadecydowano że do testów nie zostaną użyte wyspecjalizowanych narzędzia. Za takim rozwiązaniem przemawiał również fakt, że każda aplikacja mobilna powinna być ostatecznie sprawdzona przez programistę ręcznie, co odpowiada testom manualnym, użytkownika czy też „User Experience". Aplikacja uruchomiona na wirtualnej maszynie na komputerze może zachowywać się inaczej, niż na telefonie. Dodatkowo jeśli chcemy dostosować wizualną stronę aplikacji, należy sprawdzić na telefonie jak wyglądają i zachowują się poszczególne aktywności.

W projekcie przetestowano stronę mobilną aplikacji. W praktyce testowano również odpowiednie zachowanie serwera. Aby móc rozwijać aplikację bez skończonej implementacji po stronie serwera, wykorzystano atrapy obiektów, czyli tzw. „mocki", do symulacji zachowania funkcji serwera. Klasa z której korzystano do integracji z serwerem to *TheIntegrationLayer*. Na początku tworzenia projektu, była to jedyna klasa, z której korzystano. Szybko okazało się, że aby móc pracować nad projektem, występuje potrzeba symulowania zachowania serwera. Wydzielono wtedy interfejs *IntgerationLayer*, który przedstawia się następująco:

```
public interface IntegrationLayer {

    TheIntegrationLayer initialize(GoogleAccountCredential
credential);

    Profile signIn() throws IOException;

    Profile signUp(String login) throws IOException;

    boolean deleteAccount() throws IOException;

    RunStartInfo startRunWithRandom(Preferences preferences) throws
IOException;

    RunStartInfo joinFriend(Preferences preferences) throws
IOException;

    boolean hostRunWithFriend(String login, Preferences preferences)
throws IOException;

    RunStartInfo startRunWithFriend(Preferences preferences) throws
IOException;

    boolean checkIfHostAlive() throws IOException;
```

```
    RunResultPiece getEnemyResult(int forecast, RunResult myResult)
throws IOException;

}
```

Interfejs ten przedstawia wszystkie funkcje wykorzystane w aplikacji do komunikacji z serwerem. Schematycznie jest to zaprezentowane na rysunku *„Schemat podziału aplikacji”*. Stworzyłem dwie klasy, w tym samym pakiecie *integration.* Jedna z nich to właściwa klasa odpowiedzialna za komunikację z serwerem, korzystająca z chmury klasa *TheIntegrationLayer*. Druga klasa została nazwana przeze mnie *TheIntegrationLayerMock.* Implementuje ona wszystkie metody interfejsu *IntegrationLayer*. Jednak ciała metod zawierają stałe wartości. Pomaga to również w przypadku, kiedy wystąpi błąd w aplikacji lub błąd po stronie serwera. Tester jest bowiem w stanie odtworzyć scenariusz, który wywołał błąd. Do tego możliwa jest kontrola nad różnymi scenariuszami testowymi. Podczas testowego biegu można manipulować wynikiem przeciwnika, aby sprawdzić działanie aplikacji, w sytuacji kiedy przeciwnik wygrywa oraz kiedy przegrywa. Wystarczy zmiana prędkość przeciwnika. Test idealnie sprawdza zarówno poprawne zachowanie aplikacji mobilnej jak i poprawną funkcjonalność serwera. Wymienione testy, pozwoliły wyeliminować również błędy po stronie serwera. Implementacja klasy z symulowaną funkcjonalnością wygląda następująco:

```
public class TheIntegrationLayerMock implements IntegrationLayer {
    public static final String TEST = "TEST";
    public static final int COUNTDOWN = 3;
    public static final int DISTANCE = 100;
    private int time = 0;
    private int distance = 10;
    private TheIntegrationLayerMock() {}
    private static class TheIntgerationLayerHolder {
        private final static TheIntegrationLayerMock instance = new
TheIntegrationLayerMock();
    }
    public static TheIntegrationLayerMock getInstance() {
        return TheIntgerationLayerHolder.instance;
    }
   @Override
```

```
   public TheIntegrationLayer initialize(GoogleAccountCredential
credential) {
        return null;
    }
    @Override
    public Profile signIn() throws IOException {
        return new Profile().setLogin(TEST);
    }
    @Override
    public Profile signUp(String login) throws IOException {
        return new Profile().setLogin(TEST);
    }
    @Override
    public boolean deleteAccount() throws IOException {
        return true;
    }
    @Override
    public RunStartInfo startRunWithRandom(Preferences preferences)
throws IOException {
        return new
RunStartInfo().setDistance(DISTANCE).setTime(COUNTDOWN);
    }
    @Override
    public RunStartInfo joinFriend(Preferences preferences) throws
IOException {
        return new
RunStartInfo().setDistance(DISTANCE).setTime(COUNTDOWN);
    }
    @Override
    public boolean hostRunWithFriend(String login, Preferences
preferences) throws IOException {
        return true;
    }
    @Override
    public RunStartInfo startRunWithFriend(Preferences preferences)
throws IOException {
```

```
        return new
RunStartInfo().setDistance(DISTANCE).setTime(COUNTDOWN);

    }

    @Override

    public boolean checkIfHostAlive() throws IOException {

        return true;

    }

    @Override

    public RunResultPiece getEnemyResult(int forecast, RunResult
myResult) throws IOException {

        distance += 1;

        time += 1;

        return new
RunResultPiece().setDistance(distance).setTime(time);

    }

}
```

Na samym początku zdefiniowane są pola, które później zwracane są w odpowiednich funkcjach. Takie rozwiązanie pozwala na szybką zmianę wartości, aby przetestować różne scenariusze testowe. Podobnie jak właściwa klasa, tutaj również zastosowano wzorzec *singleton*. W aktywnościach zdefiniowano odpowiednie pole korzystając z interfejsu *IntegrationLayer*. Pozwoliło to na sprawne podmienianie obiektów, w zależności od tego czy testowano aplikację, czy integrację aplikacji z serwerem. W ostatniej zamieszczonej metodzie *getEnemyResult*, manipuluje się wartościami dystansu i czasu. W połączeniu jest to nic innego jak prędkość przeciwnika.

Po stronie użytkownika również wymagane było symulowanie biegu. Aby w przypadku chęci sprawdzenia działania danego modułu, wykluczyć konieczność wychodzenia z budynku i symulowania biegu wykorzystano maszynę wirtualną Androida. Dzięki niej możliwe jest symulowanie wartości zwracanych przez moduł GPS. W ten sposób można sprawdzić czy funkcje obsługujące i integrujące moduł GPS w projekcie działają prawidłowo. W praktyce, jeśli istnieje potrzeba sprawdzenia jaką dokładność pomiaru lokalizacji zwracana jest przy konkretnej pogodzie, nie ma innej możliwości testy aplikacji po za budynkiem. Podczas tego typu testów udało się odnotować, że przy idealnej pogodzie, w środku miasta, aplikacja na telefonie Samsung Galaxy SIII jest w stanie pobierać

lokalizację z dokładnością do +/- 6 metrów na 100 metrów. Przy zachmurzeniu około 60, 70 procent, dokładność ta spada do około 30 metrów. Działanie modułu GPS testowano również w inny sposób. Jeden z testów przeprowadziłem na bieżni sportowej, o długości 100 metrów. Przy dobrej pogodzie okazało się, że pomiar był prawie idealny, z niedokładnością około 2 metrów. W testach wykorzystywano również krawężnik. Posłużył ku temu chodnik w Warszawie, którego krawężnik składał się z elementów o długości 1 m. Kilkakrotnie pokonano daną trasę i upewniono się, że implementacja działa prawidłowo, dodatkowo sprawdzono, że dokładność sięgała kilku metrów.

Testy integracyjne aplikacji przeprowadzane były na kilka sposobów. Tak jak wspomniano, wykorzystana została klasa *TheIntegrationLayerMock,* do testów oprogramowania bez wychodzenia z budynku. Po zintegrowaniu aplikacji z serwerem korzystano z właściwej klasy *TheIntegrationLayer*, a „mockowany" był dystans pokonywany przez użytkownika. Dodatkowo testowane było prawidłowe działanie interfejsu użytkownika. Scenariusze testowe były następujące:

1. Logowanie
    1.1. Zalogowanie do aplikacji po raz pierwszy (zarejestrowanie użytkownika)
    1.2. Zalogowanie do aplikacji po zarejestrowaniu
2. Aktywność *EnemyPickerActivity*
    2.1. Wskaźniki
        2.1.1. GPS
            2.1.1.1. Brak połączenia z modułem GPS
            2.1.1.2. Połączenie średniej jakości
            2.1.1.3. Połączenie dobrej jakości
        2.1.2. Internet
            2.1.2.1. Brak internetu
            2.1.2.2. Połączenie z siecią komórkową
            2.1.2.3. Połączenie z siecią WiFi
    2.2. Bieg z losową osobą
        2.2.1. Wyświetlenie pól do wpisania dystansu
            2.2.1.1. Wpisanie poprawnej wartości „Desired distance" (>= 500m)
            2.2.1.2. Wpisanie poprawnej wartości „Acceptable distance" (>= 500m)
            2.2.1.3. Wpisanie niepoprawnej wartości „Desired distance" (< 500m)
            2.2.1.4. Wpisanie niepoprawnej wartości „Acceptable distance" (< 500m)
            2.2.1.5. Wpisanie poprawnych wartości ale zbyt małej różnicy (< 100m)

2.2.1.6. Wpisanie poprawnych wartości oraz poprawnej różnicy (>= 100m)

2.3. Bieg ze znajomym

2.3.1. Wpisanie istniejącego loginu przeciwnika i utworzenie biegu na serwerze

2.3.2. Wpisanie nieistniejącego loginu przeciwnika i brak biegu na serwerze

2.4. Uruchomienie biegu

2.4.1. Umożliwienie wciśnięcia przycisku „Run” przy prawidłowych wskaźnikach i polach

2.4.2. Uniemożliwienie wciśnięcia przycisku „Run” przy nieprawidłowych wskaźnikach i polach

3. Aktywność *RunActivity*

3.1. Poprawne wyświetlanie dystansu do pokonania

3.2. Poprawne wyświetlanie czasu do startu

3.3. Poprawne wyświetlanie czasu biegu

3.4. Poprawne wyświetlanie pozycji użytkownika

3.5. Poprawne wyświetlanie pozycji przeciwnika

3.6. Poprawne wyświetlanie różnicy pomiędzy biegaczami

3.7. Poprawne wyświetlanie informacji o zakończeniu biegu

4. Pomiar biegu

4.1. Sprawdzenie czy aplikacja poprawnie mierzy bieg (wykorzystanie technik opisanych wyżej, w tym samym rozdziale)

Wszystkie opisane powyżej przypadki testowe zostały przeprowadzone na telefonach:

- Samsung Galaxy S3 z oprogramowaniem Android w wersji 4.4.4,
- LG G2 Mini z oprogramowaniem Android w wersji 5.0.1,
- Samsung Galaxy Note 2 z oprogramowaniem Android w wersji 4.4.4.

Wszystkie przypadki testowe, które nie wymagały połączenia z serwerem, testowane były lokalnie (przed komputerem). Scenariusze które wymagały połączenia z serwerem, testowane były z użyciem klasy *TheIntegrationLayerMock*, a także z użyciem prawdziwych funkcji serwera. Podczas testów integracyjnych, korzystających z funkcji serwera, realizowano kroki przypadków, a następnie sprawdzano czy otrzymane są poprawne wartość od serwera. Sprawdzono także, czy na serwerze zapisywane są poprawne wartości w bazie danych. Sprawdzenie zapisów stanowiło ostatni krok przypadków testowych, którego spełnienie pozwalało uznać przypadek za poprawny.

Aplikacja jest na tyle specyficzna, że dość często konieczne było wychodzenie z budynku, aby sprawdzić jak zachowuje się ona w rzeczywistych warunkach. Często sprowadzało się to do tego, że należało pobiec kilkaset metrów, następnie znajdując błąd wprowadzić poprawki w kodzie, wgrać nową wersję i ponownie wziąć udział w biegu. Błędy

po stronie serwera zgłaszano do Pawła Stępnia, a następnie oczekiwano na ich naprawienie i ponownie przeprowadzano przypadek testowy. Cały proces pozwolił stwierdzić, że aplikacja działa poprawnie zarówno po stronie aplikacji mobilnej jak i po stronie serwera.

7.3 Logowanie zdarzeń

Przy przeprowadzaniu testów pomocne jest również wykorzystanie mechanizmu logowania zdarzeń. Jest to niezastąpione rozwiązanie, podczas znajdowania błędów aplikacji na komputerze. W wielu funkcjach, które implementowano, dodano logowanie zdarzeń. Pozwala to na późniejsze obserwowanie zachowania aplikacji. Jeśli zdarzenia pojawiały się w takiej kolejności jak tego oczekiwano, oznacza to że aplikacja działa poprawnie. W przeciwnym przypadku można było znaleźć anomalie i wyeliminować błędy. W przypadku wystąpienia błędu można było również prześledzić zdarzenia, które miały miejsce przed wystąpieniem błędu. Dodatkowo logowanie zdarzeń jest bardzo pomocne w przypadku wielowątkowych aplikacji. Można zauważyć, który wątek otrzymał czas procesora wcześniej. Również pozwala to wyeliminować błędy w aplikacji.

Przy tworzeniu oprogramowania logowanie zdarzeń nie jest wyłącznie wykorzystywane do eliminacji błędów podczas tworzenia oprogramowania. Po wydaniu aplikacji, zdarzenia powinny być logowane i zapisywane do plików. Pozwala to później przeanalizować sytuację, w przypadku występowania błędów. Nie zawsze jest możliwość odtworzenia błędu. Często dany błąd pojawia się wyłącznie na określonej maszynie lub urządzeniu, na przykład w sytuacji wyczerpania określonych zasobów. Nawet jeśli programista popełnij błąd, można wprowadzić odpowiednie poprawki.

Logowanie przy użyciu klasy *Logger* z pakietu *java.util.logging* umożliwia zapisywanie zdarzeń na różnych poziomach. Mechanizm ten daje programiście kontrolę nad tym, w jaki sposób obsługiwane będzie dana informacja o zdarzeniu. Przykładowo podczas testów, można zapisywać bardzo szczegółowe zdarzenia na poziomie *debug*. Mniej szczegółowe zdarzenia można zapisywać na poziomie *info*. Później można zdecydować, że tylko zdarzenia z poziomu *info* oraz wyższe, czyli na przykład *error*, będą zapisywane do plików. Natomiast zdarzenie na poziomie *debug* wyświetlane będzie wyłącznie na konsolę. Rozwiązanie takie daje programiście większą elastyczność i dostosowanie odpowiedniego

rozwiązania do sytuacji. Rozróżnia się tu sytuacje, w których aplikacji jest rozwijana, od sytuacji kiedy aplikacja jest faktycznie w użyciu.

Odpowiednie przeprowadzenie testów oraz dodatkowe zapisywanie zdarzeń pozwala wyeliminować większość błędów w aplikacji. Często twórca nie jest w stanie pozbyć się wszystkich błędów. Istnieje jednak szansa, że użytkownicy aplikacji nie będą rozczarowani ilością problemów na swoich telefonach. W rzeczywistości często dopiero właściwe użycie systemu przez wielu użytkowników, pozwala wykryć głęboko ukryte błędy.

8 Podsumowanie

W przedstawionej pracy inżynierskiej opisano aplikację mobilną dedykowaną na system Android służącą do rywalizacji pomiędzy dwoma użytkownikami w czasie rzeczywistym w dowolnym miejscu na świecie.

Aplikacja ta wyróżnia się na rynku aplikacji mobilnych, ponieważ istniejące rozwiązania opierają się na innej idei, a sama aplikacja kreuje nowe doświadczenia użytkownika. Istniejące rozwiązania skupiają się na zmierzeniu kalorii oraz pokazaniu rezultat naszego biegu. Zaprojektowana aplikacja odsuwa to na drugi plan, a skupia się na zabawie poprzez rywalizację. Aplikacja posiada perspektywy na dalszy rozwój. Przede wszystkim istnieje możliwość rozbudowania algorytmu wykorzystywanego do pomiaru biegu oraz algorytmu przewidującego pozycję w sytuacjach wyjątkowych. Można rozbudować funkcjonalność o rywalizację z większą ilością osób czy turnieje wielosobowe.

Przy tworzeniu systemu wykorzystane zostało wiele gotowych rozwiązań, szczególnie do implementacji serwera. Silnik Google App Engine został wybrany ze względu na swoją skalowalność oraz łatwą komunikację z aplikacjami mobilnymi. Dodatkowo przy wyborze tego rozwiązania nie trzeba martwić się o skomplikowane techniki uwierzytelniania. Krótko podsumowując, implementacja strony serwera jest dość uniwersalna i opiera się na wykorzystaniu istniejących już narzędzi.

Kluczowe jest poprawne zaimplementowanie aplikacji mobilnej, bo to jest ta część projektu, z której korzysta użytkownik, natomiast serwer służy głównie przekazywaniu informacji pomiędzy biegaczami. Mobilna strona projektu musi poprawnie mierzyć bieg oraz robić to jak najmniejszym kosztem. Koncepcja projektu jest prosta, natomiast implementacja stawia określone wymagania.

Poświęcając czas na szczegółową analizę problemu, można wyciągnąć sporo wniosków i znaleźć unikatowe rozwiązania. System mierzenia biegu można zaimplementować w całkiem prosty sposób, to znaczy wykorzystując funkcje standardowej biblioteki systemu Android. Standardowe rozwiązania są przydatne przy tworzeniu interfejsu graficznego, komunikacji pomiędzy aktywnościami czy zarządzania serwisami. Główne moduły aplikacji powinno się jednak wielokrotnie przemyśleć pod kątem możliwości wprowadzenia optymalizacji i usprawnień.

Większość aplikacji wykorzystujących globalny system pozycjonowania, musi być odpowiednio zaprojektowana pod kątem wydajności. Biblioteka Androida udostępnia różne sposoby pobierania lokalizacji. Wybór zależy od rozwiązywanego problemu i wymagań aplikacji. W przypadku omawianego projektu, użytkownik oczekuje dokładnego pomiaru, jest więc to priorytet przy projektowaniu architektury aplikacji. Aplikacja jest wymagająca jeżeli chodzi o zużycie baterii, ponieważ wymaga stałego dostępu do internetu oraz stałego połączenia z modułem GPS. Zużycie to zależy jednak od dystansu, który jest do pokonania. Dodatkowo istnieje możliwość optymalizacji wykorzystania zasobów, poprzez algorytmy aproksymujące. Podczas biegu aplikacja może działać w tle, telefon nie zużywa wtedy energii na wyświetlanie danych. Rozwijając aplikację można zaimplementować opcję wyboru przez użytkownika, czy chce aby aplikacja działała dokładniej i rzadziej aktualizowała wyniki, czy decyduje się na większe zużycie zasobów zyskując na częstotliwości aktualizacji. Dodatkowo wykorzystując krokomierz, można rzadziej pobierać informację o pozycji użytkownika. Najważniejszy pozostaje jednak fakt, że ostatecznie pobierana jest rzeczywista, dokładna pozycję. Użytkownicy mają więc pewność, że ich ostateczny rezultat jest wyliczony przez system globalnego pozycjonowania z najlepszą możliwą dokładnością.

Tworząc tego typu aplikację powinno się używać metodyk zwinnych. Odpowiednie kroki, które powinno się zastosować są następujące. Najpierw należy określić wymagania aplikacji mobilnej oraz serwera aplikacji. Implementacja obu modułów mogłaby odbywać się niezależnie od siebie. Dane przekazywane przez aplikację mobilną do serwera, a zwracane przez serwer do aplikacji mobilnej, mogłyby być atrapami (ang. mock objects). Następnie po stronie mobilnej powinna być zaimplementowana możliwość pomiaru biegu, w jak najbardziej prymitywnej formie. Dane z modułu GPS mogłyby również być atrapami obiektów. Z takim zestawem funkcjonalności po stronie mobilnej, można zintegrować ją z serwerem aplikacji. Jeżeli serwer miałby zaimplementowaną wymianę danych, podstawa systemu byłaby utworzona. Dalej wymagany byłby rozwój mierzenia biegu wraz z odpowiednimi algorytmami. Kolejne funkcjonalności są kwestią wyboru, wymagane jest na pewno dopracowanie graficznego interfejsu użytkownika.

Podsumowując chciałbym zaznaczyć, że prosty pomysł nie zawsze musi okazać się tak samo łatwy w realizacji. Szczególnie jeśli nie zna się wszystkich dostępnych narzędzi.

Trzeba poświęcić dużo czasu, na odpowiednią analizę dostępnych rozwiązań oraz wybór tych najlepszych. Nie warto również niepotrzebnie tworzyć funkcjonalności, które ktoś już wcześniej zrealizował. Czasami jednak nie ma wyjścia i tworząc projekt trzeba wymyślać koło od nowa. Każde zadanie niesie jednak naukę na przyszłość. Wystarczy pomyśleć, że mając taki system, można w łatwy sposób stworzyć inną aplikację opartą na rywalizacji w czasie rzeczywistym. Możliwości jest dużo.

9 Bibliografia

1. *Hello, Android: Introducing Google's Mobile Development Platform Third Edition Edition, Ed Burnette, Jeff Friesen*
   Podstawy programowania na aplikację Android
   Pragmatic Programmers, LLC., 2010

2. *Effective Java, Joshua Bloch, 2nd Edition*
   Wykorzystanie współbieżności z użyciem *Executor Framework*
   Addison-Wesley, 2008

3. *Clean Code: A Handbook of Agile Software Craftsmanship, Robert C. Martin, Robert C. Martin Series, 1st Edition*
   Prawidłowy sposób implementacji funkcjonalności przy tworzeniu oprogramowania
   Pearson Education, Inc., 2009

4. *Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides*
   Wykorzystanie wzorców w programowaniu, wzorzec Singleton
   Addison-Wesley Longman Inc., 1998

5. *Inżynieria Oprogramowania, Krzysztof Sacha*
   Analiza wymagań, projektowanie aplikacji, testowanie oprogramowania
   Wydawnictwo Naukowe PWN SA, Warszawa 2010

6. *Android Application Testing Guide, Diego Torres Milano*
   Testowanie aplikacji na system Android
   Packt Publishing, 2011

7. *https://cloud.google.com/appengine/docs*
   Dokumentacja Google App Engine

8. *https://cloud.google.com/appengine/docs/java/endpoints/*
   Dokumentacja Google App Engine – endpoints

9. *https://cloud.google.com/appengine/docs/java/datastore/*
   Dokumentacja Google App Engine – datastore

10. *https://cloud.google.com/appengine/docs/java/memcache/*
Dokumentacja Google App Engine – memcache

11. *http://developer.android.com/guide/components/activities.html*
Dokumentacja Android – aktywności

12. *http://developer.android.com/guide/components/services.html*
Dokumentacja Android – serwisy

13. *https://developer.android.com/guide/topics/location/strategies.html*
Dokumentacja Android – strategie pobierania lokalizacji

14. *http://developer.android.com/intl/en/reference/android/location/LocationListener.html*
Dokumentacja Android – implementacja pobierania informacji o lokalizacji

15. *http://developer.android.com/reference/android/os/AsyncTask.html*
Dokumentacja Android – zadania asynchroniczne

16. *http://developer.android.com/intl/ru/guide/topics/manifest/manifest-intro.html*
Dokumentacja Android – użycie pliku *AndroidManifest.xml*

17. *http://developer.android.com/training/basics/activity-lifecycle/index.html*
Dokumentacja Android – cykl życia aplikacji

18. *https://cloud.google.com/appengine/docs/java/endpoints/gen_clients*
Generowanie biblioteki Google Cloud Endpoint na telefon z systemem Android

19. *https://cloud.google.com/appengine/docs/java/endpoints/consume_android*
Wykorzystanie Google Cloud Endpoints na telefonach z systemem Android

20. *https://cloud.google.com/appengine/docs/java/oauth/*
Informacje o wykorzystaniu protokołu OAuth 2.0 w aplikacjach opartych na serwerze Google App Engine

21. *http://www.physics.org/article-questions.asp?id=55*
Globalny System Pozycjonowania – trójstronność

22. *https://play.google.com/store/apps/details?id=com.endomondo.android*
Grafiki aktywności Endomondo

23. *https://play.google.com/store/apps/details?id=com.nike.plusgps*
Grafiki aktywności Nike+ Running

24. *RFC 6749 - The OAuth 2.0 Authorization Framework*
Autoryzacja przy użyciu protokołu OAuth 2.0

25. *https://tools.ietf.org/html/rfc6749*
Oficjalny dokument „The OAuth 2.0 Authorization Framework”

26. *https://cloud.google.com/appengine/docs/python/oauth/*
Schemat uwierzytelniania przy użyciu OAuth 2.0

27. *http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html*
Szczegóły działania globalnego systemu pozycjonowania

28. *https://pl.wikipedia.org/wiki/Global_Positioning_System*
Działanie globalnego systemu pozycjonowania

29. *http://developer.android.com/intl/ru/reference/android/media/MediaPlayer.html*
Uruchamianie dźwięku w aplikacjach na system Android

30. *http://www.mashery.com/blog/api-data-exchange-xml-vs-json*
Komunikacja z wykorzystaniem formatu *XML* oraz *JSON*

31. *Wikipedia Wolna encyklopedia, hasło: Uwierzytelnianie;*
https://pl.wikipedia.org/wiki/Uwierzytelnianie
Zasady, które muszą być spełnione, aby uwierzytelnianie odbyło się prawidłowo

32. *https://developer.android.com/about/versions/kitkat.html*
Informacje o sprzętowym wsparciu krokomierza w nowych telefonach oraz dostosowaniu oprogramowania